UNIVERSITAT POLITÈCNICA DE CATALUNYA

LOOP PIPELINING WITH RESOURCE AND TIMING CONSTRAINTS

Autor: Fermín Sánchez

October, 1995

8 TCLP: LOOP PIPELINING WITH TIMING CONSTRAINTS

8.1 INTRODUCTION

The term *timing constraints* has previously been used to refer to two types of timing constraints:

- *Local timing constraints.* In some types of applications, i.e. the behavioral synthesis of interfaces [NT86], it is useful to specify minimum and/or maximum timing constraints between operation pairs. Approaches to solve scheduling with *local timing constraints* can be found in [NT86, KD90b, NK90].
- *Global timing constraints.* Some applications require data to be processed with a maximum delay time. In these applications, the objective is to find a loop schedule which satisfies the timing constraints; that is, a schedule no longer than the given maximum delay time. Many real-time signal processing applications in video, image, speech and telecom processing [CGD94] are examples of this kind of application. Some Integer Linear Programming approaches have been proposed to solve scheduling (not loop pipelining) with *global timing constraints* [HLH91, Ach93]. Force Directed Scheduling [PK89a] addresses both *local* and *global timing constraints*.

This chapter studies *loop pipelining with global timing constraints*, which is significantly different from *loop pipelining with resource constraints*, studied in Chapter 6. *Loop pipelining with global* *timing constraints* can be defined as follows: "*Given a fixed throughput, finding a schedule of a loop which minimizes the resource requirements*". The throughput requirement is defined as the maximum number of cycles required to execute each iteration of the loop, and is denoted by T_{max}. Even though the number of cycles of a schedule is an integer number, a non-integer *II* can be obtained by unrolling the loop. As an example, let us assume that the objective is executing a loop in no more than seven cycles ($T_{max} = 7$). Let us also assume that a single loop iteration spends 7 cycles when using a given set of resources, but a schedule in 13 cycles is found if the loop is unrolled twice. With this assumption, each iteration of the unrolled loop is executed (on average) in 6.5 cycles ($II = 6.5$). However, although the average time for each iteration is less than T_{max}, the iteration time of some iterations may be greater than T_{max} cycles. This mishap must be avoided in some applications.

This chapter presents TCLP (*Time-Constrained Loop Pipelining*), a new methodology to solve *loop pipelining with global timing constraints*. Although *loop pipelining with global timing constraints* is an NP-hard problem [GJ79], TCLP solves it very efficiently, as the results presented at Section 8.4 shows. Henceforth, we will indiscriminately use the terms *global timing constraints* and *timing constraints*.

The main contributions of TCLP with regard to the previous *time-constrained scheduling* approaches [PK89a, HLH91, Ach93] are the following:

- Absolute lower bounds are computed for each type of resource. When these bounds are met, an optimal solution is guaranteed.
- Not only is a set of resources with minimum area cost computed for a given T_{max}, but also the execution throughput is increased by using such a set of resources. This goal is achieved by exploring in increasing order of throughput different unrolling degrees of the loop with different expected initiation intervals. Farey's series are used to do such exploration.
- The number of required registers is finally reduced, producing a schedule with minimum cost in time and area (resources and registers).

8.1.1 Strategy overview

Figure 8.1 shows the flow diagram of TCLP. As we can see, the steps performed by TCLP to find a schedule for a given maximum time constraint T_{max} are the following:

1. Initially, the *minimum initiation interval* of the loop is computed. The problem has no solution when $T_{max} < MII$.
2. TCLP next attempts to find a loop schedule with minimum cost (minimum area requirements and maximum throughput). In order to compute the initial architecture, the absolute lower bound on the number of resources of each type required to execute the loop is computed. Section 8.2.2 explains how this calculation is done.
3. By using the lower bound for each type calculated at step 2 as the initial set of resources, TCLP tries to find a schedule in T_{max} cycles. If a schedule is found, the algorithm executes step 5. Otherwise, it executes step 4.

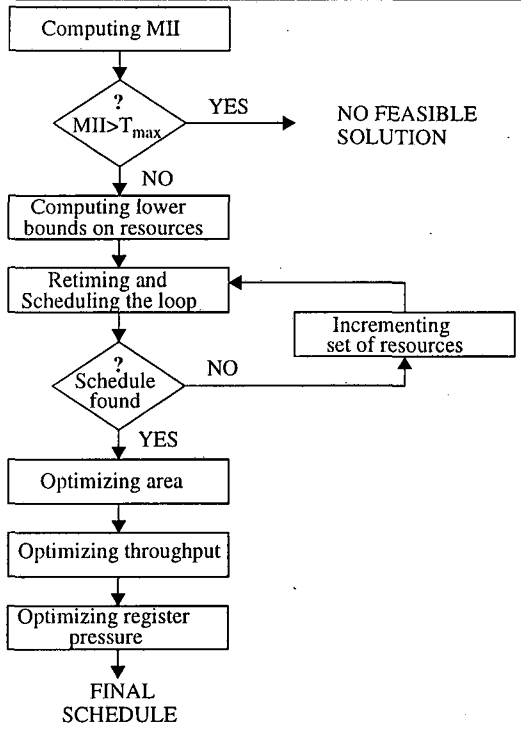
Computing MII
?
MII>T_{max}
YES
NO FEASIBLE SOLUTION
NO
Computing lower bounds on resources
Retiming and Scheduling the loop
Incrementing set of resources
?
Schedule found
NO
YES
Optimizing area
Optimizing throughput
Optimizing register pressure
FINAL SCHEDULE

Figure 8.1 Flow Diagram of TCLP

4. When no schedule is found by using the current set of resources, heuristics are used to select a type of resource. The number of resources of such a type is increased. One instance of the selected resource is added to the architecture and step 3 is executed again. Section 8.2.3 shows such heuristics.
5. At this point, a schedule in T_{max} cycles has been found for a given set of resources. Since the heuristics used at step 4 do not guarantee a minimum set of resources (and therefore a minimum area cost), TCLP attempts to reduce the number of resources of the current architecture while maintaining the throughput of the schedule. Section 8.2.4 explains the approach used to do this.
6. Once a minimum area-cost schedule has been found, the execution throughput is increased without varying the set of resources. Throughput is explored in increasing order by using Farey's series $F_{T_{max}}$, as shown in Chapter 6.
7. Finally, once a schedule with minimum area cost and maximum throughput has been found, TCLP reduces the number of required registers.

On one hand, results in Appendix C show that increasing the execution throughput usually increases the number of required registers. This is because throughput is reduced by unrolling the loop and, in general, the number of required registers increases with the unrolling degree. Therefore, step 6 may be eliminated depending on whether the area consumed by the registers is considered or not.

On the other hand, reducing the number of registers may increase the execution throughput. This is because instructions in the schedule may be compacted by *RESIS*, achieving in some cases a reduction in the *II* of the schedule. Therefore, we conclude that steps 6 and 7 may be swapped according to which objective is more important. In this assumption, throughput is reduced whilst the number of registers is not increased.

8.2 TCLP APPROACH

8.2.1 Minimum initiation interval

Since the number of resources must be decided by TCLP, only recurrences in the loop must be taken into account to calculate the *minimum initiation interval*. Therefore, the *MII* is the *RecMII* (*recurrences minimum initiation interval*) studied in Chapter 3 (definition 3.11).

8.2.2 Absolute lower bound on the set of resources

Since the expected number of cycles of the loop schedule is known in advance (T_{max}), an absolute lower bound on the number of resources of each type required to execute the loop can be computed. Two lower bounds must be taken into account:

1. Let n_i be the number of times a resource of type i is used by an iteration of the loop[1]. A lower bound on the number of resources of type i required to execute the loop, LB_i, is $LB_i = \left\lceil \frac{n_i}{T_{max}} \right\rceil$.
2. On the other hand, the execution pattern of any operation may require EP_i ($EP_i > LB_i$) resources of a given type i in a given cycle to be executed correctly.

The absolute lower bound on the number of resources of type i required to execute the loop, N_i, is the maximum between LB_i and EP_i.

$$N_i = \max(LB_i, EP_i)$$

TCLP starts with an architecture composed of N_i resources for each type of resource i. However, having N_i resources of every type i does not guarantee that a schedule in T_{max} cycles exists, since dependences in the loop may prevent a schedule from being found.

8.2.3 Increasing the number of resources

Finding a schedule in T_{max} cycles

Algorithm 6.3 (*retiming_and_scheduling*), described in Section 6.4.2, is used to find a schedule for the loop.

Retiming_and_scheduling successively transforms the initial π-graph by means of *dependence retiming*. After each transformation, the π-graph is scheduled (by using Algorithm 5.6), attempting to find a schedule in T_{max} cycles in the current set of resources. *Dependence retiming* is done until a schedule is found for some π-graph or retiming can no longer be performed because the π-graph with the highest quality (see Section 6.4.1) has been found. If no schedule is found, the number of resources of some type is increased and *retiming_and_scheduling* is executed again. This process is repeated until a schedule is found.

Which type of resource must be increased ?

Heuristics are used to determine which type of resource(s) is(are) mainly responsible for not finding the schedule. Once such a resource has been found, the number of resources of this type is increased in one unit and *retiming_and_scheduling* is executed again by using the new set of resources.

Two different reasons can preclude that a schedule in the expected number of cycles be found[2]:

[1] n_i is computed by adding the number of times each operation of the loop uses a resource of type i.

[2] We assume the critical path of the π-graph is less than or equal to the expected II. Otherwise, the schedule would be impossible.

- *Some operation cannot be scheduled because of sufficient resources are not available.* When sufficient resources are not available to schedule an operation u at its $ASAP$ time, it is deferred to the next cycle, and so on. Deferring u may produce the deferring of some successors of u. Therefore, the $ASAP$ time of some successors of u increases as well as their time frame for scheduling ($ALAP - ASAP$) decreases. As the number of resources is limited, some of these successors may not be scheduled within this range. When this happens, the deferring of u is considered responsible for not finding the schedule. The resource which causes the deferring of u is increased by one unit.

  The π-graph used to determine u is the π-graph with the greatest *quality* found by *retiming_and_scheduling*. We consider this π-graph as the one with the greatest constraints imposed by the resources, since it is the one with the fewest constraints imposed by data dependences. Figure 8.2 shows an example. The schedule of the π-graph with the highest quality (by using 2 adders and 1 multiplier) produces the deferring of instructions B and D. D is scheduled after its ASAP-time due to the deferring of B. B is scheduled after its ASAP-time because only one multiplier is available at cycle 1. Therefore, the number of multipliers is increased. The final schedule, by using 2 adders and 2 multipliers, is shown in Figure 8.2(d).

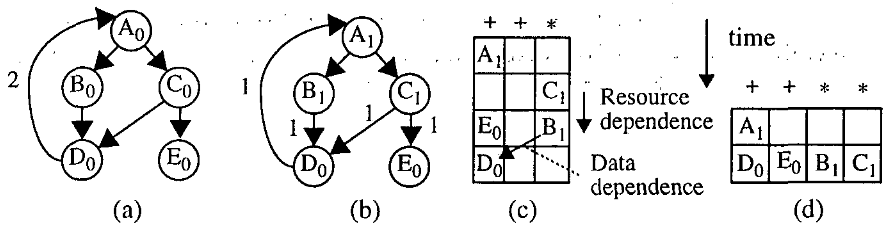

Figure 8.2 Resource responsible for not finding the schedule
(a) Example of π-graph
(b) π-graph with the highest quality
(c) Schedule for π-graph (b) by using 2 adders and 1 multiplier
(d) Schedule for π-graph (b) by using 2 adders and 2 multipliers

- *There is no time frame to schedule some instruction u.* In this assumption, $ALAP(u) < ASAP(u)$. Figure 8.3 illustrates this fact with an example. Let us assume that the instruction latency is $L_u = L_v = 2$, and $L_w = 1$. Figure 8.3(b) shows a possible schedule in which u and w have already been scheduled at cycles 0 and 3 respectively, and v has not yet been scheduled. When the scheduler attempts to schedule v, it finds that v should be scheduled after (or at) cycle 2 because of ILD (u, v) (time frame TF_2), and before (or at) cycle 1 because of ILD (v, w) (time frame TF_1). Since both time frames are disjoint, the scheduler fails. This problem cannot always be solved in loops with recurrences. When this occurs, TCLP increments the resource most used by the loop.

Algorithm

Algorithm 8.1 presents the proposed approach.

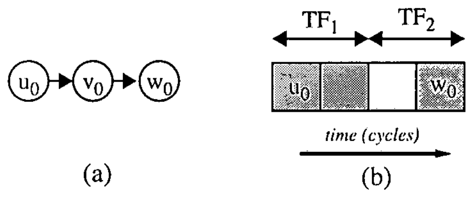

Figure 8.3 (a) Example of π-graph (b) v has no time-frame to be scheduled

```
function increase_set_of_resources($\pi$, $II$, $\mathcal{R}$);
    { $\mathcal{R}$ is the set of resources}
    loop
        found := retiming_and_scheduling($\pi$, $II$);
        exit if found;
        R:=select resource to increase;
        increase resource $R$;
```

Algorithm 8.1 Algorithm to increase the architecture until a schedule is found

Each iteration of the while loop executes in $O(V^2E + VE^2)$ time (the execution time of *retiming_and_scheduling*). Experiments have shown that the number of iterations of the loop is very small. Therefore, we conclude that the execution time of the whole algorithm is the same as the execution time of one iteration of the loop.

8.2.4 Reducing the set of resources

In some cases, the heuristics used to increase the resources might lead to solutions that are no local minima. For example, if the first instruction deferred in schedule from Figure 8.2(c) was an addition instead of a multiplication, *increase_set_of_resources* would increase the number of adders, and no schedule in two cycles would be found. Therefore, when the number of multipliers would be later increased, *TCLP* would find a schedule in two cycles by using more adders than those actually required. To avoid this, TCLP attempts to reduce the number of resources of the architecture after a schedule has been found (while maintaining the initiation interval of the schedule).

In order to obtain a schedule with a lower area cost, resources are greedily explored in decreasing order of area. That is, if a multiplier uses more area than an ALU, TCLP tries to reduce the number of multipliers before reducing the number of ALUs. It has been shown that this step is very important in the optimization of the number of resources, since it is able to correct errors introduced by the heuristics described in Section 8.2.3. The combination of both steps (increasing and decreasing resources) produces optimal results in almost all cases by using little CPU time, as shown in Section 8.4. Algorithm 8.2 (*reduce_number_of_resources*) is used by TCLP to optimize the area cost of the schedule. In the algorithm, N_i is the absolute lower bound on the number of resources of type i required to execute the loop, and R_i is the current number of resources of type i.

```
function reduce_number_of_resources($\pi$, $II$, $\mathcal{R}$);
    for each type of resource (i) do
    (explored in decreasing order of area)
        loop
            exit if $R_i = N_i$;
            remove a resource of type $i$;
            found := retiming_and_scheduling($\pi$, $II$);
            if not found then
                    add a resource of type $i$;
                    exit;
```

Algorithm 8.2 Algorithm to reduce area cost.

Given that experiments show that the loop iterates only a few times, we conclude that the execution time of *reduce_number_of_resources* is the same as the execution time of *retiming_and_scheduling*: $O(V^2E + VE^2)$.

8.2.5 Increasing throughput

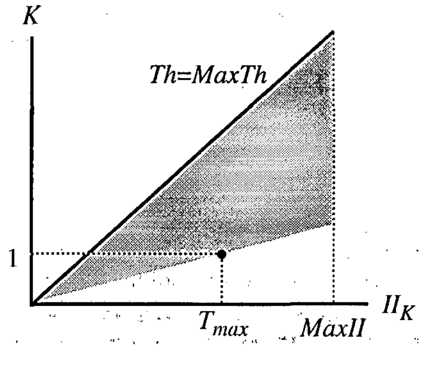

Figure 8.4 Exploration of the throughput diagram

Once the number of resources has been fixed, TCLP tries to increase the schedule throughput without varying the architecture. This is done by exploring the points in the throughput diagram in increasing order of throughput, starting at the point $(K, II_K) = (1, T_{max})$ and finishing at any point in the line $Th = MaxTh$. The shadowed triangle in Figure 8.4 shows the space to be

```
function $increase\_throughput(\pi, T_{max})$;
    $X := 1; Y := T_{max}$;
    $found := true$;
    while $found$ and $\frac{X}{Y} < MaxTh$ do
        $\pi$ :=unroll(initial_loop,X);
        $found$ :=$retiming\_and\_scheduling(\pi, Y)$;
        $\frac{X}{Y}$ :=Next element in $F_{MaxII}$;
```

Algorithm 8.3 Algorithm to find the maximum-throughput schedule

explored. Since $MaxII$ may represent the length of a schedule of several loop iterations, $MaxII$ may be greater than T_{max}. Algorithm 8.3 shows how the schedule throughput is increased.

In general, only a few points in the throughput diagram are explored. The number of points to explore does not depend on the size of the π-graph, and therefore it does not influence the calculation of the computational complexity of function *increase_throughput*. The execution time of *increase_throughput* is the same as the execution time of *retiming_and_scheduling*: $O(V^2E+VE^2)$.

On one hand, increasing the unrolling degree of the loop also increases the register pressure. Therefore, this step may be avoided when the number of registers is limited or the size of the registers has greater influence in the final area of the chip. On the other hand, reducing registers may increase the throughput in some cases, because *incremental scheduling* may move instructions from the last cycle of the schedule to previous cycles, reducing the initiation interval.

Moreover, since the schedule of a loop unrolled X times taking Y cycles does not guarantee that each iteration is executed in less than T_{max} cycles (it only can guarantee that each iteration executes on average in $\frac{X}{Y}$ cycles, with $\frac{X}{Y} < T_{max}$), this fact must be verified before considering the schedule as a valid schedule.

8.2.6 Reducing register pressure

Once a schedule with maximum throughput and minimum area (by taking only resources into account) has been found, TCLP attempts to reduce register pressure by reducing the number of registers required to store partial results. To do so, the algorithm shown at Chapter 7 is used. The execution time of such an algorithm is $O(V^3E + VE^2)$.

8.2.7 TCLP. Execution time

The computational complexity of TCLP is the sum of the execution times of all functions used by TCLP. That is $O(V^3E + VE^2)$.

8.3 EXAMPLE

Let us illustrate how TCLP works by using an example. We have chosen the Fast Discrete Cosine Transform Kernel (FDCT) from [MD90]. The maximum number of cycles of the schedule, *MaxII*, has been fixed to 50 cycles. The objective is to execute the FDCT in less than 18 cycles ($T_{max} = 18$). The FDCT consists of 16 multiplications, 13 additions and 13 subtractions, as Figure 8.5(a) shows. We will make the same assumptions as [MD90]. We will assume each operation can be executed in a single cycle in the appropriate FU (a multiplier, an adder or a subtracter).

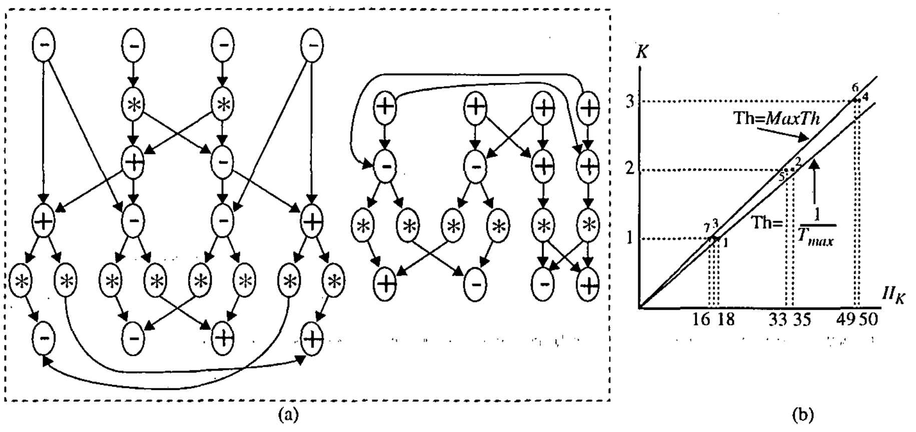

Figure 8.5 (a) DG of FDCT (b) Throughput exploration

TCLP works as follows:

- The π-graph has no recurrences. Therefore, $RecMII = 0$.
- The lower bound on the number of required resources is one multiplier, one adder and one subtracter. *Retiming_and_scheduling* finds a schedule in 18 cycles by using the previous architecture (this takes less than 0.8 seconds).
- The number of resources cannot be reduced, since it is minimal.
- TCLP then attempts to reduce the length of the schedule without increasing the number of resources. Farey's series F_{50} are explored ($MaxII = 50$), starting at fraction $\frac{1}{18}$ (point 1 in Figure 8.5(b)). Since the *MII* computed by using one FU of each type is $MII = 16$, the exploration of Farey's series will finish when a fraction less than $\frac{1}{16}$ is generated. The fractions to explore are: $\frac{1}{18}$, $\frac{2}{35}$, $\frac{1}{17}$, $\frac{3}{50}$, $\frac{2}{33}$, $\frac{3}{49}$ and $\frac{1}{16}$. These fractions are represented in Figure 8.5(b). A fraction $\frac{x}{y}$ produces the search of a schedule of x iterations of the loop in y cycles. A fraction is explored only when a schedule has been found for the previous one. TCLP finds a schedule for each one of the fractions, stopping when it finds a schedule for 1 iteration in 16 cycles (point 7 in Figure 8.5(b)). The time used to explore all the fractions was 44.2 seconds.

$Tmax$	LB FUs	req. FUs		Results			Cpu (secs)	
	+	+	MII	*II*	*R*	K/II_K	**Tf**	**Tr**
3	6	6	3	3	9	1/3	0.21	0.00
4	5	5	3.4	3.4	31	5/17	0.11	268
5	4	4	4.25	4.25	24	4/17	0.13	143
6	3	3	5.66	5.66	17	3/17	0.11	20.3
7	3	3	5.66	5.66	17	3/17	0.16	71.1

Table 8.1 Cytron's example

- After reducing the length of the schedule, TCLP attempts to reduce the number of registers. The schedule found after the exploration of Farey's series uses 18 registers. After *reducing the SPAN*, TCLP finds a schedule using 15 registers. The final schedule, found after *moving operations*, requires only 12 registers. The time used to reduce the number of registers (*SPAN reduction + incremental scheduling*) was 2.55 seconds.

8.4 EXPERIMENTAL RESULTS

We present here only results obtained for HLS systems, since there is little point in presenting results for superscalar processors and VLIW processors (the architecture is fixed).

Optimal time-constrained scheduling has been studied in [HLH91, Ach93], and some results can be found in [Ach93]. When comparing TCLP to [Ach93], TCLP obtains schedules requiring less area because [Ach93] is an ILP approach which does not perform loop pipelining. Thus, we will compare the results obtained by TCLP to the lower bounds on the number of resources and the *MII*. The *MII* has been computed for each case by using the set of resources required by TCLP to find a schedule. To our knowledge, these are the first results published for *time-constrained loop pipelining*.

Tables 8.1 to 8.5 show the results obtained. The first columns show T_{max} and the lower bounds on FUs (LB FUs) computed for each T_{max}. The following columns specify the number and type of resources required to achieve the given T_{max} (req. FUs). The *MII* calculated for each set of used FUs, the *II* of the schedule found by TCLP, the lower bound on the number of registers required for each schedule (R) and the fraction of the Farey's series which is associated to the schedule (K/II_K) are shown in the next columns. Finally, the last two columns show the time used to find an optimal schedule in area cost (Tf) and the time required to optimize the schedule throughput and reduce register pressure (Tr). Note that Tr is sometimes large, whilst Tf is usually small. We have considered $MaxII = 50$ for all the examples, except for (*) in table 8.5, in which $MaxII = 10$. This has been considered in order to show how the parameter $MaxII$ influences the execution time of TCLP, since $MaxII$ determines the number of Farey's fractions to explore when throughput is optimized. Note that an optimal solution (by taking resource requirements into account) is achieved in most cases.

$Tmax$	LB FUs		req. FUs			Results			Cpu (secs)	
	*	ALU	*	ALU	MII	II	R	K/II_K	Tf	Tr
6	2	1	2	1	6	6	6	1/6	0.10	0.00
10	2	1	2	1	6	6	6	1/6	0.18	49.8
12	1	1	1	1	12	12	6	1/12	0.18	0.00

Table 8.2 Differential Equation

$Tmax$	LB FUs		req. FUs			Results			Cpu (secs)	
	*	+	*	+	MII	II	R	K/II_K	Tf	Tr
16	1	2	2	3	16	16	10	1/16	7.18	0.00
17	1	2	2	2	16	17	10	1/17	5.00	15.8
18	1	2	2	2	16	17	10	1/17	2.92	19.2
19	1	2	1	2	16	19	9	1/19	0.73	6.36
27	1	1	1	2	16	19	9	1/19	3.42	22.3
28	1	1	1	1	26	28	12	1/28	0.70	1.70

Table 8.3 Fifth-Order Elliptic Filter with Non-Pipelined Multipliers

$Tmax$	LB FUs		req. FUs			Results			Cpu (secs)	
	*	+	*	+	MII	II	R	K/II_K	Tf	Tr
16	1	2	1	3	16	16	10	1/16	3.38	0.00
17	1	2	1	2	16	17	9	1/17	0.75	15.7
20	1	2	1	2	16	17	9	1/17	0.63	23.3
26	1	1	1	2	16	17	9	1/17	0.70	33.6
27	1	1	1	2	16	17	9	1/17	0.68	36.7
28	1	1	1	1	26	28	11	1/28	0.70	1.63

Table 8.4 Fifth-Order Elliptic Filter with Pipelined Multipliers

$Tmax$	LB FUs			req. FUs				Results			Cpu (secs)	
	*	+	-	*	+	-	MII	II	R	K/II_K	Tf	Tr
3	6	5	5	6	5	5	2.66	2.66	44	3/8	1.73	15.3
4	4	4	4	4	4	4	4	4	18	1/4	1.53	0.08
5	4	3	3	4	3	3	4.33	4.36	139	11/48	0.92	1992
5(*)	4	3	3	4	3	3	4.33	4.5	27	2/9	1.00	4.18
6	3	3	3	3	3	3	5.33	5.33	38	3/16	1.15	14.3
7	3	2	2	3	2	2	6.5	6.5	23	2/13	0.83	2.80
8	2	2	2	2	2	2	8	8	18	1/8	1.51	0.01
12	2	2	2	2	2	2	8	8	18	1/8	0.65	1.18
13	2	1	1	2	1	1	13	13	12	1/13	0.66	0.00
15	2	1	1	2	1	1	13	13	12	1/13	0.66	24.6
16	1	1	1	1	1	1	16	16	12	1/16	1.28	0.00
18	1	1	1	1	1	1	16	16	12	1/16	0.86	46.7

Table 8.5 Fast Discrete Cosine Transform. $MaxII = 50$ for all cases except for (*), in which $MaxII = 10$.

8.5 SUMMARY AND CONCLUSIONS

In this chapter we present a new algorithm for *loop pipelining with global timing constraints*. The main contributions of the proposed approach with respect to the previous techniques are the following:

- Loop pipelining is considered. Previous techniques only consider scheduling (and not loop pipelining) with global timing constraints. Results show that *TCLP* improves the results obtained by other techniques.
- Absolute lower bounds are computed for each type of resource for a given *global timing constraint*. Therefore, we can compare the results obtained with the theoretically optimal ones.
- The execution throughput and the register requirements are optimized once a schedule has been found for the given timing constraint. This can be done in both orders first optimizing throughput or first optimizing registers. Optimizing throughput may increase the number of registers required by the schedule, since the unrolling degree of the loop may also be increased. Optimizing registers may also optimizing throughput because the initiation interval of the schedule may be reduced by *incremental scheduling*. Therefore, it is a designer decision which optimization must first be performed, according to its objectives.

We have compared the results obtained by *TCLP* with the previous calculated lower bounds. *TCLP* finds a schedule by using the minimum set of resources in a high percentage of cases. In those cases in which further resources are required, we think that no solution exists for the given *timing constraint* by using the minimum set of resources. For example, first line of Table 8.3 shows that 1 multiplier and 2 adders is the minimum set of resources required to execute the fifth-order elliptic filter in 16 cycles. *TCLP* requires 2 multipliers and 3 adders to find a schedule. Results from Appendix B show that no schedule in 16 cycles exists by using less registers. ILP approaches as ALPS [HLH91] require 17 cycles for 2 multipliers and 2 adders, and 19 cycles for 1 multiplier and 2 adders. Since these are exactly the resource requirements found by *TCLP*, we claim that *TCLP* may obtain optimal results in most cases.

We conclude that the proposed approach is an appropriate methodology to solve the *time-constrained loop pipelining* problem.