A Heuristic for Clock Selection in High-Level Synthesis

J. Ramanujam* Sandeep Deshpande* Jinpyo Hong* Mahmut Kandemir†

Abstract

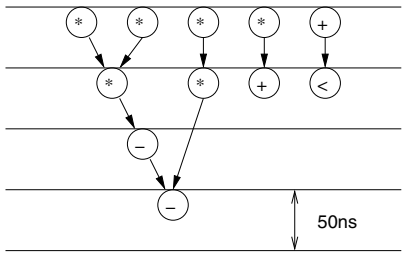
Clock selection has a significant impact on the performance and quality of designs in high-level synthesis. In most synthesis systems, a convenient value of the clock is chosen or exact (and expensive) methods have been used for clock selection. This paper presents a novel heuristic approach for near-optimal clock selection for synthesis systems. This technique is based on critical paths in the dataflow graph. In addition, we introduce and exploit a new figure of merit called the *activity factor* to choose the best possible clock. Extensive experimental results show that the proposed technique is very fast and produces optimal solutions in a large number of cases; in those cases, where it is not optimal, we are off by just a few percent from optimal.

1 Introduction

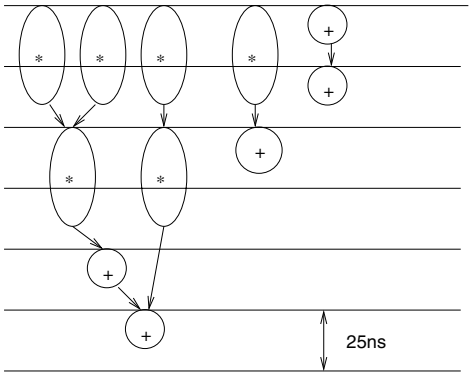
High-level synthesis (HLS) is the process of translating a behavioral description to a register transfer level (RTL) structural description [2, 5, 9]. This involves problems such as module selection, scheduling (both resource-constrained scheduling or RCS, and time-constrained scheduling or TCS), allocation and binding, and clock selection [7, 9]. Scheduling for a fixed clock helps one explore the design space in two dimensions, namely area and latency [1]. As stated in [1], this method would be useful for exploring only a small portion of a complete design space. For a more useful and exhaustive search, the clock value is also considered to be a constraint for the scheduling problem and this adds the third dimension to the search space.

Many design systems require that the clock be specified by the designer a priori. Even though the clock length is required prior to scheduling, the best value of the clock can be found only after evaluating different schedules. This interdependence makes it difficult to select a good value of clock prior to scheduling. A variety of schedules can be found, each for a different clock value and this provides the designer with greater flexibility in the choice of the final design. Since the clock value has to be known before the scheduling process itself, a good estimation of the clock value is essential. An optimal value can be found by finding all possible schedules with different candidate clock lengths exhaustively and then the best (or optimal) value can be chosen, but this would take a long time and hence it is highly undesirable. It is therefore necessary to derive algorithms that are fast and generate a near optimal value of the clock length if not the optimal value.

The choice of clock length affects the scheduling of a sequencing graph. In order to illustrate this, we consider the sequencing graph (which is the dataflow graph from which all edges other than dependencies are removed [5]) for the second order differential equation benchmark [13]; this sequencing graph is shown in Figure 1. Note that for the rest of this paper, we use the terms dataflow graphs and sequencing graphs interchangeably. We assume that all operations other than multiplication are performed on the ALU and the multiplication operations on the multiplier. Then, for this sequencing graph, we can have two different schedule lengths depending on the value of the clock used and the delay of the different functional units. If the delay of the ALU is $25ns$ and the delay of the multiplier is $50ns$ and a clock of $50ns$ is used to schedule the graph (assuming no resource constraints), then we can complete the schedule in $200ns$. This is illustrated in Figure 1(a). In this particular case, the ALU is idle for $25ns$ per clock. On the other hand, if we use a clock of $25ns$, then the schedule time is $150ns$, and in this case, none of the resources are idle as shown in Figure 1(b). *It is thus important to choose a proper value of the clock to reduce the schedule length and the idle time on the various resources.*



(a) Clock length = 50 ns Schedule length = 200 ns



(b) Clock length = 25ns Schedule length = 150 ns

Figure 1: Effect of clock length on scheduling.

Note that we have made the idealized assumption that every unit of time within a clock cycle is usable by the functional unit, just like the other works in this area [1, 4, 3, 10]. We realize that in most cases in practice a certain amount of time, say δ ns is not available in each cycle for any functional unit. This can be easily accommodated in our approach by extending the lengths of clocks by δ ns as needed.

In this paper, we present a new heuristic for clock selection. Our approach finds the clock length independent of scheduling and hence it is not computationally expensive; in this respect, it is similar to [1]. Also, as is evident from experiments, the technique generates almost as good results as compared to an exhaustive, computationally more expensive method in [4]. Note that our goal here is to reduce the amount of computation required in the rest of the

*Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803, USA. {jxr,sandeepd,jphong1}@ece.lsu.edu

†Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802, USA. kandemir@cse.psu.edu

Proceedings of the 15th International Conference on VLSI Design (VLSID'02)
0-7695-1441-3/02 $17.00 © 2002 IEEE

phases of design space exploration. Our appraoch considers only the operations on critical paths and does not consider the effect of the number of resources [11] (also referred to as resource allocation). Based on the experiments with benchmark examples, we have found that the clock value determined by our technique is close to the optimal clock value.

The rest of this paper is organized as follows. First, we discuss briefly some of the previous work on the clock selection problem as discussed in [1] and [4]. We then describe our method of finding the clock length as improvement over [1] and illustrate the differences with respect to that method.

2 Background and related work

We assume that module selection has already been done and hence delays for the functional units have already been provided from a particular library. We denote a functional unit by $\mathsf{FU_i}$ and the delay by $delay_i$. Since a clock length has to be chosen before scheduling, some of the trivial choices a designer may make are (i) the delay of one of the functional units, (ii) the delay of the slowest/fastest functional unit, and (iii) the GCD of the delays of all functional units. All of these plausible options have one common characteristic, i.e, they try to zero out the idle time on at least one of the functional units; it is important to note that this approach does not necessarily guarantee a good value for clock. The selection of the clock length not only affects the schedule, but it also affects the area and the complexity of the control circuit [5, 1]. Amongst the three methods cited above, the last one might generate a clock length that is so small that it may not be practically feasible to implement the control circuit [5]. We define the $slack$ of an operation mapped to a functional unit $\mathsf{FU_i}$ as the time during which the functional unit is idle. To illustrate this point assume that the clock length is clk, then the slack for a functional unit $\mathsf{FU_i}$ is defined as follows:

$$slack_i = clk \cdot \left\lceil \frac{delay_i}{clk} \right\rceil - delay_i. \tag{1}$$

The slack is the time for which the $\mathsf{FU_i}$ is not going to do any useful work. The utilization of the functional units can be made more efficient by decreasing the value of the slack and also possibly reducing the schedule length. We need to evaluate a set of candidate clock lengths and then choose the one for which the slack is minimum. Ideally, we would like to have zero slack for all functional units but since this is not practically possible, we try to look at possible clock values for which the slack would be minimum. The slack on a functional unit $\mathsf{FU_i}$ would be zero if the value of the clock perfectly divides its delay. This particular value of the clock need not necessarily reduce the slack on some other functional unit $\mathsf{FU_k}$. It is thus important to find a value of the clock that would reduce the slack on all the functional units. Although such a value might be difficult to find, a measure that we would like to consider is an average value of slack over all functional units as illustrated by Chang et al. [3] and then obtain a value for the clock length based on the minimum average slack. The average slack function is defined by Chang et al. [3] as

$$slack_{avg}(clk) = \frac{1}{N} \sum_j \left\{ num_j) \cdot slack_j(clk) \right\}, \tag{2}$$

where num_j is the number of operations of type j; $slack_j(clk)$ is the slack on that operation for a clock clk; and N is the total number of operations in the DFG. The objective is to minimize the above slack function with respect to the value of the clock. All the points of local minima for the above average slack function are the possible values of the clock cycle in the search space. The average slack is evaluated for all these points and the clock length value corresponding to the minimum average slack can then be chosen as the clock value. This approach does not consider any constraints either on the availability of resources or latency. It also does not take the structure of the graph into account.

Another approach to finding the optimal value of the clock is described by Walker's research in [1] and [4]. The optimal clock selection technique as described in [1] considers all possible integral clock lengths which are ceilings of the integer divisors of the delays of the different functional units used in the design. This can be expressed as

$$\mathsf{C} = \left\{ clk \mid clk = \forall i, k \left\lceil \frac{delay_i}{k} \right\rceil \right\} \tag{3}$$

subject to the condition $clkmin \leq clk \leq \max(\forall_i(delay_i))$, where $clkmin$ is a technological limitation, a clock cycle lesser than this value is not practically feasible. The set C for the candidate clock lengths can be pruned further, i.e., the size of the set C can be reduced further by considering the slacks on the various functional units for a particular value of the clock. A slack vector $\vec{slack}(clk)$ that defines the slack for m functional units types for a particular value of clock clk can be written as

$$\vec{slack}(clk) = \{slack_1, slack_2, ..., slack_m\}. \tag{4}$$

If $\vec{slack}(clk) \leq \vec{slack}(clk')$[1], then the clk' can be replaced by clk. Hence, the reduction in the size of the initial set C to C'(say). For each $clk \in \mathsf{C}'$, the resultant set of clock values, the exact solutions for the RCS (or TCS) problem are found for various resource vectors and then the clk corresponding to the minimum schedule length (minimum resource requirement) is chosen as the optimal value of the clock.

Our approach to the problem is to take the structure of the sequencing graph into account in deriving a value for clock and to make design space exploration more efficient. Consider the *ARFILTER* benchmark [8] shown in Figure 2. This graph has 16 multiplication operations and 12 add operations. Chang et al.'s [3] solution for this graph is to zero out the slack of the multiplier since there are more multiplication operations compared to add operations in the graph. On the other hand, on any *critical path* (i.e., the length of the longest path in the graph [5]; one such path is highlighted in the figure), there are 3 multiplication and 5 add operations. Thus, zeroing out the slack of the adder reduces the total idle time on the critical path. Our results using this approach are very effective; experimental results shown in Section 4 demonstrate the efficacy of our solution.

3 Our Clock Selection Technique

3.1 Critical paths

The *slack minimization algorithm* presented by Chang et al. [3] considers a weighted average of the slack of all operations in the graph; but it does *not* take into account the structure of the graph. The critical path in the sequencing graph carries the essential dependency information. We would like to reduce the slack of the operation that has a higher frequency of occurrence in the critical path since reducing the delay in the critical path would improve the completion time of the schedule. The frequency of occurrences of operations belonging to different resource classes is considered and the slack of that particular functional unit is weighted with the frequency of occurrence of that particular resource class. All possible paths formed

[1] Given two vectors $\vec{a} = (a_1, \cdots, a_n)$ and $\vec{b} = (b_1, \cdots, b_n)$, we say $\vec{a} \leq \vec{b}$ if for all i, $1 \leq i \leq n$, $a_i \leq b_i$.

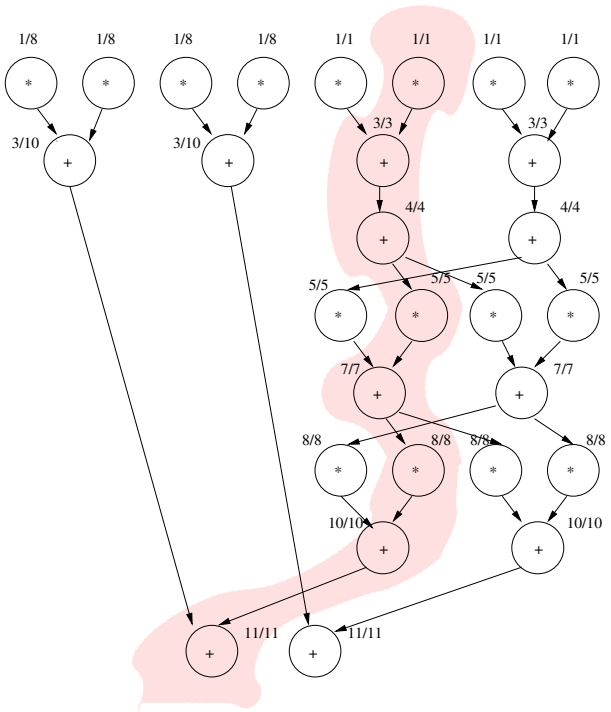


Figure 2: The sequencing graph for the *ARFILTER* benchmark and a critical path

by the critical nodes in the data flow graph are evaluated. The slack function is written as

$$Slack_{weighted}(clk) = \sum_{j} \left\{ freq_j^{critical} \cdot slack_j \right\} \tag{5}$$

where the term $freq_j^{critical}$ is the average of the fraction of the occurrence of operations belonging a particular resource class j over all critical paths in the data flow graph.

$$freq_j^{critical} = \frac{1}{P_c} \sum_{p=1}^{P_c} \left\{ \frac{\sum_j (num_j)}{N_{c,p}} \right\} \tag{6}$$

where P_c is the total number of critical paths in the data flow graph and $N_{c,p}$ is the total number of nodes on the p*th* critical path.

3.2 Generating candidate clock lengths

If the expression for slack is substituted in the above equation, then we have the following expression

$$\begin{aligned} & Slack_{weighted}(clk) \\ = & \sum_{j} \left\{ (freq_j^{critical}) \cdot ((clk \cdot \left\lceil \frac{delay_j}{clk} \right\rceil) - delay_j) \right\}. \end{aligned}$$

In the rest of this paper, we would refer to the above critical path weighted average slack as CPWslack. Our aim is to minimize the weighted slack function above and find the corresponding value of clock clk for which the function is minimum. If we consider a single term of the equation above as a function of the clock clk, then

$$f_j(clk) = K * \left\{ (clk \cdot \left\lceil \frac{delay_j}{clk} \right\rceil) - delay_j \right\}. \tag{7}$$

The above function is discontinuous at certain points which are governed by the term $\left\lceil \frac{delay_j}{clk} \right\rceil$. The discontinuous points are $\frac{delay_j}{k}$, where k is an integer. These are the points for which the slack function would reach a minimum value for a particular clock. All these points belong to the set of candidate clock lengths C. The function will be continuous between the points $clk = (\frac{delay_j}{k}, \frac{delay_j}{k-1})$. Consider some arbitrary value for $clk = \frac{delay_j}{k_r}$, where $k_r \in Z$ where Z is the set of reals and $k - 1 < k_r < k$. Then slack function is a straight line between the points $\frac{delay_j}{k}$ and $\frac{delay_j}{k-1}$ and the slope of the line is $\frac{\lceil k_r \rceil}{k_r}$.

3.3 The Activity Factor

For all possible $clk \in$ C, the value of the critical path weighted slack function is evaluated. This value of slack reflects the performance of the clock in absolute terms only. The absolute value of the slack does not convey the fraction of the useful clock. The major objective of using slack for clock selection is not only to select a clock which causes the least slack but also the slack which is relatively very less in comparison to the value of the clock. For this purpose, we define a new figure of merit called the *activity factor* to chose the best possible clock. Such a value of clock would reflect the maximum amount of useful work that can be done. We define the activity factor α for a given clock clk in general as the following

$$\alpha(clk) = \left\{ 1 - \frac{slack}{clk} \right\}. \tag{8}$$

In our definition of the slack, we will be using critical path weighted average slack, denoted by CPWslack(clk). The various candidate clocks can be pruned and the best clock selected using the activity factor, α. As an illustration, consider two clock values clk_1 and clk_2, such that $clk_1 > clk_2$. Let the critical path weighted average slacks for the two values of clocks be CPWslack(clk_1) and CPWslack(clk_1), such that CPWslack(clk_1) = CPWslack(clk_2). It is evident that the fraction, $\alpha(clk_1) > \alpha(clk_2)$, and hence clk_1 is a better value of clock than clk_2 because it reflects a better utilization of the available clock. Another important aspect is that for all values of the clock which result in an equal value of CPWslack, the highest value of the clock is always preferred. A higher value of the clock reduces hardware complexity and associated power dissipation in the circuit. The clock length causing the highest value of α is selected as the optimal clock.

The above is a scalar comparison of the activity factor values for different clocks. This is in contrast to the vector comparison over slacks for different types of resources for various clocks as discussed in [1].

3.4 Complexity of the algorithm

The complexity for the ASAP and ALAP schedules is known to be $O(V + E)$ [5]. Assuming that n is the number of candidate clock lengths, complexity of evaluating the activity factor is $O(n)$. The number of points (clock values) searched is

$$n = \sum_{i} \left\{ \left\lfloor \frac{delay_i}{minclock} \right\rfloor \right\} + 1, \tag{9}$$

where *minclock* is the minimum value of the clock length feasible.

The critical paths in the unconstrained scheduled graph or the sequencing graph are the longest paths from the source to the sink.

Table 1: Delays for functional units

FU_i	delay (ns)
multiplier	163
adder	48
subtractor	56

Table 2: Slack for different clocks

clk (ns)	mult (ns)	add (ns)	sub (ns)
163	0	115	107
82	1	34	26
81.5	0	33	25.5
56	5	8	0
55	2	7	54
54.33	0	6.33	52.66
48	29	0	40
41	1	7	26
40.75	0	33.5	25.5
33	2	18	10
32.59	0	17.19	9.19
28	5	8	0
27.16	0	6.33	25.5
24	5	0	16
23.28	0	21.85	13.85
21	5	15	7
20.37	0	13.12	5.12

Table 3: Slack results for *ARFILTER* for integral clocks

clk (ns)	CGslack (ns)	α_{CG}	CPWslack (ns)	α_{CPW}
163	49.45	0.70	71.87	0.56
82	15.19	0.81	21.62	0.74
55	**4.15**	**0.92**	**5.12**	**0.91**
48	16.53	0.65	10.87	0.77
41	15.19	0.63	21.62	0.47
33	8.88	0.73	12	0.64
28	6.29	0.77	6.87	0.75
24	**2.85**	**0.88**	**1.87**	**0.92**
21	9.3	0.56	11.25	0.46

Table 4: Slack results for *DCT* for integral clocks

clk (ns)	CGslack (ns)	α_{CG}	CPWslack (ns)	α_{CPW}
163	55.15	0.66	94.57	0.42
82	15.97	0.80	27.18	0.67
56	**5.27**	**0.91**	**6.17**	**0.89**
55	11.50	0.79	14.02	0.74
48	20.79	0.57	11.52	0.76
41	15.97	0.61	27.18	0.34
33	8.64	0.74	14	0.57
28	5.27	**0.81**	6.17	**0.78**
24	4.95	**0.79**	3.5	**0.85**
21	8.70	0.59	12	0.43

The problem of finding all the critical paths thus reduces to a problem of finding single source all possible longest paths in the original graph. However, in our approach, we consider only the edge disjoint paths formed by the critical nodes in the graph.

4 Experiments and discussion

In this section, we present experimental results of our technique used on five benchmarks: the AR-lattice filter $ARFILTER$ [8], the Elliptic Wave Filter EWF [15], the Discrete Cosine Transform DCT [12], the $FDCT$ [1] and the $F2$ [6] benchmarks. With each benchmark, the heuristic computes the weighted average slack function (abbreviated as CPWslack) and the activity factor, (α_{CPW}) for each value of the candidate clock lengths. First, we present results where chaining of operations is not considered. We used the VDP100 mudule library [10]; the delays of the operations from the VDP100 library that we used in our experiments are shown in Table 1. Table 2 shows the slacks of the different operations under various clock values. Tables 3, 4, 5, 6 and 7 show the performance of the heuristic. The values in boldface in each table show the best results. Note that these tables assume integral values for clock length. Our technique also works with non-integral clocks. However, the results for this have been omitted for lack of space.

For a comparison with the approach in [3], the values of the average slack over all operations abbreviated as CGslack and the corresponding activity factor α_{CG} have also been tabulated. In this paper, the slack on the operations is weighted by the frequency of operations in the critical path. Since the schedule length of a sequencing graph is dependent on the length of the critical path, the algorithm tries to minimize the slack on the operations in the critical path by zeroing out the slack on the most frequent operation. It assumes the availability of unlimited number of resources. This is in contrast to the approach in [1] where clock selection is based on exact scheduling under constraints. Our approach tries to extract some essential information from the structure of the DFG by emphasizing on the operations on the critical paths. A sample critical path in $ARFILTER$ is shown in Figure 2.

In addition to the heuristic for clock selection, we compared the effect of our clock selection with the approach in [1], which computes exact values of clocks using very expensive procedure. The optimal solutions have been evaluated for some typical resource vectors for two benchmarks $ARFILTER$ and EWF as shown in Table 8 and Table 9. The clock selected by our technique is shown in boldface. It can be noted that an optimal value of the clock for the least schedule length depends on the resource constraint and if sufficient resources are available, then the optimal value of the clock would approach the value computed in our approach as the best value of the clock computed by the heuristic assumes unlimited resources.

To illustrate this point, we compared our solutions to a full 3-D design space exploration used by Chaudhuri et al. [4]. Consider the exact RCS solutions in Table 8 and the heuristic solutions for integral clocks for the critical path weighted case in Table 3. The best value for clk found by the heuristic is $24ns$ (for maximum value of α_{CPW}). It can be seen from exact RCS solutions, in most cases, the same value of $24ns$ is the optimal result. The next best (or comparable) value of the clock in terms of the total schedule length (in ns) according to the heuristic is $55ns$ which is also the next best value as evaluated by exact RCS solutions. In contrast to this, if we consider the values for α_{CG} (based on the approach

Table 5: Slack results for *F2* for integral clocks

clk	CGslack	α_{CG}	CPWslack	α_{CPW}
163	83.95	0.48	76.65	0.53
82	25.09	0.69	23.00	0.72
56	7.19	0.87	7.00	0.87
55	**5.65**	**0.90**	**5.33**	**0.90**
48	7.83	0.84	9.66	0.80
41	25.09	0.39	23.00	0.44
33	13.68	0.58	12.66	0.62
28	7.19	0.74	7.00	0.75
24	**1.35**	**0.94**	**1.66**	**0.93**
21	12.30	0.41	11.66	0.44

Table 6: Slack results for *FDCT* for integral clocks

clk (ns)	CGslack (ns)	α_{CG}	CPWslack (ns)	α_{CPW}
163	68.82	0.58	77.19	0.53
82	18.98	0.77	20.64	0.75
56	**4.38**	**0.92**	**3.7**	**0.93**
55	19.67	0.64	26.42	0.52
48	23.42	0.51	27.23	0.43
41	18.98	0.54	20.64	0.50
33	9.44	0.71	9.74	0.70
28	**4.38**	**0.84**	**3.7**	**0.87**
24	6.86	0.71	8.73	0.64
21	8.72	0.58	8.61	0.59

Table 7: Slack results for *EWF* for integral clocks

clk (ns)	CGslack (ns)	α_{CG}	CPWslack (ns)	α_{CPW}
163	87.16	0.46	90.27	0.45
82	25.83	0.68	26.9	0.67
56	6.99	0.87	7.35	0.87
55	**7.19**	**0.87**	**5.92**	**0.89**
48	7.87	0.84	6.23	0.87
41	25.83	0.37	26.9	0.34
33	13.90	0.58	14.56	0.56
28	6.99	0.87	7.35	0.74
24	**1.63**	**0.93**	**1.07**	**0.95**
21	12.31	0.41	12.85	0.39

Table 8: *RCS* results without chaining for *EWF*

clk	1*,2+ ns (steps)	2*,2+ ns (steps)	3*,3+ ns (steps)
163	2608 (16)	2608 (16)	2282 (14)
82	1804 (22)	1476 (18)	1476 (18)
55	1650 (30)	1210 (22)	1210 (22)
48	1824 (38)	1248 (26)	1248 (26)
41	1804 (44)	1476 (36)	1476 (36)
33	1716 (52)	1320 (40)	1320 (40)
28	1680 (60)	1232 (44)	1232 (44)
24	**1632 (68)**	**1152 (48)**	**1152 (48)**
21	1722 (82)	1302 (62)	1302 (62)

in [3]) in Table 3, the value of $55ns$ is a better clock than $24ns$ since $\alpha_{\mathsf{CG}}(55) > \alpha_{\mathsf{CG}}(24)$. This is in contrast with the exact RCS solutions in Table 9 since the clock length of $24ns$ results in the fastest schedule.

Thus, the definition of average slack along critical path as well as relating the slack to the fraction of useful clock is a better measure for the selection of near-optimal clock length. Such an approach showed that the results obtained are no worse than the approach in [3].

4.1 Additional Issues

There are some additional engineering issues that affect the choice of clock length. Firstly, if the number of clock steps is large although the actual time to schedule is lesser, a larger and more complex controller might be required. Secondly, although we have based our approach of selection of a near-optimal clock on the activity factor, α, and the experimental results obtained compare very well with the exact RCS solutions as illustrated earlier, such an approach may not always yield the best practical value of the clock length.

Consider two clock lengths clk_1, clk_2 where $clk_1 > clk_2$; let the activity factors corresponding to these clock values be such that $\alpha(clk_2) > \alpha(clk_1)$ and the latency $\lambda(clk_1) > \lambda(clk_2)$. Although our method would choose clk_2 to be the optimal value of the clock, if the fraction, $\frac{\lambda(clk_1) - \lambda(clk_2)}{\lambda(clk_2)} \rightarrow \epsilon$ (ϵ being a very small value), and the number of control steps for scheduling using the clock value clk_2 is greater than the clock clk_1, then designing a controller for clk_1 will be better because of practical design issues like size and complexity of the controller.

We have looked at finding a near-optimal value of the clock length based on minimizing the critical path weighted average slack in a given time step and then relating this particular value of slack to the activity factor. Another important issue that affects clock selection related to *power dissipation* in a VLSI circuit is discussed in [14]. As the number of clock cycles increases, the clock distribution network in the circuit is repeatedly charged and discharged which leads to increased power consumption. Larger clock cycles may result in fewer state transitions leading to a smaller controller, and hence a reduction in controller power. On the other hand, larger clock allows for chaining of functional units and since their outputs can cause glitches, it results in increased glitching power consumption.

4.2 Summary of results

The trend of the results obtained from this method of clock selection compared quite well with the exact solution technique discussed in [1]. The clock selection methodology discussed in this paper also allows for non-integral clock lengths as opposed to the approach in [1] which formulates the problem for integral clock lengths only. For evaluating the performance of our approach, the delays of the various functional units used are shown in Table 1. The slack on different functional unit types for various candidate clock lengths are shown in Table 2. The highlight of our technique is the weighting of the slack function by the frequency of the occurrence of operations in the critical path in a data flow graph thus taking into account the structure of the graph in some form. We also define a term called activity factor, α, which can be thought of as the useful work done

Table 9: *RCS* results without chaining for *ARFILTER*

	1*,2+	2*,4+	4*,2+	6*,3+	6*,4+
clk	ns (steps)	ns (steps)	ns (steps)	ns (steps)	ns (steps)
163	2119 (13)	1630 (10)	1304 (8)	1304 (8)	1304 (8)
82	1640 (20)	1558 (19)	902 (11)	902 (11)	902 (11)
55	1595 (29)	1485 (27)	825 (15)	770 (14)	770 (14)
48	1776 (37)	1680 (35)	912 (19)	816 (17)	816 (17)
41	1640 (40)	1558 (38)	902 (22)	902 (22)	902 (22)
33	1650 (50)	1518 (46)	828 (26)	825 (25)	825 (25)
28	1624 (58)	1512 (54)	840 (30)	784 (28)	784 (28)
24	**1584 (66)**	**1488 (62)**	**816 (34)**	**744 (31)**	**744 (31)**
21	1659 (79)	1533 (73)	861 (41)	819 (39)	819 (39)

in a clock cycle. The method has also been extended to include two basic types of chaining. Using α as the figure of merit for optimal clock selection, we see from Table 8 and Table 9 that the clock value for which the total schedule time is the least is the same as the clock value for which the α is maximum. An important point can be highlighted by analyzing the results for $ARFILTER$ from table 3. The activity factor for clock value 24 ns (α=0.92) is marginally greater than that for 55 ns (α=0.91). From Table 9, it can be noted that for a resource constraint of $\langle 1*, 2+ \rangle$, the completion time (in ns) for the clock value of 24 ns is the least but the scheduling time (in ns) for a clock of 55 ns is only 11 ns greater, more importantly, the number of clock steps corresponding to a clock value of 55 ns is only 29 steps as compared to 66 steps for clock value of 24 ns. This makes 55 ns as a better value of clock from controller design point of view. In this respect, we can say that for two clock values clk_1 and clk_2, if $clk_1 > clk_2$, $\alpha(clk_1) \simeq \alpha(clk_2)$ and $\lambda(clk_1) - \lambda(clk_2) \rightarrow \epsilon$ (where ϵ is a small value as compared to the schedule lengths given by $\lambda(clk_1)$ and $\lambda(clk_2)$), then a higher value of clock length, clk_1, would lead to a lesser number of time steps. This is better in terms of minimizing the number of states for the finite state machine controller.

5 Conclusion

This paper presented a critical-path based heuristic for clock selection in high-level synthesis. Extensive experimental results on several benchmarks shows that our approach is very fast in practice and produces the optimal solutions in most cases. This would enable fast design space exploration that simultaneously considers area, latency, and clock length. Note that our goal here is to reduce the amount of computation required in the rest of the phases of design space exploration. Our appraoch considers only the operations on critical paths and does not consider the effect of the number of resources (also referred to as resource allocation). Based on the experiments with benchmark examples, we have found that the clock value determined by our technique is close to the optimal clock value. We are currently working on exploring the incorporation of the effects of non-critical operations and inlcuding the effect of resource allocation [11] (in particular through the use of bounds). Work is also in progress on extending our technique to include module selection, and to include the effects of the control circuit parameters.

Acknowledgments J. Ramanujam has been supported in part by NSF Young Investigator Award 9457768 and NSF grants 0073800 and 0121706. Mahmut Kandemir is supported in part by NSF CAREER award 0093082.

References

[1] Stephan A. Blythe and Robert A. Walker. Toward a practical methodology for completely characterizing the optimal design space. In *Proceedings of the 9th International Symposium on System Synthesis*, pages 8–13, La Jolla, CA, Nov 1996. ACM-IEEE.

[2] R. Camposano and W. Wolf. *High-Level VLSI Synthesis*. Kluwer Academic, 1991.

[3] En-Shou Chang, Daniel D. Gajski, and Sanjiv Narayan. An optimal clock period selection method based on clock slack minimization criteria. *ACM Trans. Design Automation of Electronic Systems*, 1(4):352–370, July 1996.

[4] Samit Chaudhuri, Stephan A. Blythe, and Robert A. Walker. A solution methodology for exact design space exploration in a three dimensional design space. *IEEE Transactions on Very Large Scale Integration*, 5(1):0–0, Mar 1997.

[5] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[6] Design Automation Group WWW Server. *Online Neat Sources: http://www.es.ele.tue.nl/neat*. Eindhoven University of Technology, pre-release edition, 1994.

[7] D. Gajski, N. Dutt, A. Wu, and S. Lin. *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic, 1992.

[8] R. Jain, A. Parker, and N. Park. Predicting area-time tradeoffs for pipelined design. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 35–41, Miami Beach, June 1987. ACM and IEEE Computer Society.

[9] Youn-Long Lin. Recent developments in high-level synthesis. *ACM Trans. Design Automation of Electronic Systems*, 2(1):2–21, Jan. 1997.

[10] S. Narayan and D.D. Gajski. System clock estimation based on clock slack minimization. In *Proceedings of the European Design Automation Conference*, pages 66–71, Hamburg, September 1992. IEEE Computer Society Press.

[11] A. Naseer, M. Balakrishnan, and Anshul Kumar. Optimal clock period for synthesized data paths. In *Proc. 10th International Conference on VLSI Design: VLSI in Multimedia Applications,* pp. 134–139, IEEE Computer Society Press, 1997.

[12] J.A. Nestor and G. Krishnamoorthy. Salsa: A new approach to scheduling with timing constraints. In *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pages 262–265, Santa Clara, November 1990. IEEE Computer Society Press.

[13] P.G. Paulin, J.P. Knight, and E.F. Girczyc. Hal: A multi-paradigm approach to automatic data path synthesis. In *Proceedings of the 23th ACM/IEEE Design Automation Conference*, pages 263–270, Las Vegas, June 1986. ACM and IEEE Computer Society.

[14] Anand Raghunathan, Niraj K. Jha, and Sujit Dey. *High-level Power Analysis and Optimization*. Kluwer Academic Publishers, 1998.

[15] D.E. Thomas, E.D. Lagnese, R.A. Walker, J.A. Nestor, J.V. Rajan, and R.L. Blackburn. *Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench*. Kluwer Academic, 1990.