Concurrent Analysis Techniques for Data Path Timing Optimization*

Chuck Monahan Forrest Brewer

Department of Electrical and Computer Engineering
University of California, Santa Barbara, U.S.A.

Abstract

Conventional High-level Synthesis techniques create an interconnection structure before physical design. Following physical design, connection delays and special requirements may cause the structure to fail timing or performance constraints. Alterations to this structure are often limited since it was created either after or during the binding and scheduling tasks. In this paper we present a set of techniques which analyze the timing trade-offs associated with the position-specific interconnection network given the freedom of high-level binding and rescheduling changes.

1. Introduction

Systematic evaluation of interconnection trade-offs in designs containing pre-designed cores or pre-defined structural implementation present several difficult challenges. In particular, changes to the interconnections structure appear as timing changes yet may require operation rescheduling and/or operand rebinding. This is because detailed timing information is only available late in the design cycle after scheduling, binding, interconnection synthesis and floorplanning have already been performed. One could modify the interconnection to only allow functionally equivalent alternatives, but at great cost to the design freedom. Our aim is to provide techniques for timing optimization under the constraints of a predefined data path structure and feasible interconnection alternatives.

Positional layout information is utilized to estimate the cycle time of competing designs. Fig. 1 depicts a simple RT-level data path, data flow, and floorplan which completes the data flow in seven cycles. Two additional wire segments are indicated on the data path; either of which permit six cycle execution. However, incorporating these changes will alter the affected nets, potentially

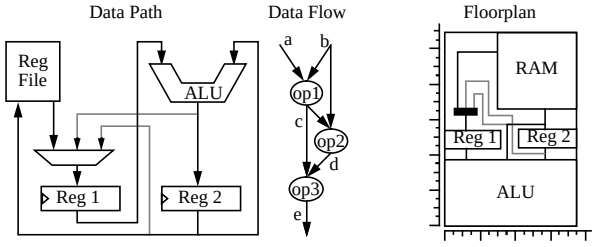


Figure 1. Example problem of data path alternatives.

* This work has been generously supported by UC MICRO 95-021 and Mentor Graphics Corp.

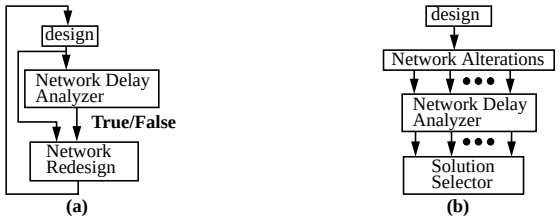


Figure 2. Feedback analysis vs. concurrent evaluation.

degrading the system cycle time. A designer with a highly constrained clock cycle may not wish to alter the design, while another who desires higher performance wants to know which of the two scenarios is faster.

An analysis of the timing due to alternative interconnection networks must account for the introduction and placement of buffers and switching elements. Designers utilize these elements to substantially alter the timing delay along the critical path(s) without changing the functional behavior. Nonetheless, reducing delays along one path usually comes at the price of increasing timing delays along alternative routes. If the penalty is large enough, a new critical path is created whose delay exceeds that of the former critical path, degrading the system performance.

In this work, we introduce a systematic method for the simultaneous evaluation of multiple designs. In contrast to feedback approaches (Fig. 2a) which may become confused by local minima, our system processes a number of designs alternatives concurrently (Fig. 2b) with equal consideration. Efficiency is maintained by pruning designs which fail to meet the designer's specifications and by sharing common information between the set of active designs.

2. Previous Work

Considerable work has been performed in the area of interconnection synthesis and design. Work by [11], [8], [5], [3], and [2] introduced many key components of interconnection synthesis. Due to a lack of positional and/or load models, those techniques measured the quality of their results by the number of multiplexers rather than timing analysis. Timing analysis for interconnection network trade-offs was discussed in [9] and [6]. The first work modeled path-dependent timing delays but restricted the set of component configurations. The second work considered rebinding of function units and memory elements but restricted the designs to one-dimensional layout in order to simplify propagation delay estimation.

Recent developments in retargetable compilers have introduced two promising data path analysis techniques. Instruction set extraction from a mixed structural/behavioral description was introduced in [4]. [10] utilized a combination of code selection and register binding but restricted their use to a restricted set of datapath components.

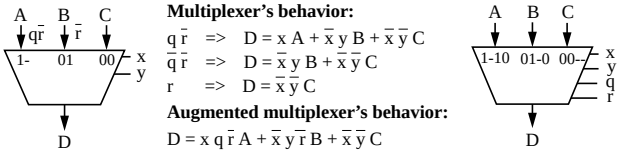


Figure 3. Modeling restricted mobility as switch behavior.

3. Problem Specification

In this work, the automata based data-path model introduced in [7] is utilized to implicitly execute a series of data flows on a detailed RTL data path. Given the additions of a floorplan, a set of alternative interconnection networks, and a set of timing ranges, the automata will produce a timed output sequence as well as characterize the timing behavior of the "care paths" for each element of the solution set. New variables were added to the automata state vector to accommodate the concurrent analysis and timing problems.

A number of additions to the data path specification were required to incorporate timing analysis. To facilitate interconnection modification, propagation delays along the wires are computed dynamically based on the set of connection points. Therefore, the positional layout of each connection point is specified. While functionally equivalent to multiplexers, the rich positional information of bus elements required their inclusion into our list of base components as well. Finally, component timing delays must be specified.

4. Conditional Connections

The problem specification describes a set of interconnection network alternatives. *Topology variables* are a set of Boolean variables used to encode each network's architecture. These variables allow an arbitrary representation of topological interconnection changes with minimal encoding.

A *conditional connection* is the connection between a wire and a component's input which exists for only a subset of the proposed interconnection networks. The behavior of these conditional connections may be modeled using techniques previously developed for switching constraints if the following restrictions are obeyed:

- The *condition* of each conditional connection is the sum of the encodings associated with the subset of networks in which the connection exists.
- Multiple conditional connections may be defined for the same component's input only if their conditions are disjoint.
- Conditional connections occur only at switch inputs. Zero delay switches are inserted to accommodate conditional connections to non-switch inputs.
- A switch with conditional connection at an input, will now only connect if the switching control variables are set to select the desired input and if the topology variables are set to meet the condition of the connection. Fig. 3 depicts such a transformation for an example multiplexer.
- The automata treats the set of switching and topology variables differently. Whereas the switching variables are allowed to change from cycle to cycle, the topology variables are incorporated into the state vector where they are used to restrict future topology settings and ensure a consistent interconnection network for all cycles.

Conditional connections provide the designer with a number of powerful features. It's always possible to describe any potential network using a minimal number of bits to distinguish potential connection. Further, the conditions for the various connections need not be disjoint. This allows the designer to propose a set of trade-offs, where the resulting solution may be selected from any combination of these trade-offs. Finally, this system shares a substantial portion of the analysis of the different networks when the differences between the designs are small.

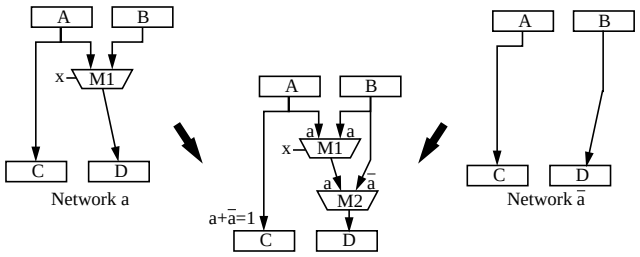


Figure 4. Labeling and merging network topologies.

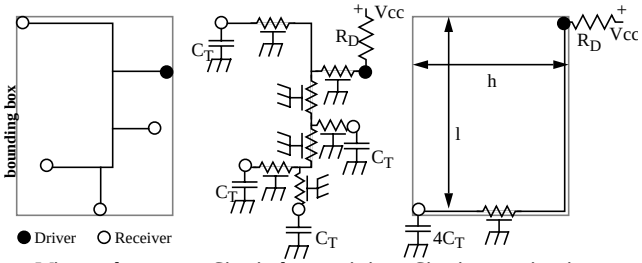
Fig. 4 depicts the process for combining two example interconnection networks. Initially each network is assigned a disjoint encoding: a and ā. The conditions for each potential connection is formulated. Finally the zero-delay switching component, M2, is introduced into the combined network to distinguish the connection options for the input of component D.

4.1. Evaluating Timing Modifications

The insertion and removal of connections affect the timing characteristics of the wires. Before discussing their effects, we first present our delay model for wires with a set of fixed terminal connection points. In the absence of detailed routing and timing information, we use a conservative approximation of the propagation delays based on a wire's connection points. While future work will allow timing delays to vary between different path down the same wire, this is not currently modeled. Finally, note that these approximations apply only to delays along multi-terminal metal nets. The delays incurred through the path-dependent switching elements are modeled according to user-supplied parameters.

The transmission delay is dependent upon the distributed RC (resistance-capacitance) of the wire, the resistance of the driver, and the cumulative capacitance of the taps. Distributed RC is a function of the wire length which estimate as half of the perimeter of the Manhattan bounding box containing the wire's set of drivers and receivers. We assume that the driver resistance and capacitance values are constant as well as the receiver capacitance. The cumulative tap capacitance is a function of the number and type of connecting points of the wire. The 50% rise time, appropriate for static CMOS circuits, is computed from these three values using the approximation in [1]. Fig. 5 displays an example wire configuration, its actual physical properties, our estimated model, and a timing equation for the communication shown.

Conditional connections alter this analysis by removing and inserting elements from a wire's set of connection points altering the bounding box and total capacitance values. Proper modeling requires instantiating each set of viable connection points to deter-



$$R_w = (l+h)\,R_{unit},\ C_w = (l+h)\,C_{unit},\ C_L = (\sum_{taps} C_T)$$

$$t_{50\%} = 0.4R_wC_w + 0.7\,(R_DC_w + R_DC_L + R_wC_L)$$

Figure 5. Wire Delay Model.

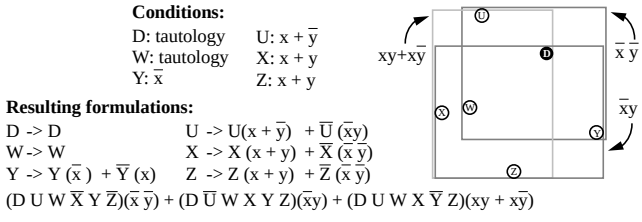


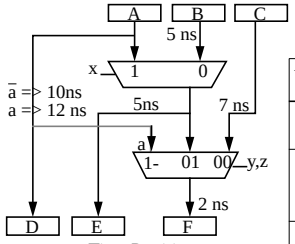
Figure 6. Instantiating viable connection sets.

mine the range of propagation delays for each wire. Typically, the user will specify a small set of connection combinations per wire as in the example point set in Fig. 6. However, the complexity of identifying these viable sets can grow exponentially in relation to the number of potential connection points. Therefore, we utilize a compressed Boolean representation to compile a list of viable connection sets. Fig. 6 shows how this approach is performed in a two step process for our example. First, we express a connection's existence as a Boolean function by XNORing a variable representing the connection with the connection's existence condition. ANDing the set of resulting terms produces a Boolean relation which describes the connection sets and the pertinent control values which formulate the set. The wire delays may be computed directly from these connection sets and then paired with their Boolean condition for the subsequent timing analysis.

5. Timing Variables

We introduce a set of *timing variables* to encode the timing delays through the data path. Each variable is associated with a unique range of timing delays. Each path delay through the data path is identified by a component's input port and control vector. The control vector operates over the set of switching variables (which decide which path is taken to the input) as well as the set of topology variables (which effect the delay of the various wire segments which comprise the path). After instantiating every path delay through the data path, the timing delay will be symbolically represented with the associated timing variable in a one-hot encoding. Delays which violate user-specified timing constraints will have their associated path removed from the automata construction. These constraints may appear as both minimal and maximal allowed delays through the circuit.

Fig. 7 depicts these timing variables in use for an example network. The network is comprised of a series of zero delay multiplexers and contains a single conditional connection linking component A to the second multiplexer. The set of timing variables are listed in the "Time Partition" tables. Given a maximum delay of 15ns and no minimal timing delay, the "Network Timing" table lists each timing delay and their delay encodings.



Time Partitions

Timing variable	*Time range*
t_1	10-11ns
t_2	12-13ns
t_3	14-15ns

Network Timing

Primary output	*Primary input*	*Control settings*	*Time delay*	*Delay encoding*
D	A	ā	10ns	t_1
		a	12ns	t_2
E	A	~~x a~~	17ns	Ø
		x ā	15ns	t_3
	B	x̄	10ns	t_1
F	A	y a	14ns	t_3
		~~x y z ā~~	17ns	Ø
	B	x̄ ȳ z	12ns	t_2
	C	y z	9ns	

Figure 7. Example partitioned delay network.

6. Automata Issues

The timing variables are incorporated into the automata state vector to derive timing attributes for elements of the solution set. The cycle time for a synchronous design is the maximal propagation delay between clocked components. Therefore, each transition between states will include a vector of timing requirements obtained from paths utilized for operand transfers. Summing these timing vectors in a state vector generates a history of timing requirements for each solution.

Modeling timing behavior on the automata structure outlined in [7] requires modeling of *null operands* during the state exploration. Null operands carry "don't care" information in the automata and are a convenient means to encode the contents of "empty" latches which frequently appear in initial conditions and in pipelined designs. Previously, the inclusion of null operands was required only when no other operand was available for a given memory element. In the presence of timing and power analysis, null operands must be considered regardless of the status of other operands. This is because placing operands into latches which could otherwise remain empty will affect the timing and power attributes of the system design. It is preferable to use null operands which have no time requirements to operands which are stored in a memory device but serve no useful function except to be subsequently overwritten. Unfortunately, the usefulness of an operand can not be evaluated beforehand. Therefore, all potential operand bindings (including regular and null operands) must be considered equally. This equal consideration may cause explosive growth in the set of potential solutions. To curb this growth, the automata is used in two phases: the first ignores timing issues and determines the solution set from the set of potential solutions and the second reevaluates this solution set using a more extensive analysis to systematically evaluate the timing requirements of the "care path" of the system.

The elimination of "false paths" from our timing analysis is one of the most elegant features of the technique. Since the timing characteristics of each solution is derived from only the required set of operand transfers, timing delays from unutilized portions of the data path do not mislead the timing analysis. Further, the set of all the care sequences can be generated for a given solution. This capacity is of great importance for the subsequent steps of verification analysis.

7. Results

An application tool was constructed to demonstrate the feasibility of the ideas proposed in this paper. The input for this tool consists of three major sections: 1) A RT-level description of the data path, including positional information and conditional interconnections; 2) A set of scheduled data flows which must be mapped onto the proposed data path; and 3) A partitioned timing range in the format described in Section 5.

This tool evaluates which of the proposed changes accommodate the full set of data flows. From the set of realizable alterations, the designs that result in the highest performance are chosen. Upon the user's request, the system will generate a set of Boolean relations representing an exhaustive list of binding options for the selected designs. Alternatively, a textual description of a single representative binding and schedule is listed.

This application was implemented on a Sun SPARC 10 in C++ using an OBDD (ordered binary decision diagram) package to represent the boolean functions. Examples were run on two manually constructed data paths. The first data path is based on Texas Instrument's TMS32010 DSP which was altered to accommodate a pipelined multiplier. For this specification, three different interconnection alternatives were specified: the use of multiplexer or

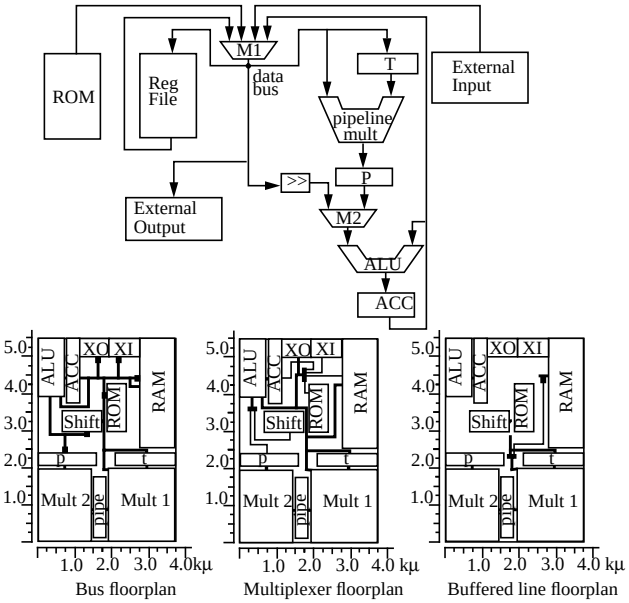


Figure 8. TMS32010 based data-path model and floorplans

equivalent bus model for switching elements M1 and M2, and a direct buffered line between the register file and multiplier was permitted. The design and it's manual floorplan, including interconnection alternatives, are outlined in Fig. 8. Additionally, we introduce a second data path accompanied by its floorplan in Fig. 9. This data path features a symmetric interconnection structure and dual register banks designed to test our limitations in operand rebinding.

TABLE 1. Results

Bench-mark	*TMS32010*				*dual register design*			
	# cycles	*execution time (s) timing*	*execution time (s) rebinding*	*cycle time (ns)*	*# cycles*	*execution time (s) timing*	*execution time (s) rebinding*	*cycle time (ns)*
diff eq	20	27.34	34.38	36.68	15	3.33	7.47	28.98
dhrc	27	28.44	32.89	36.68	21	4.95	6.73	28.98
ewf	53	40.74	278.24	36.63	45	22.65	177.46	28.98

Table 1 lists the results from analyzing three benchmark data flows. The *differential equation, differential heat release computation*, and *elliptic wave filter* scheduling benchmarks were run on each of these data paths under two different modes. The first mode utilized a predefined schedule and operand mapping to purely test the freedom in interconnection and determine fastest cycle times (timing). The second ignored the specified operand mapping, considered all alternative operand mappings, and presented the results for the fastest cycle time (rebinding). The length of the schedule for each benchmark/data path combination are listed in the columns "#cycles". The column "cycle time" reflects the computed cycle time given the floorplan and parameters listed in Table 2.

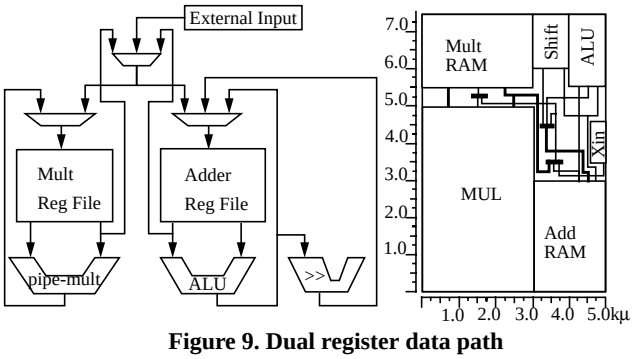


Figure 9. Dual register data path

TABLE 2. Physical Parameters

Parameter	*Value*
Rd	3.175 kΩ
Ct	20 ff
Rw	0.166 Ω/micron
Cw	0.176 ff/micron

Analysis of the TMS32010 interconnection alternatives reveals a consistency for the examples. All of the examples preferred the bus design to replace mux M1 and chose to retain mux M2. Inclusion of the buffered line is dependent upon the schedule utilized. For the examples listed in the table, the buffered line only increases the propagation delays to the ALU. But the inclusion of the line also permits extra operand mobility shortening schedules between 4-14% at the cost of longer cycle times. Additionally, one should note that designs which use a single cycle multiplier uniformly adopted the buffered line, since the multiplier replaces the ALU as the critical path.

An analysis of execution times reveals general trends for the system. The results under the pure "timing" analysis reflect the overhead for handling the longer data flows and the more complex data paths. The addition of the alternative interconnection component for the TMS32010 specification causes the increase in these execution times. When rebinding is considered, the interaction of the data path and specified schedule define the solution space. This interaction causes the surprisingly longer execution of the "diff eq" over the shorter dhrc benchmark.

8. Conclusion

In this work we have introduced techniques for concurrently analyzing differing rebinding options over multiple designs. Timing analysis is utilized to limit the search space and rate the final designs. Key benefits include optimal change control and elimination of fasle-path timing sequences. Future work will expand the model to incorporate power estimation, perform limited rescheduling, and utilize the data path to identify critical paths and suggest interconnection wire alteratives.

9. References

[1] H. Bakoglu, Circuits, *Interconnections, and Packaging for VLSI*, Addison-Wesley Publishing Company, 1990.

[2] C. Ewering, "Automated High Level Synthesis of Partitioned Busses," *Proc. IEEE Int. Conf. Computer-Aided Design*, pp 304-7, 1990

[3] A. Jerraya and B. Courtois "The SYCO Silicon Comlier and its Environment," in *Design Systems for VLSI Circuits*, Martinus Nijhoff: Dordrecht. pp 499-526

[4] R. Leupers, P. Marwedel, "A BDD-based Frontend for Retargetable Compilers," *Proc. of the European Design Automation Conference*, pp 239-243, 1995.

[5] T. Ly, W. Elwood, and E. Girczyc, "A Generalized Interconnect Model for Data Path Synthesis," *27th Design Automation Conference Proc.*, pp. 168-73, 1990.

[6] C. Monahan, F. Brewer, "Communication Driven Interconnect Synthesis," *6th International Workshop on HLS Proc.*, pp 65-73, 1992.

[7] C. Monahan, F. Brewer, "Symbolic Modeling and Evaluation of Data Paths," *32nd Design Automation Conference Proc*., pp 389-94, 1995.

[8] B. Pangrle, "Splicer: A Heuristic Approach to Connectivity Binding," *25th Design Automation Conference Proc.*, pp. 536-41, 1988

[9] J. Rabaey, "CATHEDRAL-II: Computer-aided synthesis of digital processing systems," presented at the *IEEE Custom Integrated Circuits Conf.*, May 1987.

[10] A. Timmer, *et. al*, "Conflict Modelling and Instruction Scheduling in Code Generation for In-House DSP Cores," *32nd Design Automation Conference Proc.*, pp.593-398, 1995.

[11] C. Tseng and D. Siewiorek, "Automated Synthesis of Data Path in Digital Systems," *IEEE Trans. on Computer-Aided Design*, Vol. 5, No. 3, pp 379-95, July 1986.