Optimal Clock Period for Synthesized Data Paths

Abstract

The choice of a clock period in designs with multicycle operations have a major influence on operator allocation as well as execution time. For technologies with significant interconnection delays, optimal clock period selection before/during high-level synthesis is not practical. In our approach, we start with a synthesized RTL data path structure, perform place and route and back-annotate the interconnection delays. First a bound flow graph is constructed by reflecting the allocation and binding information on the data flow graph. All potentially critical paths in this bound flow graph are identified. Execution time is computed by evaluating these path lengths and thus avoiding rescheduling. Based on execution times, a set of potentially optimal clock periods is chosen. An optimal clock period is one which results in the minimum execution time while meeting a controller cost constraint. Finally, the controller costs at these clock periods along with the execution times decide the optimal clock period.

Extensive experimental results on data paths synthesized from high-level synthesis benchmarks establish both the utility as well as the efficiency of our approach. These results clearly show that choosing a clock period to minimize the "dead time" of the multicycle operators can improve the circuit performance by upto 10% or even more. Apart from presenting a methodology to decide the clock period, the report introduces a novel way of representing and interpreting binding information (operation-operator and value-register) which may have other interesting applications.

Contents

1 Introduction and motivation **1**

2 Overall approach **5**

2.1 Input 5

2.2 Design assumptions 5

2.3 Computing clock period 6

2.4 Dominated paths 8

2.5 Algorithm 9

3 Relaxing design restrictions **11**

3.1 Pipelined operators 11

3.2 Storage elements and binding 12

3.3 Interconnection delays 13

3.4 Chaining of operations 13

3.5 Behavioral descriptions with loops and conditionals 14

3.6 Modified dominating path condition 14

4 Experimental results **16**

5 Other applications **25**

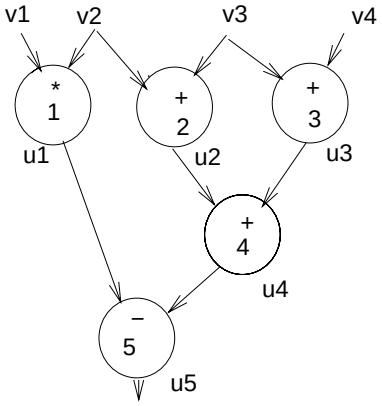
6 Conclusion and future work **26**

1 Introduction and motivation

Most of the early high-level synthesis (HLS) systems restricted all operations to single cycle and assumed that the clock period is decided post-synthesis by finding the critical delay path [1, 2]. More recent HLS approaches permit allocation of multicycle operators [3, 4, 5]. The critical path approach, though suitable for data paths with a single function unit(fu), is wasteful if there are a variety of fus with distinct delay values. As all fus would be clocked at the rate of the slowest fu, there is a lot of "dead time" (the time when the fu is not used) in all but the slowest of fus. This has prompted researchers to look into the problem of clock period selection as a pre-synthesis task [6], as a task integrated with component allocation [7] and a post-synthesis task [8, 9]. Naturally, the interconnection delays are considered only by those techniques which optimize clocks after synthesis.

Narayan & Gajski [6] estimate the optimal clock period by minimizing the average "dead time" of the operators. This technique ignores the actual allocation or schedule and gives equal weight to all operations as it finds the average on the basis of the number of operations of each type in the behavior. Further, it assumes only one type of operator is available in the library for each operation type. In the work of Gebotys [7], to limit the increase in complexity of the IP model due to variable clock period, a number of simplifications have been made. The possible clocks are limited to a few rather distant frequencies and even their selection is directly tied to the allocation of some operator through constraints. On the other hand, most of the post-synthesis work relates to estimating the critical path in the structure to get the minimum clock time while permitting operators to take multiple clocks. Mintz [8] estimate the clock cycle time using the structure, schedule as well as the interconnect delays. They use the schedule essentially to eliminate the false critical paths, i.e. paths though present in the structure but not really used by the schedule. As we also use the behavior along with the RTL structure, false paths are automatically eliminated.

In a recent work, Sri Parameswaran et al. [9] approach the problem of finding an optimal clock without changing the data path. Their work has some similarity to our work and it relies on rescheduling and resynthesizing the controller to identify the optimal clock period. They assert that the clock period which minimizes the execution time would always be an integer submultiple of state delays. They do not address the issue of computing the execution time (instead of rescheduling) with data path kept intact. Further, the approach seems to be limited to nonpipelined components.



(a) Data flow graph

Operator	Operations Supported	Delay (ns)
ADD	+	48
SUB	–	56
MUL	*	163

(b) Available operator set

Figure 1: A simple example DFG for illustration

A simple example illustrating the significant influence of the clock period on the execution time would provide motivation for our work. Fig. 1(a) shows a simple data flow graph which is synthesized using an operator library tabulated in fig. 1(b). Permitting operations to be performed in multiple clock cycles, Table 1 shows the number of clocks as well as execution time for clock period ranging from 20 to 35 ns. All execution times are based on an operator allocation of {1 ADD, 1 SUB, 1 MUL}. Fig. 2 (a to d) shows the schedules for the clock periods 20,24,28 and 35 ns respectively. It is interesting to note that even for this narrow range of clock periods, the execution times can vary by as much as 16% for the same operator allocation. Significantly, the approach based on clock slack minimization [6] (which is independent of the data path structure) selects 24 ns as the clock period instead of the optimal 28 ns.

Clock period	20	21	22	23	24	25	26	27
No. of clocks	12	11	11	11	10	10	10	10
Exe. time	240	231	242	253	240	250	260	270
Clock period	28	29	30	31	32	33	34	35
No. of clocks	8	8	8	8	8	7	7	7
Exe. time	224	232	240	248	256	231	238	245

Table 1: Execution time as a function of clock period

Based on an analysis of these results, it is easy to conclude that :

- Performance of the design measured by the total time taken (i.e no. of clocks $\times$ clock period) is not monotonic with clock period. A higher clock frequency does not mean a better performance.

- Optimal clock period depends on the source behavior and synthesized RTL structure, as well as the bindings (of operations to operators).

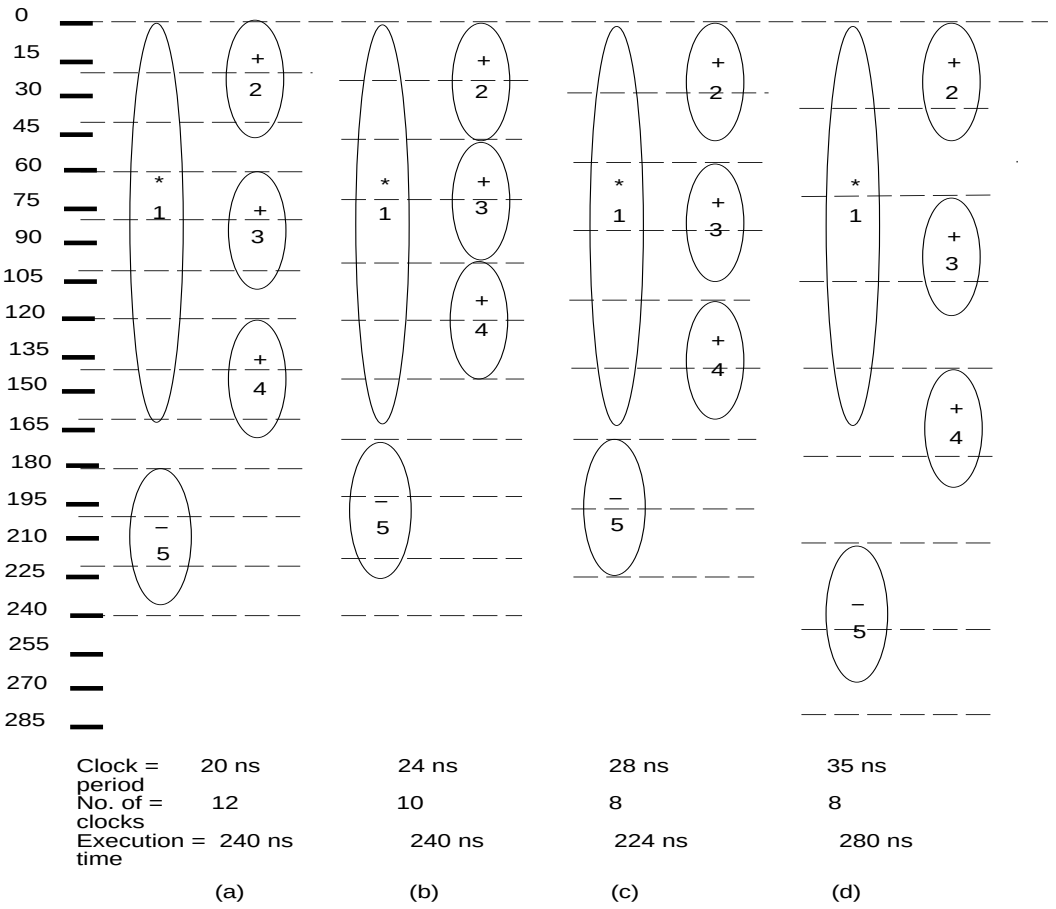


Figure 2: Schedule for different clock periods

In this report we present a novel approach to compute the optimal clock period for a synthesized data path. The difference in our approach is that the optimal clock period is tackled as a question of getting the best performance out of a synthesized structure. Similar to other post-synthesis clock estimators, we neither change the resource allocation nor the

bindings. But the major difference is that we do consider the change in the number of c_steps required by various operators with change in the clock period, which in turn, would affect the schedule (and execution time). The computed clock period is optimum as it takes care of the interconnection delays, either as an estimate from a model or from an actual layout tool. We also consider the controller cost which strongly depends on the number of states or c_steps in the schedule. The rest of the report is organized as follows. Section 2 describes the overall approach while section 3 lists some extensions for handling a range of data path structures and design styles. Results of experiments involving data paths synthesized from high-level synthesis benchmarks are tabulated in section 4. Section 5 discusses some other possible applications of the flow graph structure introduced in this report while section 6 presents some conclusions.

2 Overall approach

Before presenting our approach, some clarifications on the input as well as the design assumptions that are made are presented.

2.1 Input

The RTL structure generated by the synthesizer along with the input DFG and the binding information forms the input to our program. It would be preferable to perform the initial synthesis with as small a control step size as is feasible. A control step size closer to the gcd of "most" operator delays are expected to generate efficient structures as the dead time due to clock "quantization" is minimized. This control step period is only notional and need not be technologically realizable as a realizable clock period is chosen later. Our approach would work irrespective of the assumed control step size during the synthesis process but obviously cannot change non-optimal allocation and binding decisions made at that stage; it will still try to get an optimal clock period for the given structure.

2.2 Design assumptions

The following assumptions are made only for the sake of simplicity of presentation. Section 3 discusses modifications/extensions required to handle these situations/design styles and all assumptions, except the last one regarding synchronous nature of the design, would be relaxed.

- The operator modules are not pipelined.
- Register allocation and binding is ignored.
- Interconnections and storage elements are assumed to have a constant delay.
- No operation chaining is permitted.
- The design behavior consists of a single straight line code.
- It would be possible to synthesize the control part meeting the constraint of the selected clock period.

- The generated RTL structure is synchronous and operates on a single clock.

2.3 Computing clock period

The approach is based on computing the execution time required by a DFG for each of the feasible clock periods. Actually, the constraints imposed by the initial allocation and binding allow us to compute the execution time rather fast. Though this approach essentially deals with data path structure and operations performed on them, the control part plays a crucial role in deciding the optimal clock period. The range of clock periods is defined based on the feasibilty of synthesizing the control part with the specified clock period constraint. The evaluated or estimated control cost is used to make the final choice of the optimal clock period along with execution time.

We start with a DFG which is a directed acyclic graph with each edge from node u to node v representing the data dependency of node v on node u. We augment the flow graph with a start node α and an end node β. The start node α is connected to all nodes without predecessors and the end node β is connected to all nodes without successors (Fig. 3). The minimum possible schedule time of β is the execution time of the graph.
Each of the nodes is bound to an operator with its own delay. The maximum path length from α to any node u determines the earliest time at which u can be scheduled. But these paths ignore the binding information. For instance, in fig. 2(a), operation 3 could not be scheduled earlier than control step 4 not because of data dependency but due to resource constraints. As we intend to maintain the resource allocation and binding, we create additional edges in the DFG to indicate such dependencies. There is an operator dependency edge from node u to node v if both nodes are bound to the same operator and node u is the last operation scheduled on the operator before node v is scheduled (in the initial schedule). It should be clear from this discussion that we maintain the sequence of operations scheduled on the operator unchanged. Fig. 3(a) shows the new graph with both data dependency as well as operator dependency edges.

Each path consists of a set of nodes and each node is bound to one of the operators k with delay d_k. Thus, every path p_i is characterised by integer coefficients a_{ik} indicating nodes with delay d_k in the path. These coefficients would be referred to as delay coefficients. We assume there are K distinct delays indexed by k. We can enumerate all the paths from the start node α to the end node β. The minimum execution time for the graph is the maximum

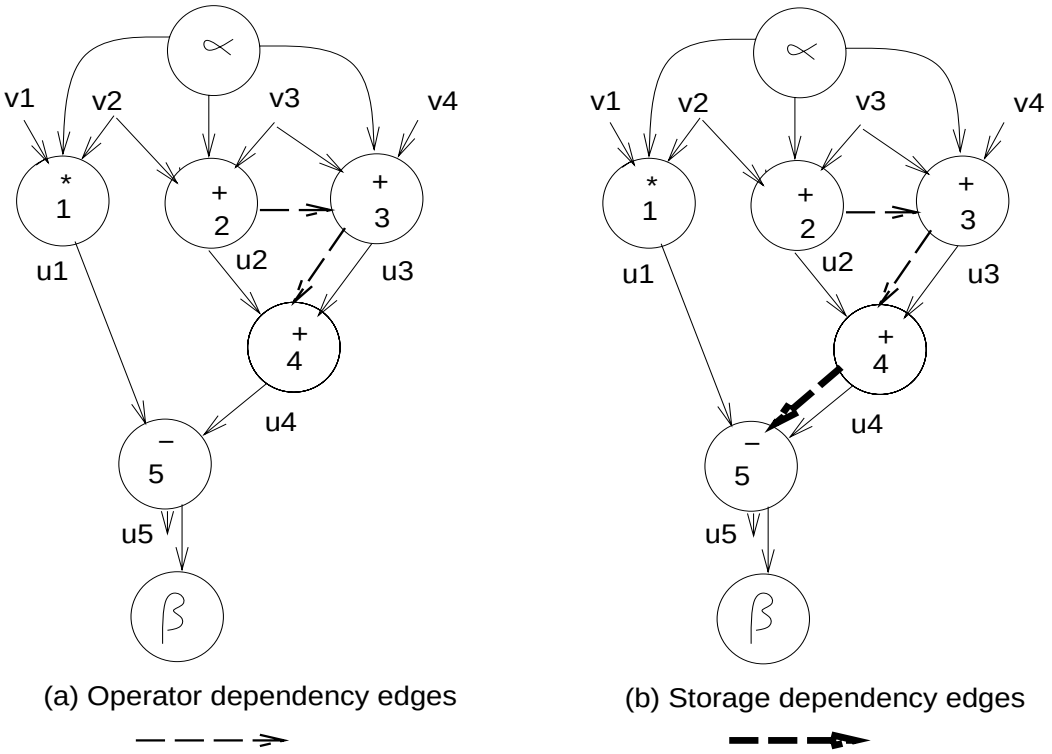


Figure 3: Resource dependency edges appended to the DFG

of all such paths to the end node β.

$$exe\_tim = max \sum_{p_\beta \ is \ a \ path \ to \ \beta} a_{\beta k} \times d_k \tag{1}$$

Now the clock is brought into the picture. The role of the clock is to quantize each of the delays d_k as an integral multiple of it. Thus for a clock with period l, the execution time is given by

$$exe\_tim_l = max \sum_{p_\beta \ is \ a \ path \ to \ \beta} a_{\beta k} \times \left\lceil \frac{d_k}{l} \right\rceil \times l \tag{2}$$

The optimal clock is the one which minimizes this time.

$$opt\_exe\_tim = \min_{all \ feasible \ clocks \ l} \{max \sum_{p_\beta \ is \ a \ path \ to \ \beta} a_{\beta k} \times \left\lceil \frac{d_k}{l} \right\rceil \times l\} \tag{3}$$

2.4 Dominated paths

As we are traversing the graph and enumerating the paths at each node, it is obvious that some paths are "dominated" by others i.e. they definitely cannot contribute to the maximum path length. To conserve memory, it is best to drop such dominated paths at each node.
A path p_i with delay coefficients a_{ik} dominating over path p_j with delay coefficients a_{jk} is denoted by $p_i \gg p_j$.

$$p_i \gg p_j \quad if \quad a_{ik} \geq a_{jk} \quad \forall \quad k = 1..K \tag{4}$$

The above condition is too stringent and we formulate another condition which covers the previous one. Without loss of generality, we can assume that the delays are arranged in the descending order implying $d_{k1} \geq d_{k2}$ if $k1 < k2$.

$$p_i \gg p_j \quad if \quad \sum_{l=1}^{k}(a_{il} - a_{jl}) \geq 0 \quad \forall \quad k = 1..K \tag{5}$$

The proof of this simply follows from the monotonicity of the ceiling operator i.e.

$$\left\lceil \frac{d_{k1}}{l} \right\rceil \geq \left\lceil \frac{d_{k2}}{l} \right\rceil \quad if \quad k1 \leq k2. \tag{6}$$

From this, it should be clear that we are interested only in differentiating delays which are distinct.

For the case shown in fig 3(a), the edges from node 2 to 3 and from 3 to 4 indicate that all three addition operations are performed on the same adder and in that sequence. The path delays for the two non-dominated paths are:

$$d_{p_1} = \{d_{MUL} + d_{SUB}\} \quad and \quad d_{p_2} = \{d_{SUB} + 3 \times d_{ADD}\}$$

Fig. 4 shows the number of clock cycles required by the two paths for the clock period in the range of 20 nsec to 60 nsec. It is clear that in certain ranges p_1 dominates (i.e. decides the minimum execution time) whereas in certain other ranges p_2 dominates.
A different allocation and binding could produce different results. If we allocate two adders instead of one and bind op2 and op3 to different adders, the two non-dominated path delays would be:

$$d_{p_1} = \{d_{MUL} + d_{SUB}\} \quad and \quad d_{p_2} = \{d_{SUB} + 2 \times d_{ADD}\}$$

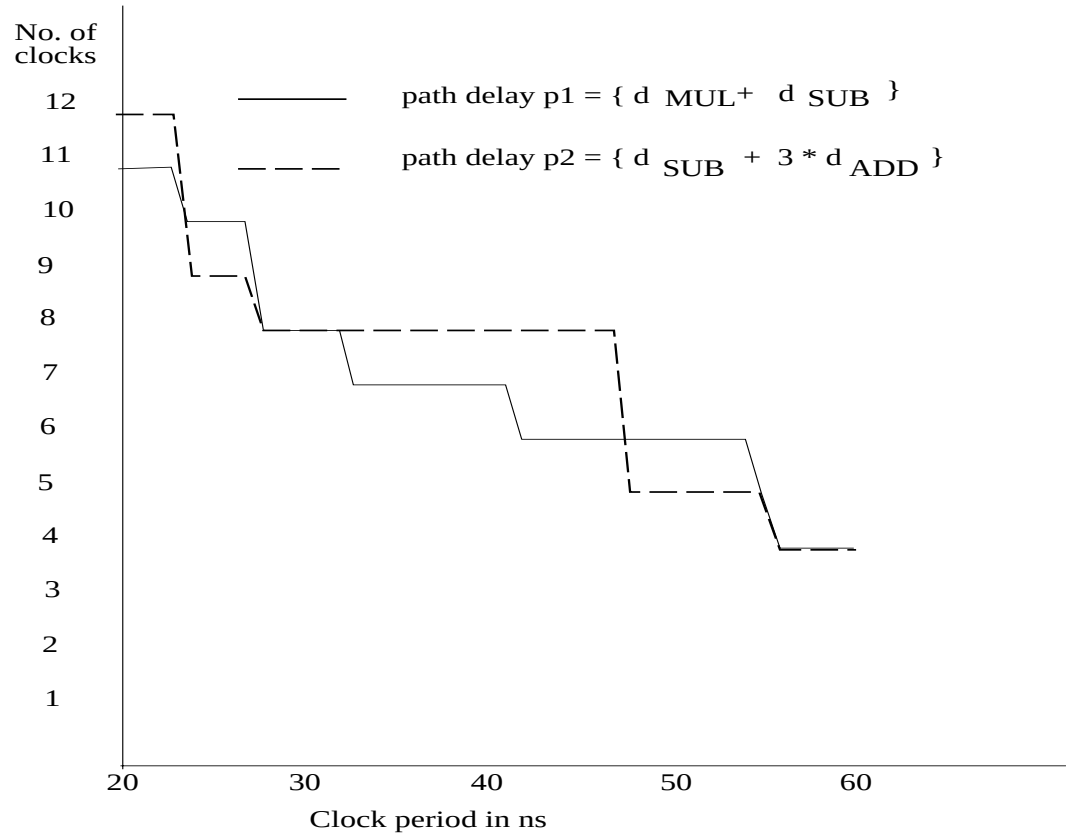


Figure 4: Clock cycles as a function of clock period for the two undominated paths

In this case, over the entire range of clock periods considered (i.e. 20 ns to 60 ns), p_1 dominates over p_2 and the optimal clock period is again 28 ns with 8 clock cycles. This implies that two adders do not produce a faster design if the clock period is optimally chosen. Thus, this approach can also have implications on operator allocation but this aspect is not explored in this report.

2.5 Algorithm

The major steps of the algorithm are listed in fig. 5. First the allocation and binding information is reflected on the DFG (step 1). This is followed by back-annotating the interconnection delays (step 2) and then identifying the non-dominated paths (step 3). The optimal clock period is identified in two steps (steps 4 and 5). Initially a set of potentially optimal clock periods corresponding to local minimas in the execution time vs. clock period graph are found. For each clock period in this set, control cost is evaluated (/estimated). The optimal clock period is selected based on both the control cost and the execution time.

Due to the fixed data path, the variation in the cost of the overall design comes only from the control part.

One of the key features of our approach is that execution time can be computed by evaluting the non-dominated path lengths and does not require any rescheduling (in step 4). Rescheduling is required only in step 5 for each of the selected clock periods to define the control signals [1]. Due to the presence of binding edges, rescheduling just involves a breadth first traversal from the start to the end node putting the operations into clock slots corresponding to the longest bound path. The time required for performing the core optimal clock period computation steps (1, 3 and 4) is fairly low in relation to steps 2 (place and route) and 5 (control cost evaluation). On the other hand, it is possible that the memory requirements grow exponentially in step 3 for keeping track of all the non-dominated paths. But as is clear from the experimental results, the pruning of dominated paths is very effective in reducing the memory requirements.

[1] Even this may be avoided with a refined control cost estimator.

ALGORITHM

- *INPUT : DFG, RTL Data Path Structure, Schedule, Binding*
- **Step 1:***Create a bound data flow graph by*
  - *Adding start and end nodes with their edges*
  - *Add the resource dependency edges*
- **Step 2:***Back-annotate interconnection delays*
  - *Perform place and route and extract interconnection delays*
  - *Associate delays with each node taking into account interconnection delays*
- **Step 3:***Traverse the bound DFG in breadth first manner and at each node*
  - *Enumerate all paths*
  - *Delete all dominated paths using conditions given by eq. 5*
- **Step 4:***Compute the set of potentially optimal clock periods by*
  - *Evaluating eq. 2 over all feasible clocks for all the non-dominated paths*
  - *Choosing the clock periods with the locally minimal execution times*
- **Step 5:***Compute the optimal clock period by*
  - *Evaluating (/estimating) the control costs for the clock periods selected in step 4*
  - *Choosing the clock period with the appropriate execution time-control cost tradeoff*

Figure 5: Optimal clock computation algorithm

3 Relaxing design restrictions

3.1 Pipelined operators

The use of pipelined operators in the design result in the following two changes in the methodology presented in the previous section.

- Normally all delays associated with operator binding edges for operator k are taken to be d_k. In case k is a pipelined operator, this delay is taken as $p \times l$ where p is the initiation interval and l is the clock period (to be calculated).
- The operator itself may restrict the minimum feasible clock period due to the critical path delay within any of its stages.

3.2 Storage elements and binding

Our approach is equivalent to rescheduling for each clock period while maintaining the initial allocation and binding. In case storage elements are ignored, it is possible that the rescheduled graph may require extra storage elements and thus invalidate the original storage allocation and binding. One option is to model the storage elements as resources with *Write/Read* operations on them. A simpler approach involves adding storage dependency edges which carry only the necessary binding information. Further, most structures have fewer variations in the delays of storage operations (i.e. setup and hold times) and thus there is no need to distinguish different storage elements. In case such variations in delays do exist (like a mix of registers and register arrays), it is taken into account by lumping it with interconnection delays as described in the next subsection.

Consider two operations u and v [2] whose results are stored in the same register R. If u precedes v in the original schedule, it is expected that all the operations using the result of u from R would have finished using this value before v stores its own result in R. This is reflected by creating storage binding edges from all nodes dependent on u to v. Fig. 3(b) shows a storage binding edge created as both u_2 and u_5, outputs of op2 and op5 respectively, share the same register R. The edge from u_4 to u_5 is expected to ensure that op5 does not get scheduled so early that op4 has not as yet finished using its input. Of course, many of these edges are redundant (and could be removed by transitive closure) and generate only dominated paths.

The delays associated with these storage dependency edges are more complex and consist of two components :

- One is from the source node i.e where the edge originates
- The other is from the destination i.e. where the edge terminates

At the destination, the associated delay is always one clock period (irrespective of the destination operator) because we only want to ensure that the destination operation should not finish earlier than one cycle until after the source operation has finished using the register. Now as far as the delay on the source side is concerned, it depends on the type of operator

[2] In this case we have to consider the primary inputs as well if they share storage with results of other operations

as one is concerned with when it finishes using its operand values. The delay values for different operator types are summarized in Table 2.

Source operator Type	Delay added to the path
Purely combinational	Operator delay
Pipelined operator	Initiation interval of the operator as multiples of clock period
Registered operator	1 clock period

Table 2: Delays due to storage binding edges

3.3 Interconnection delays

The synthesized structure can be placed and routed and consequently the interconnection delays can be obtained. Fig. 6 contains the partial structure (related only to the add operations) with delays marked on each interconnection as well as path elements like multiplexer. An analysis of these interconnection delays can generate the delay of the individual operations. Fig. 6 also shows a sample binding of values to registers and lists the delay values for the three addition operations bound to the same adder.

Just to control the complexity, all the feasible operations with associated delays on each operator are generated and categorized into a few distinct delays ignoring very small variations. Then each operation bound to this operator is classified as having one of these delay values. The delays introduced by storage elements (setup/hold time) can be accounted for here even if they have substantial variations.

3.4 Chaining of operations

Chaining of operations in the original scheduled graph is handled by creating operations with sum of delays of the chained operations. Further, operator dependency edges cover all operators used by any of the chained operations as none of these operators can be used in the same clock cycle. Chaining also has an implication on the minimum feasible clock cycle as the designer does not want to spread the chained operations across multiple clock cycles.

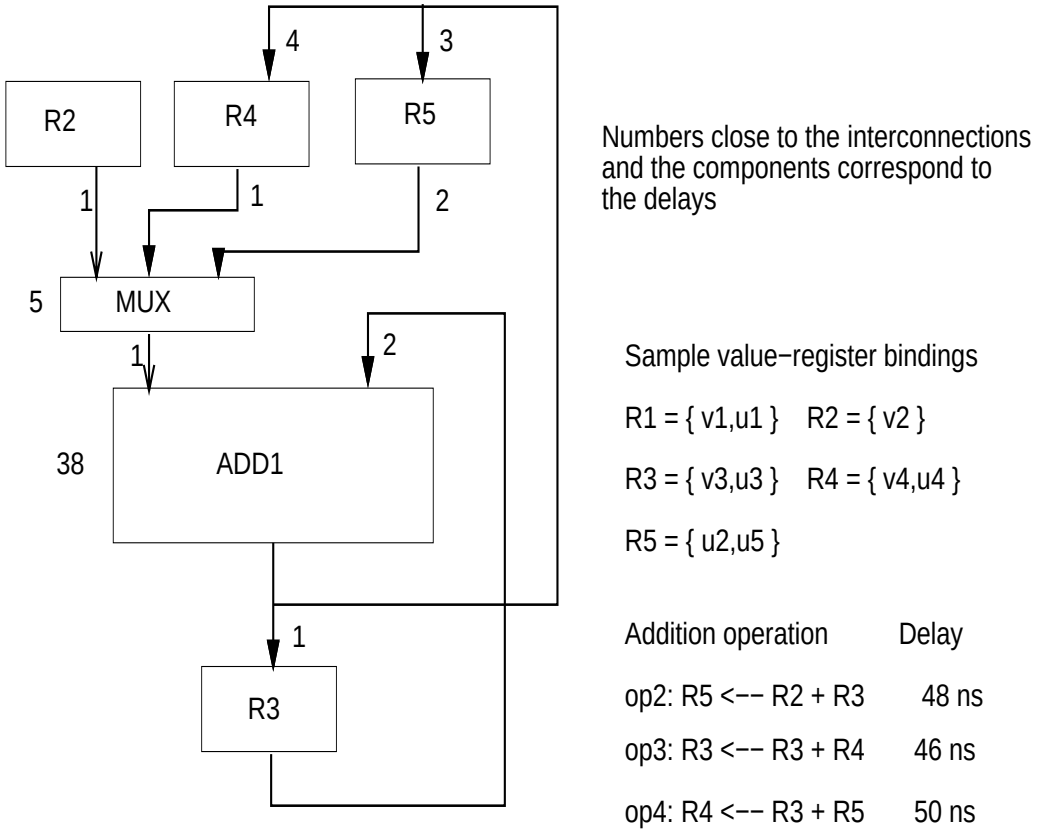


Figure 6: Partial data path structure with interconnection delays

3.5 Behavioral descriptions with loops and conditionals

The approach is extendable to any general behavioral description with conditionals and loops. It is assumed that the behavior has been profiled to generate average (/maximum) loop counts and conditional branching probabilities. The one restriction that is placed on scheduling is that the overlap of operations across basic blocks is not permitted i.e. all multicycle operations in a block have to be completed before initiating operations in the succeeding block. This permits computation of overall execution time as a weighted sum of individual basic block execution times. Further, the identification of dominating paths can be handled locally at the block level.

3.6 Modified dominating path condition

Introduction of binding edges resulting in multiples of clock period delays (pipelined operators as well as storage dependencies) require the conditions for path domination to be modified. As the clock period is undecided at the time of graph traversal, a path can be

eliminated only if it is dominated over the entire feasible clock period range. Assume the clock period has a range $< l_1, l_2 >$ and k1 is the highest index with $d_{k1} \geq l_1$ and k2 is the smallest index with $d_{k2} \leq l_2$. Now each path p_i is characterized by a set of K delay coefficients $\{a_{ik}\}$ and a clock period coefficient a_{il}. For path p_i to dominate over p_j, the additional condition to be satisfied is given by:

$$p_i \gg p_j \quad if \quad \sum_{p=1}^{k}(a_{ik} - a_{jk}) \geq (a_{il} - a_{jl}) \quad \forall \quad k = k1..k2 \tag{7}$$

It is obvious that this condition trivially holds as a subset of the previous condition (eq. 5) if $a_{il} \geq a_{jl}$.

4 Experimental results

This technique has beeen implemented as a backend tool for a RTL structure to FPGA mapper and is named OPTICLE (OPTImal CLock Evaluator). A data path synthesizer [10] generates a RTL netlist which is directly mapped onto FPGAs [11]. Here we present results from applying these techniques on some well-known HLS benchmarks. In each case, the schedule and the RTL structure generated by a data path synthesizer was given as an input to OPTICLE and an optimal clock period was computed. As interconnection delays are very significant in FPGA implementations, this is a suitable platform to show the utility of our approach.

First, some remarks about the operator set, tools and the platform. The implementation technology was XILINX XC4000 series [12]. All functional units were considered to be 8-bit wide. The multiplier was based on a CSA (carry-save-adder) implementation whereas a library component (from XBLOX) was used for performing addition, subtraction and comparison. Delays of the adder/subtractor and multiplier operators (without interconnections) were 20 ns and 204 ns respectively. For place and route as well as delay computation, proprietory XILINX software was used. All the CPU times are based on a SUN SPARC classic workstation.

Table 3 summarizes the inputs. Each of the three HLS benchmarks namely Differential equation, Elliptical filter and AR filter, have been processed with different initial control steps. This corresponds to the number of c_steps in the schedule generated by the data path synthesizer. The third column lists the operators as well as the registers in the RTL structure.

The results are discussed with reference to the steps of the algorithm (fig. 5). Table 4 gives details of the CPU time as well as the efficiency of the algorithm in pruning the dominated paths. Columns 1 & 2 specify the example. The pruning is quite effective as the ratio of total paths (column 3) to non-dominated paths (column 6) is as high as 390. Columns 4,5,7 & 8 refer to the steps 3 and 4 in the algorithm. Column 8 entries vis a vis column 5 entries show that the reduction in cpu time for optimal clock period computation is substantial and thus the pruning procedure is effective.

Table 5 shows the result of step 4 of the algorithm for the AR filter example with initial control steps as 8. Results of optimal clock period computation by ignoring the interconnection delays (columns 2-5) and with the interconnection delays (columns 6-9) are tabulated.

Benchmark example	Initial control steps (ics)	Resource Allocation	
		Operators	Registers
Differential Equation	4	$\{1+, 1<, 2*, 1-\}$	7
	7	$\{1+, 1<, 1*, 1-\}$	6
Elliptical Filter	14	$\{3+, 2*\}$	9
	15	$\{3+, 1*\}$	8
	16	$\{2+, 1*\}$	8
AR Filter	8	$\{2+, 4*\}$	10
	13	$\{1+, 2*\}$	10
	18	$\{1+, 1*\}$	10

Table 3: Details of benchmark examples

Example	ics	Without pruning dominated paths			With pruning dominated paths		
		No. of paths	CPU time(in secs.)		No. of paths	CPU time(in secs.)	
			Step III	Step IV		Step III	Step IV
Differential Equation	4	13	0.3	0.1	7	0.4	< 0.1
	7	37	0.3	0.2	7	0.4	< 0.1
Elliptical Filter	14	1856	0.9	15.3	17	0.6	< 0.1
	15	5942	1.4	41.7	47	1.0	0.1
	16	6250	1.5	46.2	16	0.7	< 0.1
AR Filter	8	174	0.4	0.7	16	0.5	< 0.1
	13	4078	1.1	25.2	18	0.5	< 0.1
	18	3146	1.1	24.7	12	0.5	< 0.1

Table 4: CPU Times and algorithm efficiency in pruning dominated paths

Clock range	Without interconnection delay				With interconnection delay			
	Clock period	Control steps	Longest path	Execution time	Clock period	Control steps	Longest path	execution time
20-100	23	39	876	897	30	46	1333	1380
30-100	34	27		918	30	46		1380
40-100	41	23		943	45	31		1395
50-100	51	19		969	54	26		1404
60-100	68	15		1020	90	16		1440
70-100	70	15		1050	90	16		1440
80-100	80	15		1200	90	16		1440
90-100	90	15		1350	90	16		1440

Table 5: Optimal clock period for different clock period ranges: AR Filter (ics = 8)

Different rows give optimal values for different clock period ranges. The results of this example are plotted in fig. 7 as a graph of execution time vs. clock period. The two plots correspond to execution time without interconnection delays (lower plot) and with interconnection delays (higher plot). It is clear from this example that though the nature of plots in the two cases is similar, the optimal values could be significantly different. Thus any clock period estimation ignoring the interconnection delays would not even be close to the optimal. Consider the clock period range of 50 to 100 ns in Fig. 7. The optimal clock period in this clock range in the lower plot is 51 ns (without interconnection delays) which is close to the local maxima in the upper plot (with interconnection delays). Such a choice would result in atleast a 10% lower performance than the optimal which is at 54 ns.

Figures 8 and 9 show similar plots for the differential equation (ics = 7) and elliptical filter (ics = 16) examples respectively.

Table 6 summarizes the results for the eight cases. In all cases, the feasible clock period range was considered to be 20 to 100 ns. Though in step 4 of the algorithm, a set of clock periods is generated for each case, only the clock period corresponding to the global minima is tabulated. Columns 4 and 3 show the optimal clock period with and without the interconnection delays respectively. It is significant to note that interconnection delays do not always increase the optimal clock period. Columns 5, 6 and 7 give details of the optimal res-

Benchmark example	ics	Optimal clock period		Control steps	Longest path delay	Execution time
		w/o inter. delays	with inter. delays			
Differential Equation	4	23	49	17	814	833
	7	34	27	61	1603	1647
Elliptical Filter	14	21	20	103	1977	2060
	15	23	27	97	2529	2619
	16	23	25	110	2692	2750
AR Filter	8	23	30	46	1333	1380
	13	23	25	108	2639	2700
	18	34	23	200	4521	4600

Table 6: Optimal clock period and execution time

ult with the interconnection delays. The number of control steps could be a rough estimate of the control cost as it corresponds to the number of states in the controller. The difference between execution time (column 7) and the longest path (column 6) reflects the "wastage" due to clock quantization even in the optimal case. detailed results for all the cases (with local minimas and execution times) are shown in Table 8.

Finally, we present some results on the controller cost. It is expected that changing the clock period would have a significant influence on the controller cost due to the change in the number of control steps/states. The control signals required by the data path are the outputs of the controller. Instead of realizing the controller as a general state machine, we implemented the state transitions with a counter. As all our examples require a large sequence of simple state transitions, this approach was more cost effective. The logic for the outputs was optimized using SIS (Berkeley tools).

Table 7 shows the control cost for the clock periods which correspond to local minimas in fig. 7. Table 8 shows consolidated results on six cases enumerating execution times and controller costs for all selected clock periods. All times are in ns and costs are in XC4000 CLB counts. Step 5 in the algorithm is a time consuming step as controller costs are being evaluated with logic synthesis rather than with a fast estimation technique. This table shows

Clock period	Number of control steps	Execution time	Controller cost in XC4000 CLBs
30	46	1380	13
34	41	1394	12
45	31	1395	12
54	26	1404	11
68	22	1496	10
90	16	1440	10

Table 7: Controller cost : AR Filter (ics = 8)

the controller cost to be rather well behaved as a function of the number of states. However, it is not so in most cases as is evident from Table 8. In each case with the data path being fixed, the number of controller input/outputs are fixed. Further, as the same behavior is being effectively rescheduled even the number of transitions on these control signals do not change significantly. In spite of this the controller cost with the same number of i/os is not a monotonic function of the number of states. Thus cost estimation models based only on the number of states, input/outputs and their transition counts are not applicable. To speed up this step in the algorithm, we do need to develop more refined models which are applicable to our target technology.

Clock period	Control steps	Execution time	Controller cost	Clock period	Control steps	Execution time	Controller cost
Differential Equation							
ics = 4, Longest path = 814				ics = 7, Longest path = 1603			
27	31	837	8	27	61	1647	10
35	24	840	7	30	55	1650	11
49	17	833	8	40	42	1680	13
50	17	850	8	45	38	1710	12
61	14	854	6	53	32	1696	13
80	12	960	7	67	26	1742	9
81	11	891	7	70	26	1820	9
93	10	930	7	89	19	1691	11
Elliptical Filter							
ics = 14, Longest path = 1977				ics = 15, Longest path = 2529			
20	103	2060	39	27	97	2619	32
32	66	2112	32	30	89	2670	38
48	44	2112	32	54	50	2700	26
50	43	2150	27	67	42	2814	20
60	38	2280	28	70	42	2940	20
100	23	2300	24	92	32	2944	22
AR Filter							
ics = 13, Longest path = 2639				ics = 18, Longest path = 4521			
25	108	2700	22	23	200	4600	24
32	86	2752	26	31	151	4681	32
48	59	2832	19	46	101	4646	21
56	50	2800	24	55	84	4620	29
70	42	2940	17	69	68	4692	18
96	30	2880	19	92	51	4692	23

Table 8: Consolidated results on execution time and controller costs

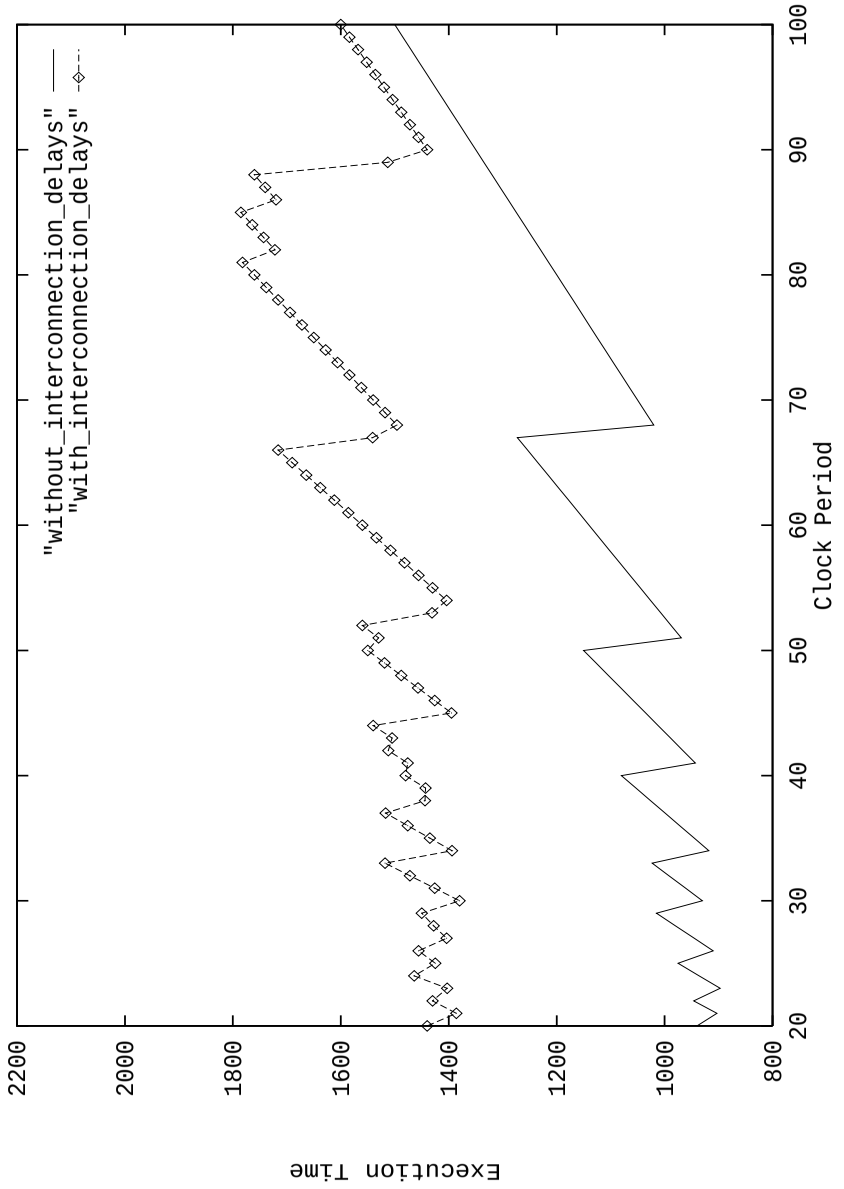


Figure 7: Execution time vs. clock period : AR Filter (ics = 8)

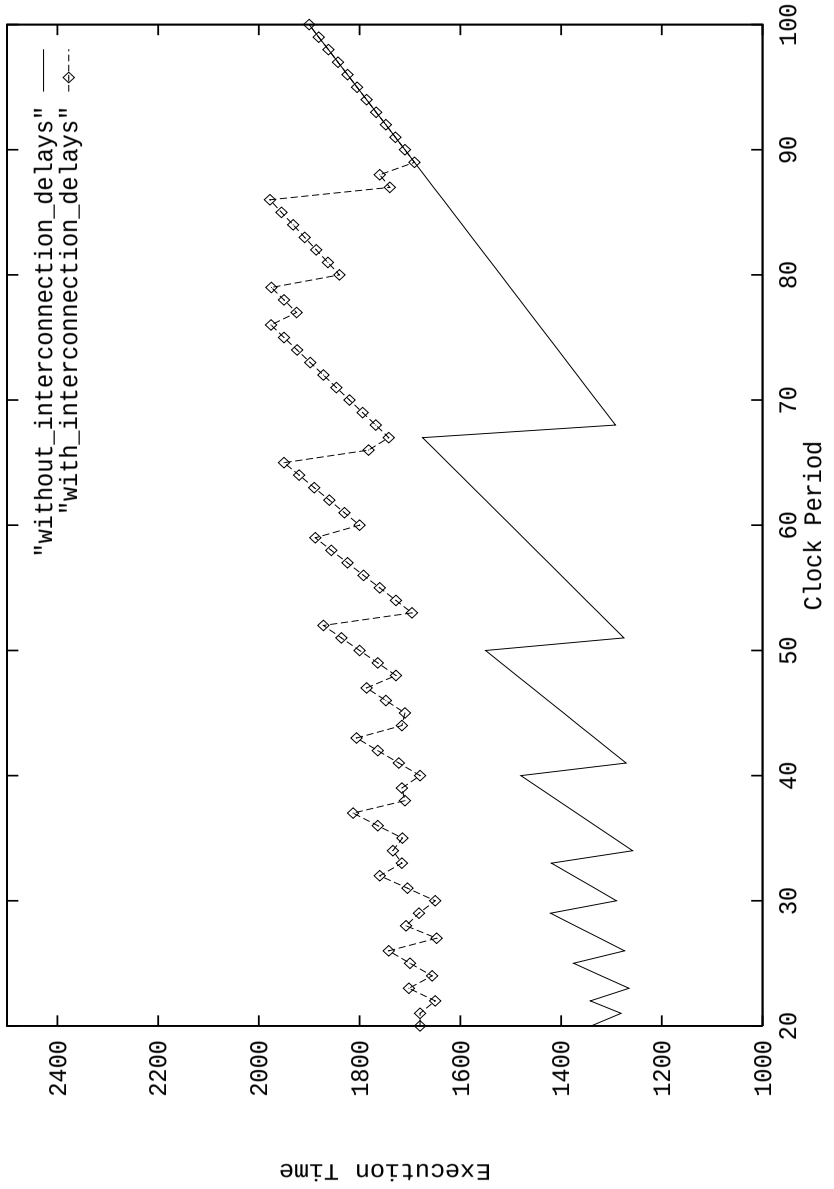


Figure 8: Execution time vs. clock period : Differential Equation (ics = 7)

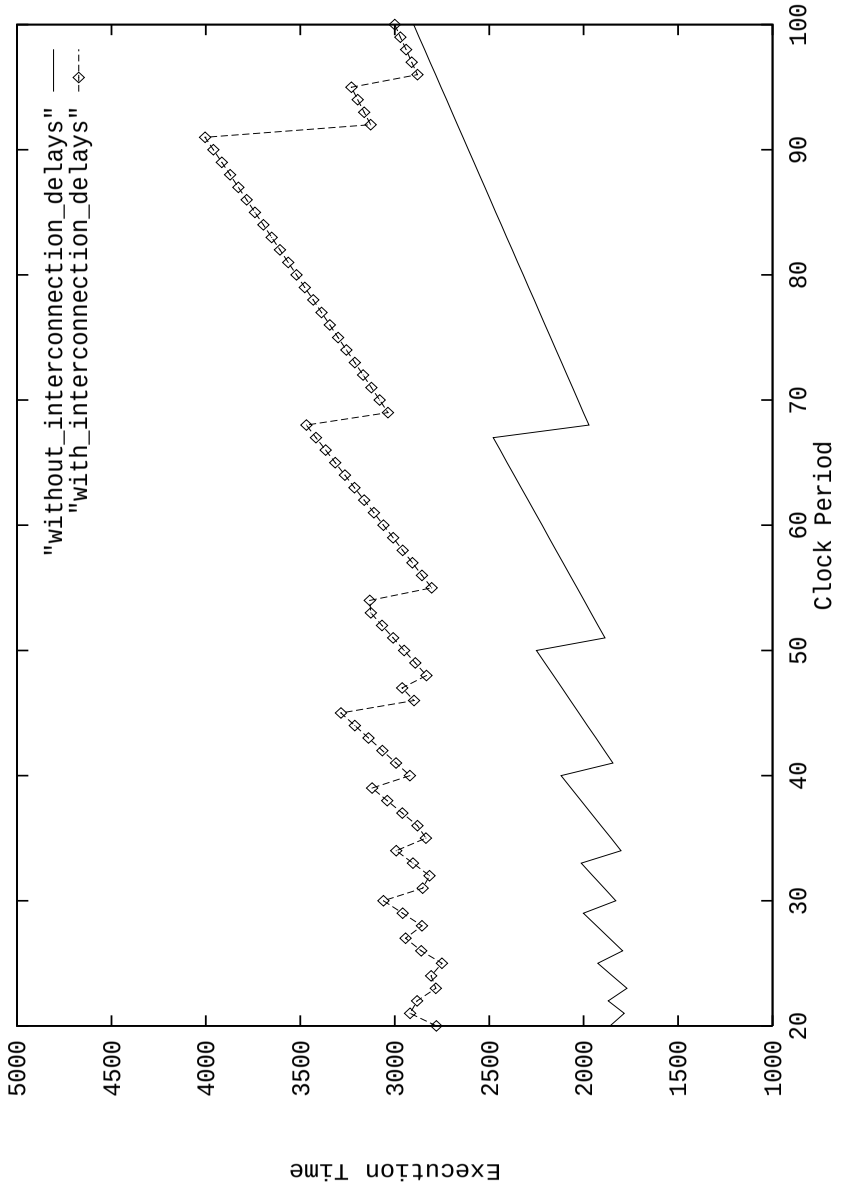


Figure 9: Execution time vs. clock period : Elliptical Filter (ics = 16)

5 Other applications

In presenting the clock period selection problem, we have also presented a novel graph structure which we refer to as bound control data flow graph (*bcdfg*). In *bcdfg*, the data and control edges of a control-data flow graph are augmented with resource dependency edges which carry the allocation and binding information. Further, we have presented an interpretation of these binding edges, in the context of scheduling, for a range of operator types. These *bcdfg*s can have many other applications and we briefly discuss one of them.

All high-level synthesis techniques have to partition a real life description before synthesis. Once these parts are being synthesized separately, there is a need to communicate allocation or binding information of one part to the other. This is important because the allocation is global and one would like to optimize the use of resources across these parts. *bcdfg* can carry partial binding information to meet these requirements e.g constraints posed by a pipelined multiplier allocated in Part A to be reused in Part B can be reflected by modifying the flow graph of B.

6 Conclusion and future work

We have proposed an efficient method for computing the optimal clock period for a specified data path. The approach is based on back-annotating the interconnection delays and then computing path lengths of all the "potentially" critical paths over the feasible range of clock periods. Experimental results on RTL structures generated from HLS benchmarks show both the utility as well as the efficiency of our approach. It is clear from these results that even though the number of non-dominated paths is not large, we still need to consider multiple paths. It has been shown that a clock period selected ignoring interconnection delays could be inefficient by 10% or even more. Permitting multicycle operations can result in a large increase in the number of states and thus it is inappropriate to choose a clock period ignoring the controller cost. Presently, we select the optimal clock period by evaluating controller costs at a set of potentially optimal clock periods.

In future, we also propose to look at the problem of developing models to estimate controller costs rather than evaluate them through synthesis. This would considerably speed up this step (step 5 : fig. 5) in the algorithm. The previously reported models for estimation were found inadequate for our purposes. Further, even CLB estimates for combinational logic have been reported only for XC2000 and XC3000 series of XILINX devices. Thus, a more versatile controller cost estimator meeting our technology requirements is needed. The clock period also has a strong influence on power consumption and thus for low power applications, a power estimate should be a part of the objective function for optimal clock period selection. Again, the existing power estimation models might have to be modified to cater for this situation. As the data path is fixed, a faster clock does not induce proportionately more transitions on most of the data path signals. The impact of the clock period change is primarily (though not solely) on the generation of control signals.

Acknowledgements

We would like to acknowledge Prof. Peter Marwedel for his comments and suggestions on the first draft of this report. Further, we would like to acknowledge the Ministry of Human Resource Development (MHRD), Govt. of India for supporting Mr. A.R. Naseer under the QIP programme.

References

[1] N.Park and A.C. Parker, *Synthesis of Optimal Clocking Schemes*, DAC-22,1985,pp.489-495.

[2] A.C.Parker et. al, *MAHA: A Program for Datapath Synthesis*, DAC-23, 1986,pp.461-466.

[3] P.G.Paulin and J.P.Knight, *Force-Directed Synthesis for the Behavioral Synthesis of ASIC's*, IEEE Trans. on CAD, June 1989, pp. 661-679.

[4] M. Rim et. al, *Optimal Allocation and Binding in High Level Synthesis*, DAC-29, 1992, pp.120-123.

[5] C.H.Gebotys and M.I.Elmasry, *Global Optimization Approach for Architectural Synthesis*, IEEE Tran. CAD, Sep. 1993, pp. 1266- 1278.

[6] S. Narayan and D.D. Gajski, *System Clock Estimation based on Clock Slack Minimization*, EURO-DAC 92, pp.66-71.

[7] C.H. Gebotys, *Optimal Scheduling and Allocation of Embedded VLSI Chips*, DAC-29, 1992, pp. 116-119.

[8] D.Mintz and C. Dangelo, *Timing Estimation for Behavioral Descriptions*, 7th Intl. Symp. on High-Level Synthesis, May 1994, pp.42-47.

[9] S. Parameswaran, P. Jha and N. Dutt, *Resynthesizing Controllers for Minimum Execution Time*, Proc. of 2nd Asia Pacific Conf. on HDL Standards and Appls., Oct. 1994.

[10] M.V.Rao et.al, *Design Space Exploration of RT-Level Components*, VLSI-Design 93, Jan. 1992, pp. 299-303.

[11] A.R. Naseer, M. Balakrishnan and Anshul Kumar,*An Efficient Technique for Mapping RTL Structures onto FPGAs*, Proc. of FPL '94, LNCS - 849, Springer Verlag, Prague, Czech Republic, Sep. 1994, pp.99-105.

[12] The Programmable Logic Data Book, XILINX, 1994.