Rapid Estimation of Control Delay from High-Level Specifications

Gagan Raj Gupta
Dept. of Computer Sciences
Univ. of Wisconsin
Madison, WI 53706
gagan@cs.wisc.edu

Madhur Gupta
School of ECE
Purdue Univ.
West Lafayette, IN 47907
guptam@purdue.edu

Preeti Ranjan Panda
Dept. of Comp. Sc. & Engg.
Indian Institute of Technology
New Delhi 110016
panda@cse.iitd.ac.in

ABSTRACT

We address the problem of estimating controller delay from high-level specifications during behavioral synthesis. Typically, the critical path of a synthesised behavioral design goes through both the datapath and the control logic; yet most scheduling algorithms account only for datapath and ignore control delay, leading to timing uncertainties in the resulting designs. We present an estimation technique for computing a fast, robust, scalable, and reasonably accurate approximation of the control delay from behavioral specifications. The delay estimate is formulated in terms of the properties of the input specification and other inputs to the synthesis process such as resource constraints.

Categories and Subject Descriptors

B.5.2 [**Register-Transfer-Level Implementation**]: Design Aids—*Automatic synthesis*

General Terms

Performance, Experimentation

Keywords

High Level Synthesis, FSM, Control Delay, Estimation

1. INTRODUCTION

The productivity advantage of behavioral synthesis comes from being able to capture design specification at a higher level of abstraction and finding an automated path to a detailed implementation. The output of this step is a datapath netlist and a controller or Finite State Machine (FSM) that controls the cycle-by-cycle behavior of the datapath through signals such as MUX-select, register load, etc.

Figure 1 illustrates the typical output of behavioral synthesis. Control signals for selecting MUX inputs and loading registers lead from the controller to the datapath. In this generated circuit, the critical path usually begins at the state register outputs, and passes through the control logic, followed by datapath elements, and terminates at datapath registers. Thus, for correct operation of the circuit, the clock period needs to be wide enough to accommodate all these cumulative delays along with the setup and hold times of the registers. Since the scheduling step determines the mapping of datapath operations to clock cycles, and takes important timing-related decisions such as chaining of operations depending on the resource delays, the *effective* clock period available for the datapath operations needs to be known to the scheduler. This clock period would be obtained by subtracting the setup/hold times and control delay from the actual clock period. Unfortunately, this is a circular problem – the FSM is known only *after* the schedule is completely determined. If we wait until synthesis and timing analysis is performed on the FSM, it is too late to influence the schedule. Depending on the size of the circuit synthesized, the FSM delay can be significant enough to invalidate the schedule completely if the control delay is ignored. If we attempt a re-schedule of the behavior using the FSM delay observed from the first pass, the schedule itself would change, thereby invalidating the FSM delay itself. This is a variant of the classical timing closure problem in VLSI design, manifested at the behavioral level of abstraction.

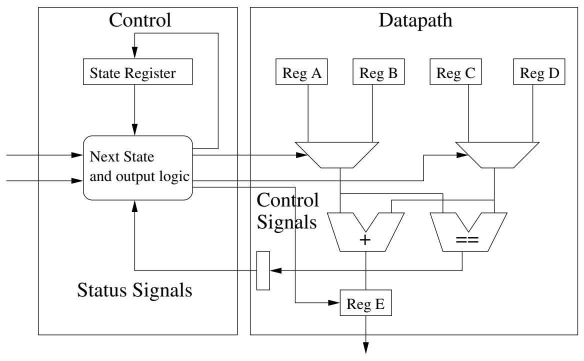


Figure 1: Output of Behavioral Synthesis

2. PREVIOUS WORK

Variants of the problem outlined above have been recognized and addressed by HLS researchers in the past. In [1], a synthesis model is presented, which spans the domains of high-level and logic synthesis. In [7], the authors take the approach of fixing the datapath template in advance, with the synthesis process resulting only in determining the programming sequence of a memory-based controller. Such approaches can reduce timing uncertainties in HLS, but will usually sacrifice the performance and area efficiency that is associated with custom hardware controllers.

The problem of estimating controller area from high level descriptions has been addressed by [4, 6], with an FPGA target architecture, based on information present in each step of the HLS process. Controller area estimation in an ASIC context was also addressed in [5], but they require the state table, which is only available after all the HLS steps have completed. Other work addressing timing issues for high-level synthesis [3, 2] build a timing model for a given schedule and estimate clock time from the critical path.

In commercial behavioral synthesis tools such as Synopsys Behavioral Compiler and SystemC Compiler, the designer is expected to give an estimate of the FSM delay (otherwise the tool assumes a default value such as 10% of clock period). Unfortunately, it is not clear how the designer is supposed to arrive at such a number. To be safe, the designer may give a large pessimistic value, resulting in very sub-optimal schedules. This is a significant drawback of the methodology.

In this paper, we handle control delay through an estimate drawn from the behavioural specification itself that is fast, accurate, and scalable. The advantage of this approach over the previous techniques is that there is no need to be pessimistic in the control delay formulation; yet the margin of error is reasonably low. The estimate is adaptable to various high level synthesis infrastructures, algorithms and constraints. Our major contribution is the result that reasonably accurate control delay estimates can actually be obtained from high-level parameters extracted from the specification, in spite of the control delay estimation problem being very difficult because of the presence of a large number of heuristic steps in the overall synthesis procedure.

3. FACTORS AFFECTING CONTROL DELAY

Our control estimation strategy is based on empirical formulas drawn from detailed experimental observation of the control delays for different types of specifications. The experimental methodology is illustrated in Figure 3. A specification generator reads the behavioral parameters to be investigated and generates a C-style specification which forms an input to behavioral synthesis. The generated specification is taken through Behavioral Synthesis steps resulting in a Datapath and FSM in VHDL. The FSM is then synthesised with Synopsys Design Compiler and the critical path delay through the gate-level netlist is observed. For different parameters, the delays are plotted and curves are fit to the data.

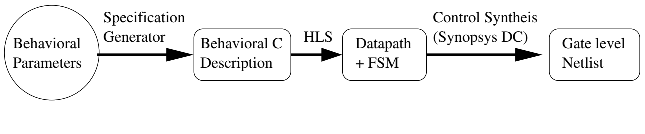


Figure 2: Experimental Methodology

The path delay through the controller can be affected by various properties of the input specification. We have enumerated some of them below, with an analysis of the extent of influence.

3.1 Number of operations

It is expected that as we increase the number of nodes in the DFG by increasing the number of operations in the specification, the complexity of the controller should increase. Figure 3(a) shows that the delay varies logarithmically with the number of operations in the input program, leading to an initial delay formulation of the form:

$$\text{Delay} = A \times log(\text{number of operations}) \qquad (1)$$

where the co-efficient A possibly depends on other characteristics of the input program. The RMS error of the curve fit was 9.26%.

The number of states in a resulting schedule is a linear function of the number of operations (assuming the number of resources is a small constant) and since we used binary encoding for the states, the number of flip-flops used in the controller would be approximately equal to log(number of states). Also, the depth of the circuit generated from the synthesis of the combinational logic would be logarithmically related to the number of input signals. These input signals include the state bits and the status signals.

One interesting behavior of the delay data was that, with the increase in the number of operations, the delay increased gradually in certain intervals, while there was a steep increase at the end of the intervals – the data pattern resembled a staircase. We conjectured that this pattern was formed because of the increase in the number of state register bits, which increases the number of input bits to the control logic, and hence, impacts the delay. As binary encoding was used for states, the jumps in the staircase pattern are expected at those points where the number of states crosses a power of two, requiring an extra storage bit. Using this insight, we refined our earlier formulation to:

$$\text{Delay} = A \times log(\#\text{operations}) \times (\#\text{state bits}) \qquad (2)$$

The curve fitted with the new equation is shown in Figure 3(b), where we study the delay variation over a larger range (up to 1200 operations). We observe that the proposed equation is able to capture the discrete jumps occurring due to the increase in the number of state bits. The RMS error of the curve fit was 10.32%. To study the effect of the number of state bits, we rescheduled the same set of specifications with an increased resource constraint (= 5 resources instead of 1), which reduces the schedule length (and hence, the number of states) while the other properties of the specification are maintained. The corresponding curve is shown in Figure 3(c). We note that the delays are smaller, which is correctly captured by the curve. The RMS error was 14.26%.

The actual number of state bits is available only after the complete schedule is finalized, when the schedule length is known. However, since we would like to perform the delay estimate without invoking the detailed HLS steps, we use a quick estimate of the schedule length by running, for each basic block, a simple list scheduling algorithm that uses a random priority function, to reduce computation time during design space exploration. We have found this to be adequate in our experiments, both in terms of the minimal run-time overhead, as well as the estimate for the number of states.

3.2 Multiple Latency Operation Types

Since resources with different latencies may impact the controller differently (the inputs need to be held stable over multiple clock cycles), we included multi-cycle operations in our study. In our first experiment, we generated programs with only multiply operations, and used two multipliers with a 2-cycle latency as resources. The conclusion was that the same formulation can be used to predict delay for programs with multi-cycle operations as well.

In real specifications, there will be, in general, a mix of resources with different latencies. Figure 3(d) shows the delay plots for input specifications using operations with two different latencies: addition (single-cycle) and multiplication (2-cycle), where the additions and multiplications are in ratio 80:20. One ALU and one multiplier are used as the resource constraint. The general observation is that the same earlier formulation suffices, with appropriate changes being made to the state bits parameter. We observed that A increases with number of resources, but remains the same for the same resource constraints, irrespective of other changes in the input descriptions. The conclusion is that, in the delay formulation, we can essentially ignore the fact that resources have different latencies – the effect of multiple latency operations is only via the increased number of states in the schedule, which is already captured.

3.3 Number of basic blocks

The presence of high level control structures such as *if, switch*, loops, etc. leads to increasing number of basic blocks. This has an effect on the FSM structure as it usually translates to branches in the FSM. We performed a number of experiments using programs with different number of basic blocks, with varying nesting levels of the conditional constructs. However, we observed no significant difference in the control delays due to these variations. The delays predicted by Equation 2 worked equally well for these programs. Of course, the effect of the complex control structures does reflect in the schedule length estimate, which in turn impacts the number of state bits and hence the delay.

3.4 Number of variables and resources

An increase in the number of global variables results in an increase in the number of registers as they are expected to be active throughout program execution. However the local variables are active only within a block and the corresponding registers could be reused. The size and the number of the MUXes can increase because of the increase in the number of variables. We have found experimentally that for most cases the controller delay does increase with increasing number of variables.

The circuit structure resulting from a given number of resources is dependent on the degree of parallelism available in the input specification. If this degree is less than the available resources, then some resources may never be used; consequently, the extra resouces will not impact the controller complexity. If the degree is more, then MUXes will be inferred at the inputs of the resources due to sharing. We worked with variable resource constraints and studied the effect of different degrees of parallelism on the controller's complexity.

The co-efficient A of our formulation seems to depend on two parameters: the number of resources, and the number of variables. We conducted two types of study: (i) varied resources keeping the same number of variables; and (ii) varied the number of variables keeping the same resource constraints. The variable count was varied between 20 and 50. The resource constraint was varied between 1 and 10. Figures 3(e) and 3(f) show two snapshots of this study. A comparison of the two figures shows that the control delay tends to increase with increasing resources (because of the increased number of FU control signals to be generated) as well as increasing variables (because of the increased number of register control signals to be generated).

3.5 Designs with memory

The impact of memory modules on the controller is somewhat different from the impact of individual registers. Increasing the number of registers leads to increasing complexity of the FSM because the register load signals need to be generated by the FSM. However, designs with memory are scalable, i.e., the control complexity does not increase with size – the read/write/enable signals have to be generated at the appropriate cycle irrespective of memory size. Thus, we treat memory as just another resource.

3.6 Overall Formulation

The overall formulation is that, given a CDFG of a specification, we first count the number of nodes representing operations. Then we perform an approximate list schedule to estimate the number of states. We finally compute the value of the co-efficient A in the delay equation using a relationship of the form:

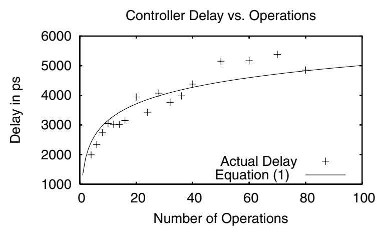
$$A = x + y \times (\#\text{FU control bits}) + z \times (\#\text{Variables}) \quad (3)$$

The number of FU control bits is essentially a dependency on the resource constraints – control bits refer to function selection inputs expected by certain resources such as ALUs. The values of x, y, and z are specific to a given ASIC library and can be empirically obtained by first running a set of synthetic examples such as the ones we have constructed for our experiments and using a linear curve fit. We must indicate that the equation for A is dominated by x – in other words, A is close to a constant.

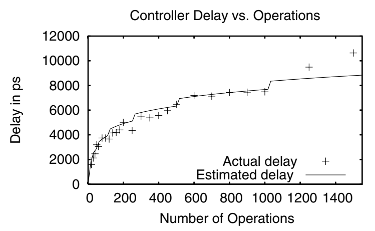
4. VALIDATION

After building our formulation on control delay estimation using synthetic examples generated from our specification generator, we validated the formulation on benchmark examples taken from different sources. The generated FSMs from high-level synthesis were synthesised using Synopsys Design Compiler with a 0.13 micron ASIC library.

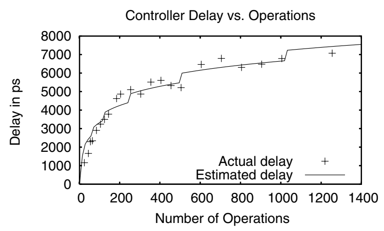
Table 1 shows the prediction results of the controller delay on the benchmark examples (Column 1) we studied. The second column shows the value of co-efficient A computed from the x, y, z values derived from the ASIC library, and the resource and variable details. We have used the values $x = 100, y = 0.6, z = 0.1$. The third and fourth columns show the estimated and actual delays in picoseconds. The last column shows the relative error in the prediction. The average error was 6.7%, with the estimation run-time being 2-3 orders of magnitude faster than actual synthesis. We observed that these errors are much smaller than the RMS errors observed with the synthetic benchmarks with



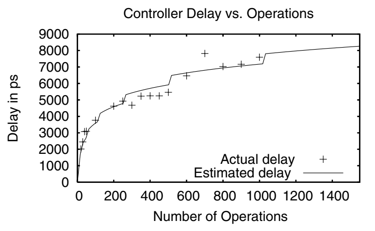
(a) Delay (Eqn 1)



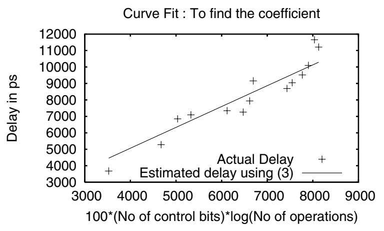
(b) Delay (Eqn 2). 1 resource.



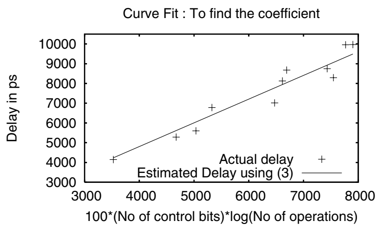
(c) Delay (Eqn 2). 5 resources.



(d) Mixed latencies: 20%-80%



(e) 20 vars. Res: 5 (+), 5 (*)



(f) 20 vars. Res: 8 (+), 8 (*)

Figure 3: Curve fit into delay data. Only representative graphs are shown

Benchmark	*A*	*Est (ps)*	*Actual (ps)*	*Err (%)*
GCD	103.0	1630.9	1879.1	13.2
FIBO	103.9	2249.0	2273.5	-7.7
FACT	104.2	1373.7	1330.4	-3.2
Histogram	110.4	5198.0	4716.5	-8.9
Bubble	106.5	2979.0	2935.9	-1.5
Newlife	108.0	5419.8	5088.0	-6.5
Jfdct	109.4	4663.0	4506.0	-3.5
Jidct	110.8	6723.8	6392.0	-5.2
ADPCM	110.0	4841.4	4482.0	-8.0
EPIC	111.7	5393.2	4970.9	-8.5
CRC32	109.2	4175.4	3887.6	-7.4
Blowfish	108.7	3643.3	3898.6	6.5

Table 1: Delay estimate for designs

randomly generated input specifications. When the ASIC library changes, the constants (x, y, z) used in the formulation will change. Our proposed overall methodology is that, for a new library, our set of synthetic examples, consisting of a large number of different input programs with varying parameters, is run to train the estimation into computing these constants. We are then ready to perform the estimates on a given input program.

5. CONCLUSION

We have presented a technique to estimate control delay from high level design specifications. Estimation of controller delay from behavioral specifications is inherently a very difficult and challenging problem because there are many heuristics and approximations used in the synthesis flow and there could be a non-trivial dependence on the actual algorithms and heuristics used in the tools. The delays could be very different even on designs with similar complexity. Thus, we consider it an important contribution of this paper that our formulation is able to capture the delays within only a 6.7% error even though we use a very simple equation framed in terms of very generic parameters. Our formulation uses empirically determined parameters x, y, and z, which can be tuned to a particular ASIC library. They can also be tuned to a particular application class, such as multimedia, which might improve the predictability. The simplicity of the estimation methodology makes it attractive to incorporate in the design space exploration phase, where speed of evaluation is essential. The fast estimate leads to a more informed scheduling strategy – now the scheduler can subtract the control delay when making decisions on the effective clock period available for datapath operations.

6. REFERENCES

[1] R. A. Bergamaschi. Bridging the domains of high-level and logic synthesis. *IEEE Trans. on CAD*, May 2002.

[2] V. Chaiyakul, et al., Timing models for high-level synthesis. In *EDAC*, 1992.

[3] A. Kuehlmann and R. A. Bergamaschi. Timing analysis in high-level synthesis. In *ICCAD*, 1992.

[4] C. Menn, et al., Controller estimation for FPGA target architectures during high-level synthesis. In *ISSS*, 2002.

[5] B. Mitra, P. R. Panda, and P. P. Chaudhuri. Estimating the complexity of synthesized designs from FSM specifications. *IEEE D&T*, Jan. 1993.

[6] A. Nayak, et al., Accurate area and delay estimators for FPGAs. In *DATE*, 2002.

[7] M. Reshadi and D. D. Gajski. A cycle accurate compilation algorithm for custom pipelined datapaths. In *CODES+ISSS*, 2005.