Incorporating Interconnection Delays in VHDL Behavioral Synthesis [1]

Yee-Wing Hsieh, Steven P. Levitan & *Barry M. Pangrle
Department of Electrical Engineering, University of Pittsburgh
*Department of Computer Science, Pennsylvania State University

Abstract

Advances in VLSI technology have provided smaller feature sizes which allow the integration of large circuit designs into a single chip. As a result, the interconnection delay between functional units is now comparable to the functional unit delay itself, and can no longer be ignored by high-level synthesis tools. In this paper, we present a library based behavioral synthesis tool that incorporates interconnection delay, based on accurate estimates of the final circuit, into the behavioral synthesis process. The tool, called TinkerTool, integrates a behavioral synthesis tool, SandS, into the Keystone VHDL design environment. TinkerTool uses a design iteration scheme to address the interdependency between the synthesized design and estimates of the critical path delay.

1 Introduction

Recent advances in sub-micron integrated circuits provide chip designers a means to pack very large circuit designs into a single chip using very small, fast gates. First-order MOS scaling theory, based upon the "constant field" model indicates that as the geometries of the transistor are scaled by $1/\alpha$, the transistor density scales up by α^2 which results in current density scaling linearly with α [WE85]. This implies that densely packed structures require wider metal conductors, which increase the RC constant of the wiring interconnects. Furthermore, for a fixed chip size, signal path length across the chip, as a rule, does not scale down. This results in constant line response time, insensitive to scaling. Consequently, the gap is narrowing between the time it takes to compute values, the functional unit delay, and the time it takes to communicate results, the interconnection delay. Therefore, any high-level synthesis tool that does not consider interconnection effects as part of its timing model may not be able to synthesize realistic designs.

However, accurately estimating interconnection delay requires a knowledge of the floorplan and the interconnection patterns in the final layout of the synthesized architecture. On the other hand, synthesizing realistic architectures requires the knowledge of the "loaded" delay characteristics of the functional units for scheduling and allocation[MPC88]. This is a vicious cycle, since each process depends on the outcome of the other. The floorplan comes from the synthesis process, and the delay estimates needed for good synthesis, must come from the floorplan.

To overcome this interdependence, we have developed an architectural synthesis tool which uses an iteration scheme to obtain realistic designs. With this scheme each design iteration is based on a performance estimation from the architecture synthesized in the previous iteration along with

[1] This research is supported by the National Science Foundation under Grant MIP-9101656

a rough interconnection delay estimation based on simplified floor planning. Using this iteration scheme, the unrealistic portions of the design space are quickly eliminated which allows the user to focus on the realistic portions of the design space and evaluate the synthesized design using low-level synthesis tools. In this way, we eliminate the inefficiencies of evaluating every design using low-level synthesis and simulation tools.

This paper presents our work on TinkerTool, a library based behavioral synthesis tool that incorporates interconnection delay into the behavioral synthesis process. TinkerTool, integrates a behavioral synthesis tool, SandS[PP92], into the Keystone VHDL design environment. In the next section we briefly present related research on systems which also incorporate delay models in behavioral synthesis. In section 3 we present an overview of TinkerTool, followed in section 4 with a discussion of our delay models. Section 5 presents the details of the iteration scheme used for converging on valid designs; and section 6 presents some experimental results from iterating on several designs with TinkerTool. In section 7 we give our conclusions.

2 Related Research

Many high-level synthesis tools produce designs that use the number of buses and multiplexors as the only measure of quality and do not consider the effects of the interconnect and the delay in the controller[PBLS91]. Consequently, these high-level synthesis systems assume output loading and intrinsic component delay as well as control delay to be negligible or zero. This assumption is valid only if the number of interconnections is small, the length of interconnect is short and the intrinsic component delays are relatively small compared to the functional unit delays. Similarly, control delay can be ignored only if the delay in the controller is relatively small compared to functional unit delays. However, as the complexity of a design grows the interconnect and the state controller grows as well and thus this assumption is no longer valid. Therefore, a synthesis system must consider output loading and intrinsic component delay as well as control delay in order to synthesize realistic designs.

There have been several high-level synthesis systems that consider physical implementation effects in high-level synthesis. BUD [McF86] uses bottom-up design methodology in behavioral synthesis by considering physical design effects through floorplanning estimates. BUD has the ability to incorporate various low-level physical design information into the synthesis process. However, BUD was designed with a single data path implementation style and handles only simple "unit time" operations. Finally, BUD does not consider the effect of control delay during synthesis.

A second system, 3D scheduling[WP91], performs placement of modules and scheduling of operations simultaneously. However, it is unclear, in this system, how wiring delays directly influence scheduling. The prototype presented in [WP91] does not consider the cost or the delay of registers, multiplexors, wiring space or control. This is a problem since interconnection delays, which include register delays, multiplexor delays, bus delays, and wiring delays, may cause some operations in the synthesized system to be on the critical path which were not anticipated by the design system.

A third system, Chippe, uses a "knobs" and "gauges" [BG86] approach to behavioral synthesis. In this research, an expert system automatically selects resource constraints, knobs based on its quality measurements evaluated from previous synthesized designs, against the design goals of

area, time and power [BG90]. However, Chippe does not perform floorplanning. Rather, Chippe estimates wire length based on the total area of the functional units, not their physical dimensions.

For our system, critical path delay is one of the criteria used to determine when to terminate iteration through the design space. If the critical path delay is greater than the cycle time of the synthesized architecture, then that architecture may not be valid, and thus, further iteration is required. If critical path delay is less than or equal to the cycle time then the current architecture is valid, and thus further iteration is not required.

We note that static critical path analysis is insufficient, because false paths will overestimate the critical path delay. For any given state, a functional unit, multiplexor, register etc. may not be operating and thus any path through that device would be a false path. In addition, a functional unit which executes over a number of cycles contributes only a fraction of its total delay to the critical path delay (total delay of function unit / number of cycles). To use the total delay of a multi-cycled functional unit in the critical path delay calculation would grossly overestimate the critical path delay. Consequently, critical path delay must be evaluated dynamically based on the current state of the state controller. Therefore, in TinkerTool an exhaustive search over all possible state configurations is used for the determination of the critical path delay. This gives us a more accurate estimate of the true critical path delay and increases the likelihood that the synthesized architecture will operate correctly within the required cycle time.

We describe the operation of TinkerTool in the next section and discuss the design iteration technique in more detail.

3 System Overview

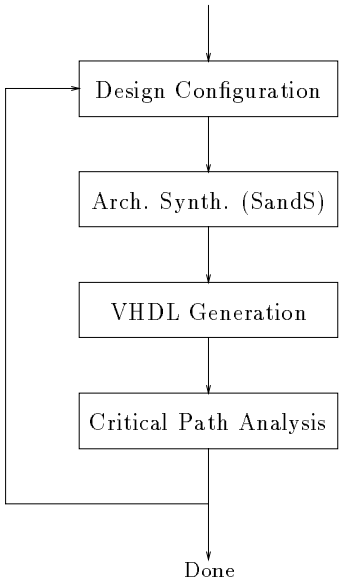

Figure 1: TinkerTool

Figure 1 shows a diagram of the TinkerTool system[Hsi92, LPH91]. The system is divided into four tasks: design configuration, architectural synthesis, VHDL generation and critical path analysis. In the design configuration task, the user interacts and specifies the parameters of a design to make the architectural decisions which control the synthesis tool, SandS.

SandS[PBLS91, PP92] is a behavioral synthesis system that schedules operations based on *mobilities*, which are the differences between *as late as possible* (ALAP) partitioning and *as soon as possible* (ASAP) partitioning. Nodes on the critical path, (i.e., nodes with node mobility of zero), are scheduled first [Pan87]. If two nodes have the same mobility, the node with the most successors is scheduled first. A symbolic (RTL) representation of this architecture is passed to the next task, the VHDL generation task.

In the VHDL generation task, a complete VHDL description of the architecture is generated. This task uses the RTL description of the synthesized architecture and the state controller obtained from SandS. Using the VHDL description of the architecture, the complete synthesized design can be verified using a VHDL compiler, and simulator[LMOI89].

Finally, the design is evaluated by the critical path analysis task. As discussed below, critical path delay is the primary iteration criteria. If the critical path delay is greater than the required cycle time, then the current architecture is not valid and further iteration is required. This process continues, under user control, until the iteration process converges on a valid architecture.

Once the user is satisfied with the design, behavioral tools like Psycho[Hou91] and layout synthesis tools, Cluster[OI88] and Artist[OI89] can then be used to synthesize the state controller and produce a mask layout of the entire design. After extraction, simulation tools can be used to verify the functionality and, more importantly the timing of the system. These tools are only used after the user has reduced the size of the search space of possible architectures to one which is likely to meet the performance requirements of the design.

4 The Delay Model

There are two difficult issues in incorporating low-level physical design information into the behavioral synthesis process: How to get good interconnection delay and control delay estimates; and how to incorporate this delay information into the synthesis process. In this section, we discuss both of these issues and present our solutions to these problems.

In our system we take a hybrid approach to estimating and using critical path delays. We define *critical path delay* as the worst case delay from the system clock edge, through the state controller, through a source register, a source bus, a possible multiplexor, a function unit, a destination bus, another possible multiplexor, and back to either a destination register or back to the state controller (for branches). This delay is calculated for each state of the state controller, with only the appropriate paths enabled for that state. For function units that perform operations which span several clock cycles, the total delay of the function unit is divided by the number of cycles it uses. The values used for the estimates of these delays come from various estimation techniques.

The control delay is the propagation delay from a transition on the clock signal to a transition

in the state controller outputs. The control delay is obtained by TinkerTool through simulation of the *magic*[SMHO85] layout generated by Artist/Cluster using IRSIM. An exhaustive set of state transition simulation vectors is generated automatically by TinkerTool to obtain the worst case control delay. Unlike the rest of the architecture which is built from components, we have found no reliable way to estimate the delays in the state controller without performing layout synthesis and extraction for this part of the design.

The delay of the interconnection paths must come from some notion of the wire length. TinkerTool uses simplified floorplanning to estimate wire length. Floorplanning of the components is not actually performed, because the Artist/Cluster tools are not based on module generators. Cluster actually re-partitions the design as part of its optimization process, it does not guarantee that components will remain "intact" during synthesis. However for our estimates a simple bounding box approach is used where components that are connected together are assumed to be clustered together. Worst case wire length is assumed to be the sum of the pre-calculated library dimensions of the components that are connected together.

Component delays have two parts, the unloaded delay of the component, and the output delay due to loading factors. Unloaded, delays for both the "intrinsic" components (e.g., registers and multiplexors) as well as the function units (e.g., adders and subtractors) are pre-calculated and stored in a database. These delays are augmented with a derating factor which is a function of the number of inputs.

Both wiring delay and output delay are calculated using first order linear interpolation with data obtained empirically using Spice on 2 μm CMOS technology. Wiring and component delay calculations are summarized in equations 1 through 3. We use these approximations, since we do not want to take the time to perform layout synthesis of the entire design during the iteration cycle.

$$wire\_length = \sum_{i=1}^{n} height_i + width_i \tag{1}$$

$$wiring\_delay = \mathcal{F}(wire\_length) \tag{2}$$

$$componet\_delay = \mathcal{G}(intrinsic\_delay, \quad number\_of\_outputs) \tag{3}$$

The second important issue for our delay model, is how do we use these estimated delays in the synthesis process. Unlike many synthesis engines, SandS does support control delay as an input to the scheduling process. However, it does not support models of interconnect delay, or intrinsic component delays. Therefore, TinkerTool uses the delay estimations derived during delay analysis to add "lumped" delays to the operation times of the corresponding function units in the component library passed to SandS.

As well as the state based critical delay calculation, a second calculation is performed to compute the worst case static delay from the output of each function unit, to the input of some other function unit. The static delay for each functional unit includes the total delay of the functional units, which may include delays imposed by multiplexor, buses, wires and registers that binds one functional unit to another functional unit. Unlike the critical delay estimation, where we are concerned with minimum cycle time, this time is *not* divided by the number of cycles used, because we are interested in the total execution time from input to output of each functional unit. Therefore, when SandS

schedules the operations on the next iteration using static delay of each functional unit, interconnect propagation delays are also allocated along with the functional unit delays.

Unfortunately, depending on the synthesized design, each functional unit may have its own unique interconnect, which implies that each functional unit may have its own unique interconnection delay, including those functional units that are multiple copies of one another. On the other hand, making the compromise of augmenting each unique operator with the worst case interconnection delay, selected out of that operator group, allows us to use SandS to converge quickly to a feasible design. Worst case delay is chosen because otherwise the design synthesized may not converge to a realistic design if the interconnection delay of the critical path is underestimated. This approach may at times not synthesize an optimum architecture for a design, but the design synthesized will always be a valid one. The design convergence issue is discussed below.

5 The Iteration Scheme

In this section we discuss the details of the design iteration scheme. The idea of iterating the synthesis process with the delay estimation process is key to the success of quickly exploring the design space, and converging on valid designs which meet the cycle time requirements of the architecture.

The use of an iteration scheme implies the need for an initial architecture of the design to serve as an initial base of the iteration scheme. To obtain an initial architecture of the design, we first consider only the basic functional unit delay characteristics and do not consider output delay and intrinsic component delay. The interconnection delay of this initial architecture is then a lower bound on the actual delay. For control delay, we allow the user to specify an initial guess, and obtain delay estimates for the next iteration based on the state controller synthesized.

The iteration scheme described here allows TinkerTool to desensitize the interdependence of delay estimates and target architecture by refining the architecture based on the previous iteration delay estimates. The following is the iteration scheme used by TinkerTool to iterate designs synthesized by SandS.

1. Select component resources and set cycle time constraints of the design.
2. Obtain initial architecture without considering interconnection delay.
3. Calculate critical path and control delays based on synthesized architecture.
4. Modify delay model of resource components based on current architecture.
5. Obtain refined architecture based on modified component delay.
6. Iterate steps 3-5 until a stable architecture is reached.

The first task in the iteration scheme is setting the constraints on the design by the user. This includes selecting the number and the types of components (operators) in the list of component resources as well as selecting the cycle time constraint for the design. The decisions made here will influence the architectures synthesized.

The second task in the iteration scheme is to obtain an initial architecture. This is achieved by TinkerTool, running SandS using the resources and the cycle time constraint set by the user and

using the default component delays. The initial architecture obtained here will serve as a starting point for the interconnection delay estimation and the control delay estimation.

The third task in the iteration scheme is to calculate both the state controller delay and the full critical path delay of the synthesized design. Both of these delays are needed to determine if the design is valid. Specifically, if the critical path delay is greater than the cycle time, then the current architecture is not valid, and thus further design iterations are required. Similarly, if the control delay of this iteration is different from the control delay of the previous iteration, then the control delay input to SandS was not accurate, and SandS might not have generated the best schedule. On the other hand, if the critical path delay is less than or equal to the cycle time and if the control delay of this iteration is same as in the previous iteration, then the cycle time constraint has been met, and thus further iteration is not required and the design iteration terminates.

If further design iteration is required, TinkerTool first returns control to the user. The user may decide to continue on with the design iteration or modify the constraints of the design. If the user decides to continue on with the design iteration, TinkerTool goes to step four of the iteration scheme. If the user decides to modify the constraints of the design, TinkerTool goes to step one of the iteration scheme.

The fourth task in the iteration scheme is to modify the delay model of the functional units in the component resources for the next design iteration. The interconnection delays of each functional unit are calculated by TinkerTool based on the interconnects of each functional unit in the current synthesized architecture. As mentioned in Section 4, the worst interconnection delay out of each unique functional unit group is incorporated in the delay model of that functional unit group.

The fifth task in the iteration scheme is to obtain a refined architecture by running SandS using the modified functional unit delay model and the control delay of the previous architecture. The architecture obtained here will serve as the starting point for the next design iteration in step three.

A synthesized architecture is valid only if the critical path delay and the control delay meet the cycle time constraints *and reach a stable state*. It should be made clear that SandS will often meet it's cycle time constraint, with what was given to it for the component delays, and yet the synthesized architecture will not be valid. That is because the critical path delay for new architecture could be worse than in the previous iteration and so the new architecture would have been scheduled incorrectly. Therefore the design must remain stable through the last two iterations.

When an architecture does not meet the specified constraints, then the design is iterated further through the iteration scheme presented above to obtain a refined architecture for the design. There are three types of modifications that can be made on an architecture to meet the specified cycle time constraints[Hsi92]. First, if the estimated critical path delay reported by TinkerTool is greater than the user specified cycle time, the architecture would be valid if the user chooses the estimated critical path delay as the cycle time constraint. Second, if the user chooses not to change the cycle time constraint, the next iteration of TinkerTool will invoke SandS with updated component delays. SandS will change the architecture in one of two ways. If the critical path delay is due to some chained operation [Pan88], then that chained operation will be broken up by SandS to reduce the critical path delay. Third, if the critical path delay is not due to a chained operation, then the user specified cycle length is too short for the prescribed execution cycle of the components. In

```
PROCESS(in_y,in_x)
    VARIABLE x,y,q:
        BIT_VECTOR(7 DOWNTO 0);
BEGIN
    y := in_y;
    x := in_x;
    q := in_q;
    WHILE (y >= x) LOOP
        y := y - x;
        inc(q);
    END LOOP;
    out_q <= q;
END PROCESS;
```

Figure 2: Division Algorithm

```
PROCESS(in_x,in_y,in_u,in_dx)
    VARIABLE dx,x,y,u,u1,u2:
        BIT_VECTOR(7 DOWNTO 0);
BEGIN
    dx := in_dx; x := in_x;
    y := in_y; u := in_u;
    WHILE (x < a) LOOP
        u1 := 5 * x * u * dx;
        u2 := 3 * y * dx;
        y  := y + (u * dx);
        u := u - u1 - u2;
        x  := x + dx;
    END LOOP;
    out_u <= u; out_y <= y;
END PROCESS;
```

Figure 3: Differential Equation Algorithm

this case, SandS "slices" these critical operations to execute over a larger number of cycles so that a short cycle length can be used[PP92].

The process of slicing operations to more and more cycles may continue iteration after iteration to meet the cycle time constraint. In each iteration, control delay increases, due to an increase in the number of control states. This implies that the time allocated for operation execution, becomes shorter and shorter. As a result, an upper bound can be reached when the control delay alone is longer than the cycle length. In that case, nothing is gained by further modifications on the execution cycle of the operation. This implies that the given resource and cycle time constraint is simply not realizable.

6 Experimental Results

In this section we present two sets of experiments. The first set illustrates the iteration technique in detail, using a simple integer division algorithm. The second set shows how we can use TinkerTool to perform design tradeoffs, using a standard differential equation example.

6.1 Iterations on a Design

In this section we will use a simple integer division algorithm shown in Figure 2 to illustrate the iteration steps on a design. The first step in the iteration scheme is to select the component resources and the cycle time constraints for this integer division design. For this example, the user selects from TinkerTool's basic component library an incrementer (inc), a subtractor ($-$) and a greater-than-or-equal-to comparator ($\geq$) functional units as the component resources. The three components selected have component delays of 19ns, 30ns and 22ns, respectively. For this example we set the cycle time constraint to 75ns and set the lower bound initial control delay to 20ns.

The second step in the iteration scheme is to obtain an initial architecture to use as a starting point in the design iteration. Using the component resources and cycle time constraints set in the

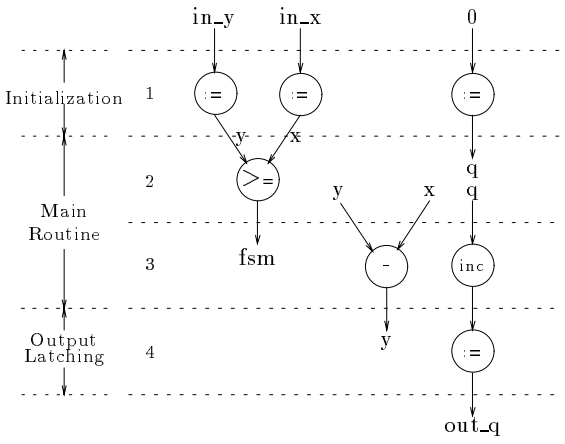

Figure 4: Four State Schedule

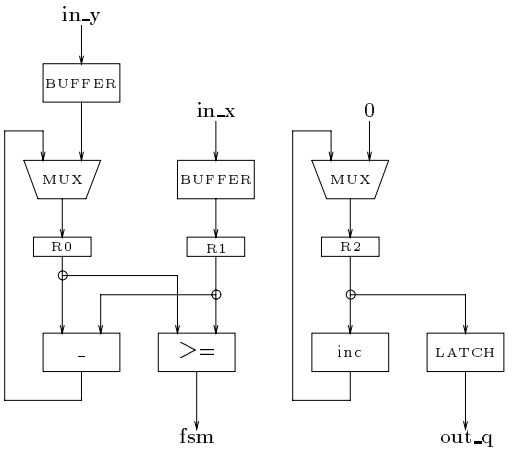

Figure 5: Block Diagram

previous step, TinkerTool obtains an initial architecture with a four state schedule from SandS. In this four state schedule, shown in Figure 4, each operation is executed over one cycle. The block diagram illustrating the interconnects for this initial architecture is shown in Figure 5.

Using the initial architecture obtained from step two, TinkerTool calculates the critical path delay and the control delay of the this architecture. The control delay for this four state controller is estimated at 25ns. The dynamic critical path analysis performed by TinkerTool on this architecture shows that the critical path lies in the subtraction operation, from the subtractor to the multiplexor and to the register R0. The sum of these component delays and wire delays is the critical path delay of this initial architecture, which TinkerTool estimates to be 78ns. Since the critical path delay does not meet the cycle time constraint, we continue on to step four of the iteration scheme.

The fourth step in the iteration scheme is to modify the delay model of each functional unit to account for the interconnection delay effects on the design. To determine the interconnection delay for each component, TinkerTool uses static analysis to determine the worst case path for each component. Returning to Figure 5, the worst case path for the increment operation starts from the incrementer to the multiplexor and to the register R2. The worst case path for the subtractor is the critical path mentioned above. Finally the worst case path for the comparator operation starts from the multiplexor to the register R0 and to the comparator. The interconnection delays for each component, is the sum of all intrinsic component delays and wire delays along the worst case path of each component, which TinkerTool estimates to be 44ns, 48ns and 47ns, respectively. Incorporating these interconnection delays into the functional unit delay models, the modified functional unit delay is calculated to be 63ns, 78ns, and 69ns, respectively.

Using the modified functional unit delay models and the modified control delay estimate, TinkerTool obtains a refined architecture from SandS. In the second iteration, TinkerTool obtains a six state schedule with critical components, (inc), ($-$) and ($\geq$) executing over 2 cycles. For this simple example, the interconnects for this architecture are unchanged, only the scheduling of operations is modified. The critical path delay of this architecture, estimated by TinkerTool, is 63ns, which meets the cycle time constraint of 75ns. The control delay of this architecture, estimate by TinkerTool, is 31ns. Since the control delay estimate in this iteration differs from the previous iteration,

ITER #	Component Delay				Critical Delay	New FSM	Control States
	inc	$-$	$\geq$	fsm			
1	19	30	22	20	78	25	4
2	63	78	69	25	63	31	6
3	63	78	69	31	63	31	6

Table 1: Integer Division (Cycle Time = 75ns)

EXAMPLE Run	Component Resources					Cycle Time
	$*$	$+$	$-$	$+-$	$<$	
A	2	1	1	0	1	200
B	2	1	1	0	1	150
C	2	0	0	1	1	150
D	1	0	0	1	1	150

Table 2: Differential Equation Resources

we continue to iterate the design.

As stated above, the interconnect of this architecture is the same as the one in the previous iteration, therefore, the interconnection delays for the three functional units remain the same as in the last iteration. Consequently, the modified functional unit delays remain the same as well. Iteration continues until the cycle time constraint is met and a stable state in the control delay is reached. In this example stable state is reached at the third iteration with critical path delay of 63ns and control delay of 31ns.

In this simple example we have shown how estimates are calculated and used as a criteria for iteration. We have also shown how quickly we reached a stable state in only a few iterations. Iteration results for this simple integer division are shown in Table 1.

6.2 Design Tradeoffs

In this section we will use a differential equation presented in HAL [Gir85] to illustrate the tradeoffs that can be made to search for the "optimal" architecture in terms of execution time and layout area. The behavioral VHDL description of the differential equation example is listed in Figure 3.

Table 2 lists component resources and cycle times of four different example runs of this circuit design. Table 3, Table 4, Table 5 and Table 6 show the results of each of the iteration runs. The design decisions made on each example run are evaluated based on the execution time and the interconnects of architectures synthesized are listed in Table 7. Ideally, we want to use layout area as a cost function, but logic synthesis is a time consuming process. Therefore, it is impractical to use layout area as a cost function. However, we can assume layout area is proportional to the sum of the areas of the components used in the design, and thus, can be used to infer the layout cost. Example B has the fastest design using 2 multipliers, one adder, one subtractor and one comparator with a cycle time of 150ns and 11 control states. Example D has the slowest design using 1 multiplier, one *add_sub* multifunction unit and one comparator with a cycle time of 150ns and 14 control states. We had expected example D with only one multiplier to have lowest area cost, but as it turns out the increase in control delay and in the interconnection delay offset the advantage of having one less multiplier. Clearly, out of the four example runs, example B has the best area and execution time tradeoff.

7 Conclusions

TinkerTool addresses the issue of incorporating interconnection delay into the synthesis process through the augmentation of the delay in the delay model of module units. TinkerTool addresses

ITER #	Component Delay					Critical Delay	New FSM	Control States
	$*$	$+$	$-$	$<$	fsm			
1	72	38	39	18	50	223	54	6
2	156	112	94	74	54	187	62	9
3	148	104	109	73	62	187	62	9

Table 3: Differential Equation Example A (Cycle Time = 200)

ITER #	Component Delay					Critical Delay	New FSM	Control States
	$*$	$+$	$-$	$<$	fsm			
1	72	38	39	18	50	133	52	7
2	153	104	117	74	52	100	62	11
3	150	104	117	74	62	100	62	11

Table 4: Differential Equation Example B (Cycle Time = 150)

ITER #	Component Delay				Critical Delay	New FSM	Control States
	$*$	$+-$	$<$	fsm			
1	72	48	18	50	181	63	10
2	182	139	75	63	157	85	17
3	182	139	80	85	146	95	18
4	182	139	82	95	146	95	18

Table 5: Differential Equation Example C (Cycle Time = 150)

ITER #	Component Delay				Critical Delay	New FSM	Control States
	$*$	$+-$	$<$	fsm			
1	72	48	18	50	166	54	8
2	164	131	86	52	142	73	13
3	157	129	87	73	142	74	14
4	157	129	88	74	142	74	14

Table 6: Differential Equation Example D (Cycle Time = 150)

EXAMPLE Run	Intrinsics		Execution Time (ns)	Transistor Count	Component Σ Area
	reg	mux			
A	10	12	1400	11846	8761610
B	10	10	1350	11858	8915295
C	10	10	1800	12340	9789233
D	11	18	2400	12380	10757901

Table 7: Evaluation of Synthesized Differential Equation Circuit

the issue of interdependency between the synthesized architecture and the delay estimates through the use of a design iteration scheme. The idea of iterating the synthesis process with the delay estimation process is key to the success of quickly exploring the design space, and converging on valid designs which meet the cycle time requirements of the architecture.

8 References

[BG86] Forrest Brewer and Daniel Gajski. An expert system paradigm for design. In *Proc. of the 23th Design Automation Conference*, July 1986.

[BG90] Forrest Brewer and Daniel Gajski. Chippe: A constraint driven behavioral sythesis. In *IEEE Transaction on Computer Aided Design*, July 1990.

[Gir85] P. Paulin J. Knight E. Girczyc. Hal: A multi-paradigm approach to automatic data path synthesis. In *Proc. of the 22th Design Automation Conference*, p 587–594, 1985.

[Hou91] Pao-Po Hou. *Mid-Level Hardware Synthesis*. PhD thesis, Dept. of Computer Science, Penn State Univ., University Park, PA, May 1991.

[Hsi92] Yee-Wing Hsieh. Architectural synthesis via vhdl. Master's thesis, Dept. of Electrical Eng., Univ. of Pittsburgh, Pittsburgh, PA, 15261, December 1992.

[LMOI89] S.P. Levitan, A.R. Martello, R.M. Owens, and M.J. Irwin. *Proc. 9th Inter. Symposium on Comp. Hard. Description Lang.*, chapter Using VHDL As A Language For Synthesis Of CMOS VLSI Circuits, pages 331–346. Elsevier Science Publishers, B.V., June 1989.

[LPH91] Steven P. Levitan, Barry M. Pangrle, and Yee-Wing Hsieh. Architectural synthesis via vhdl. In *Third Physical Design Workshop*. ACM/SIGDA, May 20-23 1991.

[McF86] Michael C. McFarland. Using bottom-up design techniques in the synthesis of digital hardware from abstract behavioral description. In *Proc. of the 23th Design Automation Conference*, pages 602–608, 1986.

[MPC88] Michael McFarland, Alice Parker, and Raul Camposano. Tutorial on high-level synthesis. In *Proc. of the 25th Design Automation Conference*, pages 330–336, 1988.

[OI88] R.M. Owens and M.J. Irwin. Exploiting gate clustering in VLSI layout. Technical Report CS-88-09, Computer Science, Penn State Univ., Univ. Park, PA, 1988.

[OI89] R. M. Owens and M. J. Irwin. A comparison of four two-dimensional gate matrix layout tools. In *Proc. of the 26th Design Automation Conference*, pages 698–701, 1989.

[Pan87] B. Pangrle. *A Behavioral Compiler for Intelligent Silicon Compilation*. PhD thesis, Dept. of Computer Science, University of Illinois, June 1987.

[Pan88] B. Pangrle. "Splicer: A Heuristic Approach to Connectivity Binding". In *Proc. of the 25th Design Automation Conference*, pp. 536–541, 1988.

[PBLS91] Barry M. Pangrle, et. al. Relevant issues in high-level connectivity synthesis. In *Proc. of the 28th Design Automation Conference*, pages 607–609, 1991.

[PP92] Usha Prabhu and Barry M. Pangrle. Superpiplined control and data path synthesis. In *Proc. of the 29th Design Automation Conference*, pages 638–642, 1992.

[SMHO85] Walter S. Scott, Robert N. Mayo, Gordon Hamachi, and John K. Ousterhout. 1986 VLSI tools: Still more works by the original artists. Technical report, Computer Sciences Division (EECS) University of California, Berkeley, CA 94720, December, 1985.

[WE85] Neil H. E. Weste and Kamran Eshraghian. *Principles of CMOS VLSI Design*. Addison-Wesly Publishing Company, Reading, MA, 1985.

[WP91] Jen-Pin Weng and Alice C. Parker. 3d scheduling: High-level synthesis with floorplanning. In *Proc. of the 28th Design Automation Conference*, pages 668–673, 1991.