Average-Case Optimized Technology Mapping of One-Hot Domino Circuits*

Wei-chun Chou† Peter A. Beerel† Ran Ginosar‡** Rakefet Kol‡
Chris J. Myers§ Shai Rotem¶ Kenneth Stevens¶ Kenneth Y. Yun‖

†EE-Systems, University of Southern California, Los Angeles, CA, USA.
‡VLSI System Research Center, EE and CS Depts., The Technion, Haifa, Israel.
§EE Department, University of Utah, Salt Lake City, UT, USA.
¶Intel Corp., Hillsboro, OR, USA.
‖ECE Dept., University of California, San Diego, CA, USA.
** on sabbatical leave at Intel Corp., Hillsboro, OR.

Abstract

This paper presents a technology mapping technique for optimizing the average-case delay of asynchronous combinational circuits implemented using domino logic and one-hot encoded outputs. The technique minimizes the critical path for common input patterns at the possible expense of making less common critical paths longer. To demonstrate the application of this technique, we present a case study of a combinational length decoding block, an integral component of an Asynchronous Instruction Length Decoder (AILD) which can be used in Pentium® processors. The experimental results demonstrate that the average-case delay of our mapped circuits can be dramatically lower than the worst-case delay of the circuits obtained using conventional worst-case mapping techniques.

1 Introduction

Asynchronous circuits are attractive alternatives to synchronous circuits because they have the potential advantages of higher average-case performance [12, 13, 6], lower power consumption, and freedom from clock-skew problems. Recent emerging asynchronous designs have shown impressive results for digital signal processing [9, 23, 19] and microprocessors [7, 10, 4], but the lack of CAD support is still limiting their advances in some areas. This paper focuses on a CAD tool for a specific type of design, combinational circuits that convert data signals into control signals. These circuits typically perform instruction decoding of some type and, due to their complexity, are often the bottleneck in both synchronous and asynchronous microprocessors. We focus on a new design style and an accompanying CAD tool which can remove this bottleneck, offering in some cases dramatic improvements in average-case delay.

Traditionally, combinational circuits that convert data into control signals are implemented using single-rail bundled-data techniques. This method unfortunately implies that the delay of the circuit is determined by the most complex data needed to be decoded (rather than the most common data). Dual-rail techniques, in which each signal is encoded with two bits, can also be used to design these circuits and facilitate the optimization for average-case performance. Traditional dual-rail designs, however, are typically larger, consume more power, and are slower (due to the complex completion sensing structures required) than single-rail designs.

In this paper, we consider a different design style for these decoders which applies a combination of domino logic, dual-rail signaling, and one-hot encoded outputs. Chris Myers initially conceived of this design [11] and Benes et al. independently developed a similar technique that they used in a decompression circuit for embedded processors [4]. Domino logic is used for its well-known speed advantage over static logic and because it guarantees that the outputs are hazard-free. However, a single stage of domino logic can only realize functions that are monotonic. Thus, to implement all functions, some dual-rail inputs and some dual-rail internal signals are sometimes needed. Moreover, the

*This research is funded in part by a gift from Intel Corporation and a NSF CAREER Grant MIP-9502386.

design style uses one-hot encoded outputs to reduce the overhead of completion detection of the evaluation phase of the domino logic. The completion detection of the precharge phase is simply removed with a timing assumption on the precharge signal. The key advantage of this design style is that the domino logic can be optimized to prioritize the computation of instructions depending upon the instruction frequency, potentially leading to dramatic improvements in average-case delays. The circuits, however, can be large and complex, and thus could benefit substantially from supporting CAD tools.

In this paper, we focus on the technology mapping problem for this class of circuits. The circuits are specified with a set of *incompletely-specified input patterns*, each associated with a probability that reflects the input pattern's relative frequency of occurrence. In practice, these probabilities can be derived from architectural simulation of the design on typical data. In addition, we assume that the degree of sharing between cones of logic has been determined by technology-independent optimization. More specifically, we assume that the unmapped circuit structure is given in the form of a NAND-decomposed graph.

The key obstacle to technology mapping of these circuits is that the delay of a circuit for an incompletely-specified pattern cannot be precisely determined because the critical path is unknown when a primary input is specified to be an "X". Fortunately, one-hot domino circuits have a special property that allows us to easily *bound* the delay for an incompletely-specified pattern. In particular, for each incompletely-specified pattern c, we identify two *representative*, completely-specified patterns, c_l and c_u, that yield lower and upper bounds of the delay for pattern c.

Based on this theory, we propose to reduce the technology mapping problem of one-hot domino circuits to the completely-specified input-pattern dependent approach proposed in [2, 3], which is modified slightly to handle domino logic. Specifically, we replace each incompletely-specified pattern by one of its two representative patterns. Then, we call the mapping routines described in [2, 3] to minimize the average-case delay. Finally, we use the representative patterns to derive bounds of the average-case delay of the mapped circuit.

We demonstrate our approach with a case study of an asynchronous instruction length decoder (AILD) for Pentium® processors. In particular, we describe *two* combinational blocks for length decoding which are key components of a fast asynchronous length decoder. Our experiments support three important results:

- The range of average-case circuit delays that we derived by our *representative* patterns is narrow (within 11%), thereby illustrating the precision of our bounds.
- The average-case delays of both our circuits are significantly smaller than the average-case delays of the comparable circuits derived using synchronous techniques, thereby illustrating the potential power of our new technology mapper.
- The average-case delays of both our circuits are dramatically smaller than the worst-case delay of the comparable synchronous circuits, demonstrating the potential performance benefit of asynchronous circuits.

The remainder of the paper is organized as follows. Section 2 provides the necessary background on technology mapping. Section 3 describes the features of one-hot domino logic. Section 4 describes the extensions to existing technology mapping techniques to handle incompletely-specified patterns and domino logic. Section 5 presents the case study in which this technique is applied to the design of an asynchronous instruction length decoder. Finally, Section 6 gives our conclusions.

2 Technology mapping background

For synchronous circuits, technology mapping is often reduced to directed acyclic graph (DAG) covering which can be efficiently approximated by a sequence of optimal tree coverings [14]. The optimized equations (obtained from the technology-independent optimization) are decomposed into a DAG where each node is a base function. Particularly, the DAG is called a *NAND-decomposed* graph if the set of base functions consists of only a NAND2 and an INVERTER [5]. The technology mapping problem is to find a minimum cost covering of the decomposed graph using available library gates. For area optimization, the cost of a cover is defined as the sum of the gate areas. For delay optimization, the cost of a cover is defined as the worst-case delay of the circuit. Both Chaudhary and Pedram [5], and Touati [17] extend these works to solve the minimum area problem under delay constraints. However, they consider only synchronous static circuits and employ pessimistic static timing analysis to determine the worst-case critical paths.

For fundamental mode designs, such as burst-mode circuits, Siegel and De Micheli show that with only small modifications, synchronous technology mapping technique can be applied to asynchronous circuits [16]. They use Unger's result [18] to perform hazard-free

decomposition and present an algorithm to identify library gates which might be hazardous for mapping. Their results demonstrate that most library gates can be used safely except some complex gates. The key shortcoming of this technique is that the underlying synchronous technology mappers are limited to optimizing worst-case performance, not average-case performance.

In [2, 3], Beerel et al. extended these works to perform decomposition and covering that optimized the average-case delay of the burst-mode asynchronous control circuit. The possible inputs to these circuits are given by a set of completely-specified patterns each of which is associated with a frequency of occurrence. Then, an input-pattern-dependent approach is used to minimize the weighted sum of the delay incurred by each input pattern, thereby minimizing average-case delay. The techniques used include rotating the NAND-decomposed network to push more frequent primary inputs closer to the input and a dynamic-programming-based technique to explore mappings of the optimized decomposed network that are deemed likely to minimize average-case delay.

Here, we further extend this work to combinational circuits implemented using domino circuits and one-hot outputs. Unlike burst-mode circuits, the possible inputs to these circuits are specified with a set of *incompletely-specified* patterns each of which is associated with a frequency of occurrence which complicates the technology mapping problem.

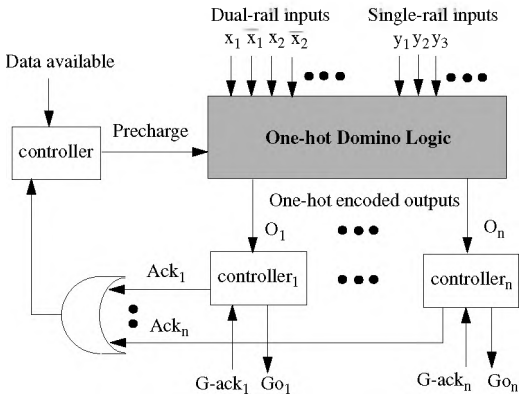

Figure 1: A block diagram of the *one-hot domino* logic design style for combinational circuits.

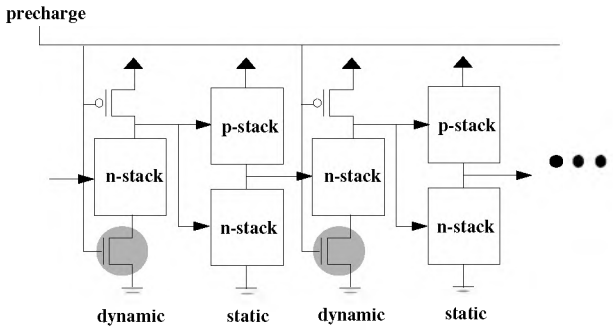

Figure 2: An illustration of domino logic.

3 One-hot domino logic

The basic block diagram of a one-hot domino combinational logic block in an environment is shown in Figure 1. This section describes the structure and operation of the logic as well as its advantages over other currently known approaches.

3.1 The domino core

Domino logic is widely used in high-speed circuits because of its inherent performance advantages. It has smaller parasitic capacitance [20] and separates the pull-up and pull-down events to avoid the fight between the precharge and discharge current [21], often yielding circuits that are faster than circuits obtainable with static CMOS.

Domino logic consists of two types of gates: static CMOS and dynamic precharged gates, both of which must be inverting. As illustrated in Figure 2, the type of gates alternates along any path from inputs to outputs. This is sometimes referred to as the *domino constraint*. Notice that we allow the static gate to be *any* inverting CMOS gate [21], whereas, traditionally, the static gate is restricted to be an inverter [20].

Notice that all dynamic (static) gates precharge (discharge) simultaneously during the precharge phase. Thus, the precharge time is fast, and essentially data-independent. Consequently, we need only optimize the evaluation delay of the circuit.

The gates closest to the primary inputs, referred to as *PI gates*, should be dynamic rather than static. This is because the primary inputs can be assumed to be stable but it is not known whether they will be stable 1 or stable 0 at the start of evaluation or precharge. Consequently, if the PI gate is static, a stable 0 at the primary input may cause a value of 1 to appear at the input to the subsequent dynamic gate at the start of evaluation, possibly causing accidental discharge.

Although we restrict our mapped circuits to the style of domino logic depicted in Figure 2, we note that it may be desirable to further optimize the circuits after technology mapping. For example, in some cases, the pull-down transistor driven by the precharge

line (shaded in Figure 2) can be removed creating what is sometimes called *semi-controlled* domino logic. This can lead to faster evaluation times because it reduces the stack size, but may lead to significant short-circuit current during precharge [21]. The short-circuit current sometimes creates reliability problems, which can be avoided by *staggering* the precharge signal [23].

In addition, we note that charge-sharing problems are always crucial to domino circuits [20]. We assume that either the problems are minimized by precharging each transistor of the pull-down network of each dynamic gate or that further charge-sharing analysis is applied to the mapped circuits.

A key feature of domino logic is that, when designed properly, it can have only monotonic transitions [20]. Consequently, by its very nature, it is hazard-free. It can therefore be easily used in asynchronous circuits by controlling the precharge signal via an asynchronous controller rather than a global clock [8, 23, 22]. Further descriptions of the expected operation of this controller will be given below.

One complication of domino logic is that one stage of domino logic can implement only those functions which are monotonic in their inputs. In particular, binate functions cannot be implemented. Fortunately, this is not a serious limitation because by introducing some dual-rail primary inputs, any function can be implemented [20].

3.2 Completion sensing

A naive means of detecting completion of a one-hot encoded combinational logic block is to explicitly derive a done signal from the logical OR of all one-hot encoded outputs. When the done signal rises, the subsequent operation can then be initiated. This means that the start of the next operation is delayed by at least the delay associated with a possibly wide OR gate. Fortunately, there are many instances in which a much better approach can be used.

Consider, for example, the case in which each output O_i should initiate a different operation i, as depicted in Figure 1. To implement this, a different controller associated with each operation can be used. When the i-th controller senses signal O_i rising, it can trigger the start of the next operation (by rising Go_i) simultaneously with acknowledging the completion of the one-hot logic (by rising Ack_i). The logical OR of all acknowledgments, Ack_i, can trigger the precharge phase. Thus, the completion sensing delay of the evaluation phase can be completely hidden. We note that this approach was recently used by Benes et al. in the implementation of a high-speed decompression circuit for embedded processors [4].

3.3 Precharge phase

In a purely speed-independent implementation the precharging of the logic block must also have some type of completion detection. In this design style, however, timing assumptions can be used to remove the need for an explicit completion detection mechanism. Specifically, all dynamic gates are simultaneously precharged making the precharge time essentially fixed and data-independent. Consequently, control circuitry can easily be used to guarantee that the precharge signal does not become de-asserted until after all gates have been precharged. If some gates are semi-controlled, however, a delay line may be necessary to model the precharge delay [4]. An efficient technique to combine the delay line with the precharge logic for an asynchronous adder is described in [23].

3.4 Comparison to other approaches

We first contrast one-hot domino logic with traditional single-rail, bundled-data approaches in which the output control signals are all latched and the output of the latches, which are guaranteed to be hazard-free, are used to drive controllers. Using one-hot hazard-free outputs completely avoids the latch overhead, including the latch propagation delay and the set-up and hold-times. Moreover, single-rail techniques must minimize the worst-case delay among all outputs for all input combinations. Using one-hot techniques, each output can be independently minimized to prioritize the most frequently occurring input combinations which make it fire. Our experimental results suggest that this flexibility can lead to significant speed advantages. The disadvantage of this technique compared with single-rail approaches is that one-hot logic may be larger, domino logic typically consumes more power, and domino logic often requires careful attention to layout to ensure correct operation.

We note that it is also possible to build these combinational circuits using the speculative completion signaling approaches proposed by Nowick et al. [12, 13]. In this approach, the core logic can be optimized for the common case and side logic can be created to identify when common input data arrives and trigger the done signal to designate that the result is obtained. This approach can lead to some reduction in the average-case delay, but it is unclear how easy it would be to generate the side logic for general functions. The advantage of speculative completion approaches is that they can be applied to static logic, which is simpler to design.

We also note that the concept of using domino circuits in asynchronous designs is not new. For example, Williams demonstrated the power of domino circuits

very convincingly in his landmark asynchronous divider [22]. In addition, Yun et al. used it effectively in asynchronous adder and multiplier designs [23].

4 Technology mapping

We now describe how we can extend the technology mapping techniques in [2, 3] to accommodate one-hot domino logic. In particular, we show how we perform technology mapping in the presence of incompletely-specified input patterns and the domino constraint.

4.1 Incompletely-specified patterns

An incompletely-specified pattern is a function from primary input variables to the set $\{0, 1, X\}$. The problem is that the delay of the circuit for an incompletely-specified pattern cannot be precisely defined because the exact set of gates that will evaluate is unknown in the presence of a primary input with the value "X". Moreover, since the exact set of evaluating gates is unknown, it is unclear which paths the technology mapper should optimize.

To address these problems, we interpret an incompletely-specified pattern as a collection of minterms over the input variables, where each minterm corresponds to a *compatible completely-specified pattern*. Formally, a completely-specified pattern is a function from primary inputs variables to the set $\{0, 1\}$. A completely-specified pattern i is compatible with an incompletely-specified pattern c if the assignments agree on all input variables not assigned to "X" in c.

It is clear that the circuit delay for a completely-specified pattern is well defined and can be established through simulation. Consequently, we can define a range of delays for an incompletely-specified pattern c as follows. The *minimum* (*maximum*) of the range is the smallest (largest) circuit delay incurred by any compatible pattern. Note that the number of compatible patterns can be exponential in the number of circuit variables. Thus, exhaustively simulating all compatible patterns is computationally very expensive.

Fortunately, the special nature of domino logic can be used to simplify this analysis. Specifically, this section proves that two easily identifiable compatible patterns, referred to as *representative* patterns, yield the lower and upper bounds of the pattern delay for an incompletely-specified pattern. The section then describes how we can use these representative patterns in technology mapping.

4.1.1 Bounding the delay of incompletely-specified patterns: intuition

The intuition behind our theory may be described with an analogy to the game called dominos (which is the origin of the name "domino logic"). In this game, rectangular tiles are often arranged in a linear fashion (or sometimes in more complex networks) such that the first tile falling causes a chain reaction of falling tiles. The delay of the chain reaction is the time in between the first and last tile falling. Notice that more than one tile can fall simultaneously to start the chain reaction and that some tiles may remain standing after the chain reaction completes.

Consider further the case where the set of tiles that start the reaction is not fully specified. In particular, consider the case where certain combinations of tiles can be chosen to start the chain reaction but the choice of which combination is unknown. In this case, the chain reaction delay cannot be determined. However, a lower bound on the chain reaction delay can be obtained by tipping over any tile which is tipped over in any combination. Similarly, an upper bound on the chain reaction delay can be obtained by tipping over only those tiles which are tipped over in all combinations.

The analogy is that a domino gate is like a tile. We say a dynamic (static) gate *evaluates* if its output falls (rises). Gates that evaluate are like tiles that fall; they cannot return to their original value until the precharge phase. When one gate evaluates it can cause other gates to evaluate in what is like a chain reaction. Moreover, the evaluation delay is analogous to the delay of the chain reaction. Finally, an incompletely-specified input pattern is analogous to the situation where the set of tiles that starts the chain reaction is not fully specified.

Thus, to find a lower bound of the delay for an incompletely-specified pattern, we force any PI gate that evaluates under any compatible pattern to evaluate. Similarly, to find the upper bound of the delay we force only those PI gates that evaluates under all compatible patterns to evaluate.

In our application, the PI gates are restricted to be dynamic (see Section 3.1). Thus, to find the lower bound we set all unknown inputs to one. Similarly, to find the upper bound we set all unknown inputs to zero.

More formally, we define two *representative* patterns for an incompletely-specified pattern c. The *lower pattern* c_l is obtained by switching all $X's$ in c to 1 and yields a lower bound of c's pattern delay. Similarly, the *upper pattern* c_u is obtained by switching all $X's$ in c to 0 and yields an upper bound of c's pattern delay.

It is important to note that the bound is loose in the presence of dual-rail inputs a^T and a^F since in reality both a^T and a^F cannot be set to the same value.

4.1.2 Bounding the delay of incompletely-specified patterns: theory

This section formalizes our intuition. First, we introduce some additional terminology.

Definition 4.1 (Controlling input) *An input f of a gate g is a controlling input of g iff f has a value or a transition which independently forces g to evaluate. An input which is not controlling is referred to as non-controlling.*

Given a pattern i, let $FC(i,g)$ denote the set of controlling inputs to g. Similarly, $FNC(i,g)$ denotes the set of $g's$ non-controlling inputs. Let g_f denote a gate which connects g to its input f. Let $d(g,f,i)$ denote a pin-to-pin delay of g for input f. If g evaluates and has a controlling input, the *pattern arrival time* of g for pattern i, denoted $pat(i,g)$, is defined as follows:

$$pat(i,g) = \min_{f \in FC(i,g)} [pat(i,g_f) + d(g,f,i)] \quad (1)$$

If g evaluates but has only non-controlling inputs, $pat(i,g)$ is defined as follows:

$$pat(i,g) = \max_{f \in FNC(i,g)} [pat(i,g_f) + d(g,f,i)] \quad (2)$$

Each gate's pattern arrival time can be computed by recursively applying Equation 1 and 2 in postorder of gates in the circuit. Note that since the circuit is one-hot encoded, any input pattern can make only one PO gate (any gate that drives a primary output) evaluate. Let $po(i)$ denote a function which returns the evaluating PO when pattern i is applied. The *pattern delay* of the circuit for pattern i, denoted p_delay_i, is equal to $pat(i, po(i))$.

In addition, let F_i denote the set of all PIs whose value is 1 when pattern i is applied. Moreover, let $FI(g)$ denote the set of all inputs of gate g and let $E(i,k)$ denote the set of all evaluating gates in level k of the circuit when pattern i is applied.

The following two lemmas prove our intuition that the representative patterns c_l (c_u) yields the lower (upper) bound of the delay for an incompletely-specified pattern c. Informally speaking, the first lemma proves that the more PIs set to one the more gates will evaluate and the second lemma proves that the more gates that evaluate the smaller the resulting pattern delay. Their proofs are given in the appendix.

Lemma 4.1 *If all PI gates are dynamic and $F_i \subseteq F_j$, then $E(i,l) \subseteq E(j,l)$ for every level l.*

Lemma 4.2 *If all PI gates are dynamic and $F_i \subseteq F_j$, then, for every level l and all $g \in E(i,l)$, we have that $pat(j,g) \leq pat(i,g)$.*

The following corollary follows directly from the application of Lemma 4.2 on the primary outputs from which it is easy to conclude our argument.

Corollary 4.1 *If all PI gates are dynamic, $F_i \subseteq F_j$ then $p\_delay_j \leq p\_delay_i$.*

Theorem 1 *Let c_l and c_u be the lower and upper pattern of an incompletely-specified input pattern c, respectively. Assuming all PI gates are dynamic, then for all c, $p\_delay_{c_l}$ ($p\_delay_{c_u}$) is a lower (upper) bound of all pattern delays for all completely-specified patterns that are compatible with c.*

Proof: Consider a completely-specified pattern i that is compatible with c. Since pattern c_l (c_u) is generated by switching all $X's$ in pattern c to 1 (0), $F_{c_u} \subseteq F_i \subseteq F_{c_l}$. Therefore, according to Corollary 4.1, $p\_delay_{c_l} \leq p\_delay_i \leq p\_delay_{c_u}$. □

4.1.3 Optimizing for representative patterns

As mentioned earlier, the technology mapping algorithms presented in [2, 3] cannot handle input combinations described using incompletely-specified patterns. One means of working with incompletely-specified patterns is to optimize with respect to *all* compatible patterns. However, this has two problems. First, it is unknown how the probability of an incompletely-specified pattern is distributed over all of its compatible patterns. Thus, only approximate measures of overall pattern delay could be computed. Second, since the number of compatible patterns could be quite large, analyzing all compatible patterns independently can be computationally intractable.

In this paper, we propose to optimize the circuit for one representative pattern for each incompletely-specified pattern. The choice of representative patterns is very important and different input representative patterns can lead to very different results.

In this paper, we tested two sets of representative patterns to optimize for. For a set of incompletely-specified input patterns C, we define $L = \{c_l | \text{ for all } c \in C\}$ as the *lower set* of C, and $U = \{c_u | \text{ for all } c \in C\}$ as the *upper set* of C. We run the optimization procedure twice, once optimizing the benchmark for the lower set and once optimizing the benchmark for the upper set. Since the average-case delay is the weighted sum of all pattern delays for all incompletely-specified patterns [2, 3], we can easily conclude that the average-case delay for the lower (upper) set is the lower (upper) bound of the average-case delay for the original incompletely-specified patterns.

Therefore, for each of the two optimization results, we use the upper and lower sets again to obtain a range of average-case delay. Then, we let the user select the better result.

4.2 Handling the domino constraint

Recall that the input to the covering is a NAND-decomposed DAG, referred to as a *subject graph*. Our goal is to cover the subject graph with a set of library gates which are all inverting and either static or dynamic. Let $\langle N, E \rangle$ be a subject graph where N is a set of nodes and, E is a set of edges ($E \subseteq N \times N$).

Recall also, that one stage of domino logic can implement only monotonic logic. This limitation is manifested in technology mapping by the fact that not all decomposed networks can be mapped using domino logic. Consider the decomposed network in which there are two reconvergent fanout paths from u to v, where u is a gate driven by a primary input. Let the first be $u, n_1, n_2, \ldots, n_l, v$ and let the second be $u, n'_1, n'_2, \ldots, n'_{l'}, v$. If l and l' are both odd (both even) then the domino constraint demands that v is implemented with a dynamic (static) gate. If l is even and l' is odd (or vice-versa) then no mapping exists. Fortunately, this situation can be resolved by duplicating portions of the NAND-decomposed network and introducing dual-rail inputs [20]. The result is an altered NAND-decomposed graph which is domino-feasible, as defined below.

Definition 4.2 (Domino-feasible DAG) *A domino-feasible DAG is a triple $\langle N, E, \lambda \rangle$, where N is a set of nodes and, E is a set of edges ($E \subseteq N \times N$) and λ is a labeling function $NI \rightarrow \{Dynamic, Static\}$ that satisfies $\lambda(u) = Dynamic$ for all $u \in I$ and $\lambda(u) \neq \lambda(v)$ if $(u, v) \in E$ where $u, v \in NI$.*

Then, to extend the technology mapping technique in [2, 3] to domino circuits we simply restrict the matching of static (dynamic) nodes to only static (dynamic) gates. The remaining parts of the algorithm need not be changed and we refer the reader to [2, 3] for more details.

5 A case study

We now describe the key combinational block of an asynchronous instruction length decoder (AILD). The overall architecture of the instruction decoder and the associated control circuits are outside of the scope of this paper and will hopefully be reported in separate papers.

5.1 Instruction format

Figure 3 shows the general instruction format for the Pentium® processor [1]. Instructions consist of 4 optional instruction prefixes, opcode bytes, an optional

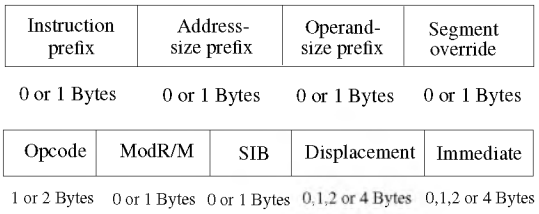

Figure 3: The Pentium® instruction format.

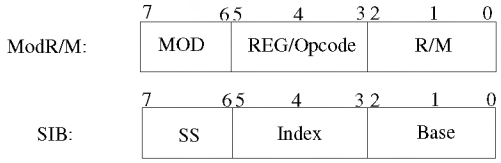

Figure 4: The ModR/M and SIB fields.

address specifier consisting of the ModR/M byte and the SIB (Scale Index Base) byte, and optional displacement and immediate fields.

Each prefix is one byte long. Only the operand-size prefix and the address-size prefix affect the instruction length. Because these are very rare, we choose to trap and handle them using slower exception logic which will not be discussed here. The opcode represents the operation of the instruction. It identifies the size of the operation, the displacement, and the immediate. It is either one byte long or two bytes long where the first byte is always 0F. The ModR/M byte identifies a special addressing form for instructions that refer to an operand in memory. The ModR/M byte always follows the opcode. Some ModR/M bytes are followed by the SIB byte, a second addressing byte. ModR/M and SIB also determine the existence and size of the displacement and immediate. The displacement follows the opcode, or ModR/M, or SIB (which ever is last). The immediate, if present, is always the last field of an instruction. Both the displacement and immediate fields can be one, two, or four bytes long. The maximum valid instruction length is 15 bytes. Figure 4 shows the ModR/M and SIB byte format. The details of each field can be found in [1].

5.2 Instruction length frequencies

The motivation of the asynchronous design stems from an analysis of several benchmark programs in which instruction lengths are monitored. This analysis led to the frequency histogram presented in Figure 5. This chart clearly shows that instructions of lengths two and three are very frequent, whereas others are much less frequent. Instructions of length greater than seven are extremely rare. This motivates our design to be optimized for instructions of length 7 or less. Longer

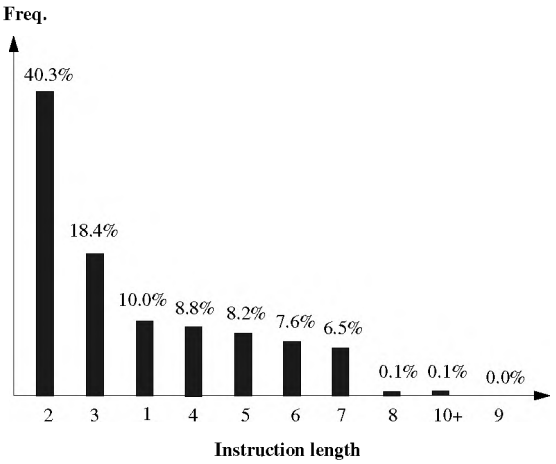

Figure 5: The frequency of instruction lengths.

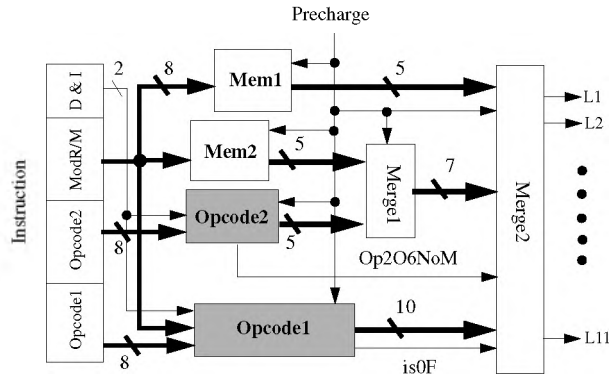

Figure 6: The block diagram of the asynchronous instruction length decoder.

instructions are handled separately using slower logic that is not discussed here.

5.3 One-hot domino logic blocks

One-hot domino logic forms the combinational block that inputs an instruction and yields the one-hot encoded instruction length for the instructions with lengths less than 7. Specifically, as shown in Figure 6, this block is decomposed into 6 one-hot domino logic blocks: Opcode1, Opcode2, Mem1, Mem2, and two length merging blocks, Merge1 and Merge2. The Opcode1 and Opcode2 blocks compute the length contributed by the first and second opcode byte, respectively. The Mem1 and Mem2 blocks compute the length contributed by the ModR/M byte for one-byte and two-byte opcodes, respectively. The two merging blocks add these contributions to form the final length outputs.

The Opcode1 block generates the following 11 one-hot encoded outputs: Op1O1NoM, Op1Oc2M1, Op1O2NoM, Op1Oc3M1, Op1O3NoM, Op1Oc4M1, Op1O4NoM, Op1O5NoM, Op1Oc6M1, Op1O7NoM, and is0F. Op1Oc2M1, for example, denotes that the first byte of the instruction is the only opcode byte and it contributes two bytes for the total length and the ModR/M byte is present. Op1O2NoM denotes the same information as Op1Oc2M1 except that no ModR/M byte is present. The other outputs have similar interpretations. Note that Op1O6NoM, for example, is not possible. The is0F output is asserted when the opcode consists of two bytes (in which case the first byte must be 0F).

The Opcode2 block generates 6 one-hot encoded outputs defined similarly. Op2Oc3M2, for example, denotes that the second opcode byte contributes three bytes (including the first opcode byte 0F) and the ModR/M byte is present.

The Mem1 (Mem2) block checks the ModR/M byte for the one-byte (two-byte) opcode to generate 5 one-hot encoded outputs: M10, M11, M12, M14, and M15 (M20, M21, M22, M24, and M25). These represent that the ModR/M byte contributes 0, 1, 2, 4, and 5 bytes for the total length, respectively.

The Merge1 block combines the Opcode2's outputs (except Op2O6NoM) and the Mem2's outputs to obtain the length for the instructions having a two-byte opcode (see Table 1). The Merge2 block then combines the outputs of the Opcode1, Mem1, and Merge1 (along with the Op2O6NoM from the Opcode2) to obtain the final one-hot length outputs, as defined in Table 2. This configuration means that the instructions having a two-byte opcode will have longer length computation time than the instructions having a one-byte opcode except the one represented by the Op2O6NoM. This improves the average-case delay of the length computation because most one-byte-opcode instructions are more frequent than the two-byte-opcode instructions. The Op2O6NoM is chosen to be fed directly to the Merge2 since it is also frequent and it need not be ANDed with any Mem2's output.

L	Out	Equation
3	L3_0F	Op2Oc3M2*M20 + Op2O3NoM
4	L4_0F	Op2Oc3M2*M21 + Op2Oc4M2*M20 + Op2O4NoM
5	L5_0F	Op2Oc3M2*M22 + Op2Oc4M2*M21
6	L6_0F	Op2Oc4M2*M22
7	L7_0F	Op2Oc3M2*M24
8	L8_0F	Op2Oc3M2*M25 + Op2Oc4M2*M24
9	L9_0F	Op2Oc4M2*M25

Table 1: The length equations implemented in the Merge1 block.

L	Out	Equation
1	L1	Op1O1NoM
2	L2	Op1Oc2M1*M10 + Op1O2NoM
3	L3	Op1Oc2M1*M11 + Op1Oc3M1*M10 + Op1O3NoM + is0F*L3_0F
4	L4	Op1Oc2M1*M12 + Op1Oc3M1*M11 + Op1Oc4M1*M10 + Op1O4NoM + is0F*L4_0F
5	L5	Op1Oc3M1*M12 + Op1Oc4M1*M11 + Op1O5NoM + is0F*L5_0F
6	L6	Op1Oc2M1*M14 + Op1Oc4M1*M12 + Op1Oc6M1*M10 + is0F*Op2O6NoM + is0F*L6_0F
7	L7	Op1Oc2M1*M15 + Op1Oc3M1*M14 + Op1Oc6M1*M11 + Op1O7NoM + is0F*L7_0F
8	L8	Op1Oc3M1*M15 + Op1Oc4M1*M14 + Op1Oc6M1*M12 + is0F*L8_0F
9	L9	Op1Oc4M1*M15 + is0F*L9_0F
10	L10	Op1Oc6M1*M14
11	L11	Op1Oc6M1*M15

Table 2: The length equations implemented in the Merge2 block.

5.4 Product term frequencies

For each combinational logic block, a two-level minimizer is used to obtain an optimized set of product terms. Then, architectural simulations is used to obtain frequency statistics of each product term. We then associate with each product term an incompletely-specified pattern and use the normalized product-term frequencies as an estimate of the frequency of the incompletely-specified pattern. The resulting frequency distributions of the incompletely-specified patterns for the Op1O1NoM and Op1Oc2M1 outputs are given in Figure 7.

The distributions of all patterns for all outputs of both the Opcode1 and Opcode2 blocks, along with the output's optimized NAND-decomposed network, are then input to our technology mapping program.

5.5 Experimental results

This section reports the technology mapping results for both the Opcode1 and Opcode2 blocks which are the shaded blocks in Figure 6. A summary of the complexity of each output logic is given in Table 3. Notice that the fourth column reports the number of incompletely-specified input patterns which cause the output to evaluate to 1. The fifth column reports the number of nodes in the NAND-decomposed DAG. The sixth column reports the relative frequency of each output evaluating to a 1.

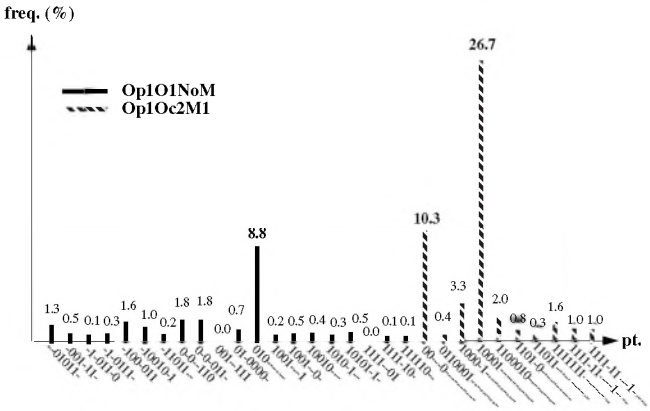

Figure 7: The frequency distribution of product terms of Op1O1NoM and Op1Oc2M1. The first opcode byte and the ModR/M byte are inputs of the product terms. For Op1O1NoM, the inputs from the ModR/M byte are don't-cares (not shown for simplicity).

Note that all mappings are performed using the *lib2* gate library (that is available in the tool SIS [15]) which is modified in two ways. First, we remove all non-inverting gates because such gates cannot be used in domino logic. Second, for each inverting static gate, we add a corresponding dynamic gate with the same area and delay characteristics. This made it possible for us to compare our results with those obtained using worst-case mapping techniques that do not ensure the domino constraint [5]. All experiments were performed on a 120-MHz Pentium® Processor with manageable CPU times.

Table 4 reports the average-case delays obtained by optimizing the logic for both the lower set (the 2nd and 3rd columns) and the upper set (the 4th and 5th columns). Not surprisingly, the results indicate that when we optimize for the lower pattern set, the lower bound is typically smaller than when we optimize for the upper pattern set. Similarly, optimizing for the upper pattern set leads to smaller upper bounds. When comparing circuits, we always try to be conservative and thus report the upper bound of our circuits. Consequently, it appears that optimizing the upper bound of our circuits generally leads to more favorable conservative comparisons.

Interestingly, the ranges in average-case delay obtained by optimizing for the upper pattern set are always smaller than those obtained by optimizing for the lower pattern set. This may be because the critical path for an upper pattern with high frequency is typically very short because it has been highly optimized. Consequently, when the corresponding lower pattern is applied, the path is still critical. On the other hand, when we optimize for the lower pattern set, we may optimize for a critical path that is differ-

Description of each combinational logic output					
Circuit	# PIs	# POs	# Patts.	# Nodes	Freq.
Op1O1NoM	16	1	20	574	0.202
Op1Oc2M1	16	1	10	321	0.473
Op1O2NoM	16	1	9	322	0.114
Op1Oc3M1	18	1	6	280	0.065
Op1O3NoM	18	1	7	307	0.018
Op1Oc4M1	18	1	4	231	0.004
Op1O4NoM	8	1	1	56	0.001
Op1O5NoM	19	1	8	361	0.056
Op1Oc6M1	18	1	4	227	0.015
Op1O7NoM	12	1	2	112	0.000
Op2O2NoM	16	1	16	427	0.001
Op2Oc3M2	16	1	15	379	0.025
Op2O3NoM	6	1	1	42	0.000
Op2Oc4M2	11	1	2	95	0.001
Op2O4NoM	11	1	2	74	0.003
Op2O6NoM	5	1	1	33	0.022

Table 3: Summary of complexity of each combinational logic output.

ent from the one that is critical for the upper pattern, thereby yielding a large range.

Table 4 also presents data derived from circuits obtained using the worst-case mapping techniques described in [5] (columns 7-10). Using this data we can make two major comparisons.

First, we compare the average-case delay of our best circuits (optimized for the upper pattern set) with the average-case delay of circuits obtained with worst-case mapping techniques. This is of interest because it establishes the potential benefit of explicitly optimizing average-case delay during technology mapping. To be conservative, we compare the upper bound of our mapped circuits with the lower bound of the circuits derived using worst-case mapping techniques. The results demonstrate that our circuits are at least 31% faster on average than that of worst-case mapped circuits.

Second, we can compare the average-case delay of our circuits with the worst-case delay of the comparable synchronous circuit. This comparison can give us an estimate of the potential benefit of asynchronous circuits. It is important to note, however, that this estimate assumes that the synchronous circuit adopts the same decomposition of blocks that is described here. Specifically, we cannot account for the possibility that a different decomposition might be better suited for optimizing for worst-case delay. With this caveat stated, the results indicate that our circuits are on average at least 54% faster than the comparable synchronous circuits.

6 Conclusions

The paper focuses on the design of asynchronous combinational circuits that incorporate domino logic and one-hot logic with timing assumptions that are easily met. In particular, we discuss a novel technology mapping technique for this design that leverage off of existing work. We apply this technique to two combinational logic blocks that are an integral part of a fast asynchronous instruction length decoder.

We compare our circuits with those obtained using a more conventional synchronous technology mapper (that optimizes for worst-case delay). Our experimental results suggest that our mapped circuit is at least 31% faster than the average-case delay of the conventionally-mapped circuit, illustrating the utility of our new technique. Moreover, the average-case delay of our circuit is more than 50% smaller than the (worst-case) delay of the conventionally-mapped circuit, demonstrating the potential advantage of asynchronous one-hot domino circuits over both synchronous implementations and conventional bundled-data implementations.

Appendix: Proof of lemmas

Lemma 4.1 *If all PI gates are dynamic and $F_i \subseteq F_j$, then $E(i,l) \subseteq E(j,l)$ for every level l.*

Proof: (By induction)

Base: Let $l = 1$. $F_i \subseteq F_j$. Since all PI gates are dynamic, $E(i,1) \subseteq E(j,1)$.

Inductive hypothesis: For $l = k$, $E(i,k) \subseteq E(j,k)$.

Inductive step: Let $l = k+1$. Let $g_{k+1} \in E(i,k+1)$. First consider the case where g_{k+1} has a controlling input $f_k \in FI(g_{k+1})$ which is driven by a gate g_k that evaluates when i is applied. Since $E(i,k) \subseteq E(j,k)$, g_k must evaluate in pattern j. Since the controlling nature of an input is pattern-independent (because an evaluating gate always drives its output to a value that is independent of the pattern applied), f_k must also be a controlling input of g_{k+1} when j is applied. Therefore, g_{k+1} must evaluate when j is applied, i.e., $g_{k+1} \in E(j,k+1)$. Thus, $E(i,k+1) \subseteq E(j,k+1)$.

Now consider the case where g_{k+1} evaluates and all $f_k \in FI(g_{k+1})$ are non-controlling inputs to g_{k+1} in pattern i. Since $E(i,k) \subseteq E(j,k)$, all g_k's must evaluate and all corresponding f_k's must be non-controlling inputs of g_{k+1} when j is applied. Thus, g_{k+1} must evaluate when j is applied, i.e., $g_{k+1} \in E(j,k+1)$. Therefore, $E(i,k+1) \subseteq E(j,k+1)$. □

Lemma 4.2 *If all PI gates are dynamic and $F_i \subseteq F_j$, then, for every level l and all $g \in E(i,l)$, we have that $pat(j,g) \leq pat(i,g)$.*

Average-case Mapping vs. Worst-case Mapping											
Circuit	Average-case (AC)					Worst-case (WC)				Improve	
	$ACD_u^{a,l}$	$ACD_l^{a,l}$	$ACD_u^{a,u}$	$ACD_l^{a,u}$	Area^{a,u}	ACD_u^{w}	ACD_l^{w}	WCD	Area^{w}	AA	AW
Op1O1NoM	3.672	1.570	2.965	2.229	114144	3.802	3.279	5.130	97904	10%	42%
Op1Oc2M1	2.404	2.131	2.186	2.018	55680	3.510	3.459	4.070	55216	37%	46%
Op1O2NoM	1.775	1.698	1.811	1.800	54752	4.012	4.002	4.440	51040	55%	59%
Op1Oc3M1	2.201	2.047	2.170	2.070	43616	3.206	3.151	4.010	46400	31%	46%
Op1O3NoM	2.821	2.821	2.821	2.821	50576	3.542	3.542	4.200	49648	20%	33%
Op1Oc4M1	2.761	2.761	2.761	2.761	33872	3.104	3.104	3.370	39440	11%	18%
Op1O4NoM	1.587	1.587	1.587	1.587	6032	1.587	1.587	1.590	6032	0%	0%
Op1O5NoM	2.800	2.800	2.800	2.800	61248	3.516	3.516	4.200	61248	20%	33%
Op1Oc6M1	2.647	2.647	2.647	2.647	37584	3.181	3.181	3.370	39440	17%	21%
Op1O7NoM	2.400	2.400	2.400	2.400	14384	2.400	2.400	2.750	14384	0%	13%
Ave.(Op1)	2.620	2.017	2.364	2.114	471888	3.604	3.462	5.130	460752	32%	54%
Op2O2NoM	4.391	1.810	2.150	2.043	77024	4.270	4.181	4.790	70528	49%	55%
Op2Oc3M2	3.206	1.507	2.567	2.224	74704	3.624	3.137	4.680	65424	18%	45%
Op2O3NoM	1.400	1.400	1.400	1.400	5104	1.400	1.400	1.440	5104	0%	3%
Op2Oc4M2	1.675	1.675	1.675	1.675	12064	2.403	2.403	2.410	12064	30%	30%
Op2O4NoM	1.537	1.537	1.537	1.537	10208	1.627	1.627	2.070	9280	6%	26%
Op2O6NoM	1.335	1.335	1.335	1.335	4640	1.335	1.335	1.330	4640	0%	0%
Ave.(Op2)	2.311	1.445	1.962	1.794	183744	2.530	2.293	4.790	167040	14%	59%
Ave.(Op1+2)	2.604	1.987	2.343	2.098	655632	3.548	3.401	5.130	627792	31%	54%

Table 4: Delay and area of average-case mapping vs. delay and area of worst-case mapping. ACD denotes the average-case delay while WCD denotes the worst-case delay. Subscripts and superscripts on ACD and Area denote the type of optimization performed and the bound of the average-case delay reported. Specifically, the superscript a denotes the use of our average-case mapper while w denotes the use of the worst-case mapper. The superscripts u and l denote the optimization is performed for the upper set and the lower set, respectively. In contrast, the subscripts u and l denote the numbers reported are the upper and lower bound of the average-case delay, respectively. For the percentage improvements, the numbers in column AA are computed using (1-$ACD_u^{a,u}/ACD_l^{w}$)*100%, and the numbers in column AW are computed using (1-$ACD_u^{a,u}$/WCD)*100%.

Proof: (By induction)

Base: Let $l = 1$. Since $F_i \subseteq F_j$, according to Lemma 4.1, $E(i,1) \subseteq E(j,1)$. For $g \in E(i,1)$, two conditions that $FC(i,g) \subseteq FC(j,g)$ and $FNC(i,g) = FNC(j,g)$ must hold. Thus, $pat(j,g) \leq pat(i,g)$.

Inductive hypothesis: For $l = k$, and for all $g_k \in E(i,k)$, $pat(j,g) \leq pat(i,g)$.

Inductive step: Let $l = k+1$. Consider $g_{k+1} \in E(i,k+1)$.

Case 1: g_{k+1} has a controlling input. According to Equation 1, $pat(i,g_{k+1}) = \min_{f_k \in FC(i,g_{k+1})} pat(i,g_k) + d(i,g_{k+1},f_k)$, and $pat(j,g_{k+1}) = \min_{f_k \in FC(j,g_{k+1})} pat(j,g_k) + d(j,g_{k+1},f_k)$. According to Lemma 4.1, since g_k evaluates when i is applied, g_k must evaluate when j is applied. Since f_k is a controlling input for g_{k+1} in i, we know that it must be a controlling input for g_{k+1} in j. Thus, $FC(i,g_{k+1}) \subseteq FC(j,g_{k+1})$. By the inductive hypothesis we also know that $pat(j,g_k) \leq pat(i,g_k)$.

Moreover, we know that the pin-to-pin delay of an evaluating gate is pattern independent, i.e., $d(i,g_{k+1},f_k) = d(j,g_{k+1},f_k)$. Therefore, we conclude that $pat(j,g_{k+1}) \leq pat(i,g_{k+1})$.

Case 2: g_{k+1} has only non-controlling inputs. From Equation 2, $pat(i,g_{k+1}) = \max_{f_k \in FNC(i,g_{k+1})} pat(i,g_k) + d(i,g_{k+1},f_k)$, and $pat(j,g_{k+1}) = \max_{f_k \in FNC(j,g_{k+1})} pat(j,g_k) + d(j,g_{k+1},f_k)$. According to Lemma 4.1, all g_k's that evaluate in i must evaluate in j. Since all f_k's are non-controlling in i they must be non-controlling in j. Therefore, $FNC(i,g_{k+1}) = FNC(j,g_{k+1})$ must hold. Also, by the inductive hypothesis we know that $pat(j,g_k) \leq pat(i,g_k)$. Moreover, we know that $d(i,g_{k+1},f_k) = d(j,g_{k+1},f_k)$. Thus, we conclude that $pat(j,g_{k+1}) \leq pat(i,g_{k+1})$. □

Acknowledgments

We would like to acknowledge Peter Yeh, You-Pyo Hong, and Aiguo Xie of the University of Southern California for help comments on this paper.

References

[1] Intel architecture software developer's manual, volume 2: Instruction set reference manual. http://developer.intel.com/design.

[2] P. A. Beerel, K. Y. Yun, and W. -C. Chou. A heuristic covering technique for optimizing average-case delay in the technology mapping of asynchronous burst-mode circuits. In *Proc. European Design Automation Conference (EURO-DAC)*, September 1996.

[3] P. A. Beerel, K. Y. Yun, and W. -C. Chou. Optimizing average-case delay in technology mapping of burst-mode circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April 1996.

[4] M. Benes, A. Wolfe, and S.M. Nowick. A high-speed asynchronous decompression circuit for embedded processors. In *Proceedings of the 17th Conference on Advanced Research in VLSI*, Los Alamitos, CA, september 1997. IEEE Computer Society Press.

[5] K. Chaudhary and M. Pedram. Computing the area versus delay trade-off curves in technology mapping. *IEEE Transactions on Computer-Aided Design*, pages 1480–1489, December 1995.

[6] A. Davis and S. M. Nowick. Asynchronous circuit design: Motivation, background, and methods. In Graham Birtwistle and Al Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pages 1–49. Springer-Verlag, 1995.

[7] S. B. Furber, J. D. Garside, S. Temple, J. Liu, P. Day, and N.C. Paver. AMULET2e: An asynchronous embedded controller. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, April 1997.

[8] S. B. Furber and J. Liu. Dynamic logic in four-phase micropipelines. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, March 1996.

[9] J. Kessels and P. Marston. Designing asynchronous standby circuits for a low-power pager. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, April 1997.

[10] A. J. Martin, S. M. Burns, T. K. Lee, D. Borkovic, and P. J. Hazewindus. The design of an asynchronous microprocessor. In Charles L. Seitz, editor, *Advanced Research in VLSI: Proceedings of the Decennial Caltech Conference on VLSI*, pages 351–373. MIT Press, 1989.

[11] C. J. Myers. *Private communication*, July 1995. C. J. Myers is an assistant professor at the University of Utah.

[12] S. M. Nowick. Design of a low-latency asynchronous adder using speculative completion. *IEE Proceedings, Part E, Computers and Digital Techniques*, 143(5):301–307, September 1996.

[13] S. M. Nowick, K. Y. Yun, P. A. Beerel, and A. E. Dooply. Speculative completion for the design of high-performance asynchronous dynamic adders. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, April 1997.

[14] R. Rudell. *Logic Synthesis for VLSI Design*. PhD thesis, U. C. Berkeley, April 1989. Memorandum UCB/ERL M89/49.

[15] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, University of California, Berkeley, May 1992.

[16] P. Siegel, G. De Micheli, and D. Dill. Automatic technology mapping for generalized fundamental-mode asynchronous designs. In *Proc. ACM/IEEE Design Automation Conference*, pages 61–67, June 1993.

[17] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang. Performance-oriented technology mapping. In W. J. Dalley, editor, *6th MIT Conference on Advanced VLSI Conference*, pages 79–97, 1995.

[18] S. H. Unger. *Asynchronous Sequential Switching Circuits*. Wiley-Interscience, John Wiley & Sons, Inc., New York, 1969.

[19] K. van Berkel, R. Burgess, J. Kessels, M. Roncken, F. Saeijs, and A. Peeters. Asynchronous circuits for low power: A DCC error corrector. *IEEE Design & Test of Computers*, pages 22–32, Summer 1994.

[20] N. H. E. Weste and K. Eshraghian. *Principles of CMOS VLSI Design*. Addison-Wesley, 2nd edition, 1993.

[21] T. E. Williams. Dynamic logic: Clocked and asynchronous, 1996. ISSCC Tutorial.

[22] T. E. Williams and M. A. Horowitz. A zero-overhead self-timed 160ns 54b CMOS divider. *IEEE Journal of Solid-State Circuits*, 26(11):1651–1661, November 1991.

[23] K. Y. Yun, P. A. Beerel, V. Vakilotojar, A. E. Dooply, and J. Arceo. The design and verification of a high-performance low-control-overhead asynchronous differential equation solver. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, April 1997.