Minimizing Register 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 software pipelining has been recently proposed to optimize mono-phase clocked sequential circuits. The resulting circuits are multi-phase clocked sequential circuits, where all clocks have the same period. To preserve functionality of the original circuit, registers must be placed according to a correct schedule. This schedule also ensures the maximum throughput. In that method, it is question of (1) how to determine a schedule that requires the minimum number of registers, and (2) how to place these registers optimally. In this paper, problems (1) and (2) are tackled simultaneously. More precisely, we deal with the problem of determining schedules with the minimum register requirements, where the optimal register placement is done during the schedule determination. To optimally solve that problem, we provide a mixed integer linear program that we use to derive a linear program, which is polynomial-time solvable. Experimental results confirm the effectiveness of the approach, and show that significant reductions of the number of registers can be obtained.

1. Introduction

Software pipelining is a powerful technique for increasing the instruction-level parallelism for parallel processors. This method overlaps the execution of successive iterations. It has recently been used to develop a method for optimizing mono-phase clocked sequential circuits [2]. 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, according to that schedule. Finally, once the registers are placed, the phases are determined.

With this method, it is question of (1) how to determine a schedule that produces the minimum number of required registers, and (2) how to place the minimum number of registers even if that schedule is already determined. Solving (1) and (2) is of great interest, since reducing the number of registers allows to reduce the number of control signals, the area of the circuit, and the power consumption.

In [4], the authors have provided two polynomial-time solvable methods to determine schedules for reducing register requirements, and the number of the required phases. Compared to the original method [2], these methods proved very efficient in reducing the number of registers and the number of the required phases. Nevertheless, the problem of how to efficiently place registers in the circuit is not addressed.

In this paper, we focus on solving simultaneously (1) and (2) that are outlined above. More precisely, we tackle the problem of determining schedules that yields the minimum number of registers, where the optimal register placement is done during the schedule determination. To optimally solve that problem, we provide a mixed integer linear program (MILP), which we use to derive a linear program (LP) that is polynomial-time solvable. To test the effectiveness of the approach, we experiment the MILP and the LP on well known benchmarks, and we show the superiority of that approach over the original method [2].

This paper is organized as follows. The next section gives some notations and definitions used in this article. Section 3 briefly reviews the registers placement step in the method based on software pipelining, which was outlined above. Also, it shows that the algorithm used to place registers is greedy. The problem we tackle and its optimal solution are presented in Section 4, and a linear program for that problem is given in Section 5. Section 6 provides experimental results and Section 7 concludes the paper.

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 [2]) 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 $d(v) \in N$, which is assumed to be fixed. 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 to illustrate the initial specification for the problem to optimize. The initial specification is in general a synchronous circuit with a single-phase clock. The minimum clock period of the circuit in Figure 1 as specified is 7, which is equal to $d(v_5) + d(v_1)$.

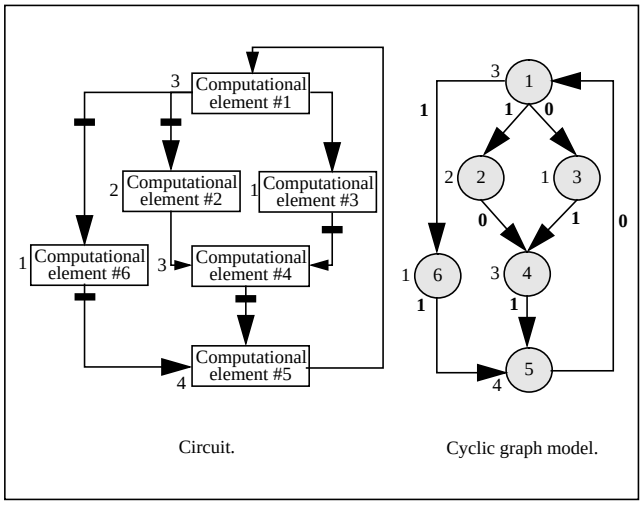


Figure 1. Sample circuit and its directed cyclic graph model.

2.2. Periodic schedules

We define a schedule s [1, 2] as a function $s : N \times V \rightarrow Q$, 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: $\forall n \in N,\ \forall v \in V: s_{n+1}(v) = s_n(v) + P$. (1)

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:

$$\forall n \in N,\ \forall e_{u,v} \in E : s_{n + w(e_{u,v})}(v) \geq s_n(u) + d(u)\,. \qquad (2)$$

2.3. Maximum throughput of synchronous sequential circuits

The throughput, T, of a 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 [1]:

$$T = Min_{c \in C}\left(\left(\sum_{e_{u,v} \in c} w(e_{u,v})\right) \Big/ \left(\sum_{\forall v \in V \text{ and } e_{u,v} \in c} d(u)\right)\right), \qquad (3)$$

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 [6, 10], which can be solved in the general case in $O(|V| \cdot |E| \cdot \log(|V| \cdot d_{max}))$ [10], where $d_{max} = Max_{v \in V}(d(v))$. A possible method to solve this problem is to iteratively apply Bellman-Ford's algorithm [5] for longest paths on the graph $G_P = (V, E, d, w_P)$ derived from G by letting:

$$w_P(e_{u,v}) = d(u) - P \cdot w(e_{u,v})\,, \qquad (4)$$

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 [1]. Without loss of generality, we assume that P is greater than or equal to the execution delay of each computational element in the circuit.

For the example in Figure 1, we have that P = 6. This value corresponds to the cycle defined by vertices v_1, v_2, v_4, and v_5.

2.4. Schedule for a given throughput

From equation (1) and inequality (2), we have that:

$$\forall e_{u,v} \in E,\ s_0(v) - s_0(u) \geq d(u) - P \cdot w(e_{u,v})\,. \qquad (5)$$

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 (5). To solve this system, the graph G_P, previously described, may be used. To find an ASAP schedule, Bellman-Ford's algorithm [5] 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 (4). Finally, step 3, the ALAP schedule is obtained by multiplying each result in step 2 by -1. Relatively to $v_x = v_1$, the ASAP schedules of vertices v_1, v_2, v_3, v_4, v_5, and v_6 of the circuit in Figure 1 are 0, -3, 3, -1, -4, and -3, respectively. Their ALAP schedules are 0, -3, 4, -1, -4, and 1, respectively.

2.5. Schedule graph

As in [2], a periodic schedule, with period P, is expressed 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 $w_s : E \rightarrow Q$ 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:

$$\forall e_{u,v} \in E,\ w_s(e_{u,v}) = s_{w(e_{u,v})}(v) - s_0(u)\,. \qquad (6)$$

Because s is periodic with period P, equation (6) may be written as follows: $\forall e_{u,v} \in E,\ w_s(e_{u,v}) = s_0(v) - s_0(u) + P \cdot w(e_{u,v})$ (7)

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 (2). Figure 2 shows a consistent schedule graph, where edges are labeled with w_s values, for the circuit in Figure 1, using the ASAP schedule determined in Section 2.4.

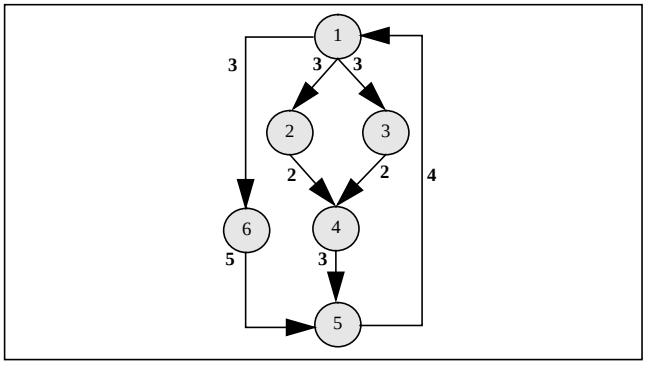
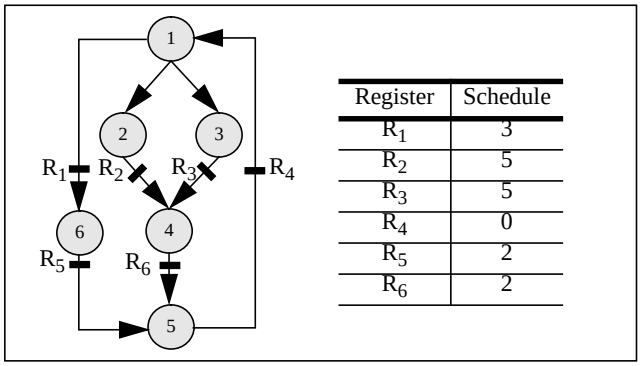


Figure 2. Schedule graph.

3. Register placement

In the method proposed in [2], 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 [2] 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 depicted by Figure 3. The number of registers that are placed is 6 and the number of phases is 4.



Register	Schedule
R_1	3
R_2	5
R_3	5
R_4	0
R_5	2
R_6	2

Figure 3. Register placement an their schedules.

The algorithm for register placement in [2] is not exact. Indeed, for Figure 3, R_1 can be omitted.

4. Problem formulation and optimal solution

As mentioned in Section 1, two problems arise in the method based on software pipelining proposed in [2]: the first one is how to determine a schedule that yields the minimum number of required registers, and the second one is how to place the minimum number of required registers even if that schedule is already determined. Our focus in this paper is to simultaneously solve these two problems. More precisely, the problem (*Prob*) we tackle is to determine a schedule with the minimum register requirements, where the register placement is done during the schedule determination. To optimally solve *Prob*, we provide a mixed integer linear program (MILP), and use it to derive a linear program which is polynomial-time solvable.

Before presenting that MILP, let us first give some requirements. Figure 4 gives a portion of the cyclic graph modeling the circuit, where i and j are two computational elements. $x_{i,j}$ denotes the number of registers that must be placed on the arc $e_{i,j}$ to guarantee that the length, $l_{i,j}$, of every path that goes to j via i, is less than or equal to the optimal clock period P. $l_{i,j}$ will be defined in the following. Note that as in [2], operation chaining is assumed, and hence no register is required if $l_{i,j} \le P$. Suppose that paths that go to j via i are already examined in order to determine if some registers must be placed on them or not. Let m_i be a no-negative real greater than or equal to each rest obtained by dividing the length of each one of those paths by P. The length $l_{i,j}$ of every path that goes to j via i is the sum of m_i and $w_s(e_{i,j})$, where $w_s(e_{i,j})$ is defined by equation (7). $y_{i,j}$ is the rest of the division of $l_{i,j}$ by P. We require that $m_i \le (P - d(i))$, which guarantee that if a register R is on the output of computational element i, then its schedule will be after i finishes its execution.

Figure 5 presents a mathematical formulation to *Prob*. The objective function expresses the number of registers to be placed in the circuit. Equations (8), (9), and (10) are equivalent to the definition of $x_{i,j}$, $y_{i,j}$, and m_i, respectively. Inequality (11) is equivalent to (5). (13) is required, since the number of registers must be an integer. In this formulation, the variables are $x_{i,j}$, $y_{i,j}$, m_i and the schedule $s_0(u)$ for each computational element u.

The formulation in Figure 5 can be linearized as follows. Using the fact that $\lfloor x \rfloor \le x < \lfloor x \rfloor + 1$, and that no register is required if the length of a path is less than or equal P, equation (8) can be replaced by:

$$\forall e_{i,j} \in E,\ x_{i,j} \le (s_0(j) - s_0(i) + P \cdot w(e_{i,j}) + m_i)/P \le x_{i,j} + 1 \quad (14)$$

Equations (9) and (10) together can be replaced by

$$\forall e_{k,i} \in E,\ m_i \ge (s_0(i) - s_0(k) + P \cdot w(e_{k,i}) + m_k) - P \cdot x_{k,i} \quad (15)$$

After linearizing the formulation in Figure 5, we obtain the MILP to optimally solve *Prob* as presented in Figure 6. In this figure, equations (16) and (17) are equivalent to (14). (18) is equivalent to (15). (19), (20) and (21) are equivalent to (12), (11) and (13), respectively. The variables are not negative.

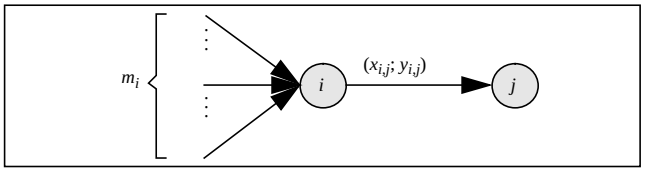


Figure 4. Illustration of the variables of the MILP.

$$Minimize \left(\sum_{\forall e_{i,j} \in E} x_{i,j} \right)$$

Subject to:

$$\forall e_{i,j} \in E,\ x_{i,j} = \left\lfloor (s_0(j) - s_0(i) + P \cdot w(e_{i,j}) + m_i)/P \right\rfloor \quad (8)$$

$$\forall e_{i,j} \in E,\ y_{i,j} = (s_0(j) - s_0(i) + P \cdot w(e_{i,j}) + m_i) - P \cdot x_{i,j} \quad (9)$$

$$\forall i \in V,\ m_i \ge (y_{k,i})_{k \in Fanin(i)} \quad (10)$$

$$\forall e_{i,j} \in E,\ s_0(j) - s_0(i) \ge d(i) - P \cdot w(e_{i,j}) \quad (11)$$

$$\forall i \in V,\ m_i \le P - d(i) \quad (12)$$

$$\forall e_{i,j} \in E,\ x_{i,j} \text{ is integer} \quad (13)$$

Figure 5. A mathematical formulation to solve *Prob*.

$$Minimize \left(\sum_{\forall e_{i,j} \in E} x_{i,j} \right)$$

Subject to:

$$\forall e_{i,j} \in E,\ -P \cdot x_{i,j} - s_0(i) + s_0(j) + m_i \ge -P \cdot w(e_{i,j}) \quad (16)$$

$$\forall e_{i,j} \in E,\ P \cdot x_{i,j} + s_0(i) - s_0(j) - m_i \ge P \cdot w(e_{i,j}) - P \quad (17)$$

$$\forall e_{k,i} \in E,\ P \cdot x_{k,i} + s_0(k) - s_0(i) + m_i - m_k \ge P \cdot w(e_{k,i}) \quad (18)$$

$$\forall i \in V,\ m_i \le P - d(i) \quad (19)$$

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

$$\forall e_{i,j} \in E,\ x_{i,j} \text{ is integer} \quad (21)$$

Figure 6. A MILP to optimally solve *Prob*.

5. A linear program for solving *Prob*

Linear programs are polynomial-time solvable [7, 8]. A linear program for solving *Prob* can be obtained by deleting the constraint that $x_{i,j}$ is integer in Figure 6. In this case, once the linear program is solved, the number of registers to be placed on the arc $e_{i,j}$ is $\lceil x_{i,j} \rceil$. Due the space limitation, details on why it is possible to place $\lceil x_{i,j} \rceil$ registers on the arc $e_{i,j}$ can be found in [3].

6. Experimental results

To test the effectiveness of our approach, the MILP in Figure 6 and the corresponding linear program (LP), obtained by ignoring the constraint *integer* in (21), are experimented on well known benchmarks. Circuits from the ISCAS89 benchmark suite are used to test the efficiency of the LP in terms of the run-time and of the reduction of the number of registers inserted in the circuit. The mathematical formulations for each circuit are automatically generated by a module we coded in C++ and integrated in a tool we developed in [4]. We did not implement the cited polynomial-time algorithms for linear programs, but the Lp_Solve tool [11] (in the public domain) is used to solve the generated mathematical formulations. Obtained results are given in Tables 1 and 2, where the first column gives the name of the circuit and the second column presents the number, N_1, of registers placed using the algorithm in [2] that uses ALAP as a schedule. The number, N_2, of registers placed by MILP or by LP are presented in the third column. The fourth column gives the relative gain defined as $((N_1 - N_2)/N_1) \times 100\%$. For Table 2, the fifth column gives the run-time in seconds on an UltraSparc 10 with 1GB RAM. As Table 1 reports, significant reductions of the number of required registers are obtained. Substantial reductions are also obtained using the LP. Indeed, as summarized by Table 2, reductions as high as 77.46% are obtained in less than 181.4s.

Table 1. Register placement by [2] vs. by MILP.

	#registers placed by [2]	**#registers placed using Figure 6**	**Relative gain**
Figure 1	6*	5	16.66%
SOIIR Filter [1]	3	2	23.33%
Polynomial Div. [1]	9	4	55.55
Correlator [2]	6	6	0%
FOWDEF [9]	14	8	42.85%

Table 2. Register placement by [2] vs. by LP.

	#registers placed by [2]	**#registers placed using the LP**	**Relative gain**	**Run-time (s)**
Figure 1	6*	5	16.66%	0.01
SOIIR Filter [1]	3	3	0%	0.02
Polynomial Div. [1]	9	4	55.55%	0.02
Correlator [2]	6	6	0%	0.01
FOWDEF [9]	14	12	14.28%	0.06
S344	131	53	59.54%	1.63
S641	142	32	77.46	1.25
S1423	422	216	48.81%	17.31
S5378	1033	581	43.75%	181.4
S9234	1042	466	55.27%	93.29

*: ASAP schedule is used.

7. Conclusions

A method based on software pipelining has recently been proposed to optimize mono-phase clocked sequential circuit. The resulting circuit is a multi-phase clocked circuit, where all clocks have the same period. To preserve the behavior of the original circuit, registers are placed according to a schedule, which has the maximum throughput.

In that method, two problems arise: how to determine schedules that lead to a minimal register requirements, and how to place the minimum number of required registers even if these schedules are already determined.

In this paper, we have simultaneously tackled these two problems. We have provided a mixed integer linear program and used it to derive a linear program, which is polynomial-time solvable. Experimental results on well known benchmarks confirmed the effectiveness of the approach we propose. Indeed, significant reductions of the number of required registers have been obtained in very short run-time.

References

[1] I.-E. Bennour, *Estimation de la performance et méthodes d'allocation dans la synthèse de systèmes numériques*, *Thèse de doctorat*, DIRO, Université de Montréal, 1996.

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

[3] F.-R. Boyer, E.-M. Aboulhamid and Y. Savaria, "An Efficient Verification Method for a Class of Multi-phase Sequential Circuits", *The 7th IEEE International Conference on Electronics, Circuits & Systems*, Lebanon, December 17-20, 2000.

[4] N. Chabini, E.-M. Aboulhamid and Y. Savaria, "Reducing Register and Phase Requirements for Synchronous Circuits Derived Using Software Pipelining Techniques", *Proceedings of the IEEE Computer Society Annual Workshop on VLSI*, Orlando, Florida, April, 2001.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, New York, NY: McGraw-Hill, 1990.

[6] 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.-1*, No. 1, Vo. 39, Jan. 1992.

[7] N. Karmakar, "A New Polynomial-Time Algorithm for Linear Programming", *Combinatorica*, Vo. 4, 1984. xxx

[8] L.-G. Khachian, "A Polynomial Algorithm in Linear Programming", *Soviet Math. Doklady*, Vo. 20, 1979.

[9] S. Y. Kung, H. J. Whitehouse and T. Kailath, *VLSI and Modern Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985, pages: 259-60.

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

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