

CHAPTER 1

INTRODUCTION

Synchronous systems are those controlled by a central clock. These systems have dominated logic design since the advent of the integrated circuit and the necessary design styles and tools have become mature and well understood. However, with trends in the integrated circuit industry pushing performance to physical limitations, the timing issues involved with global clock synchronization have become increasingly difficult to resolve. Numerous studies have shown the significant design burden that clock skew and clock line load has placed on the synchronous system designer [11, 19, 35, 56, 58]. As clock speeds increase, the amount of time available per clock period decreases to the point where serious design problems and even outright failure can occur if these issues cannot be resolved. As a result, logic designers are turning to more aggressive design styles to address this particular issue.

Asynchronous design styles remove the need to address clocking issues by removing the global clock itself. However, asynchronous designs are difficult to implement reliably because of the presence of *hazards*. In many cases, the advantages inherent to asynchronous design are wasted due to the overhead required to implement hazard-free circuits. *Timed circuits* are a class of asynchronous design that uses explicit timing information in circuit synthesis [39]. The use of explicit timing information has been shown to potentially outperform synchronous designs and other asynchronous design styles as well. This was demonstrated in the Intel RAPPID project in which an asynchronous instruction length decoder for an x86 processor was designed using timed circuits which was three times faster while using half the power of the comparable synchronous design [53].

Researchers have developed numerous CAD tools to support asynchronous design styles. These tools have focused on the specification and synthesis stages of the design flow. Designs are specified at a high level and synthesis produces a logic description that is guaranteed to be hazard-free. Since logic synthesis is best optimized when it is not constrained by implementation, the resulting logic equations are unlikely to conform directly to available library parts. Currently, very few CAD tools are available to assist the asynchronous designer in implementing the synthesized equations. *Technology mapping*, also called library binding, is the process whereby a *technology independent* logic representation is mapped to a *technology dependent* library. For timed asynchronous circuits, this process is often done by hand as there are currently no automated CAD tools available that address the issue of hazards.

Synchronous designers have complete CAD tool-sets available from several companies and high-level, technology independent design descriptions can be implemented in available libraries with little or no custom work needed by the designer. These tools and the algorithms they employ cannot be directly applied to timed asynchronous designs because they have little or no ability to deal with hazards. Although performance pressure is causing designers to expand their consideration of other design methodologies, the adoption of timed asynchronous circuits is unlikely until the supporting CAD tools are complete and have been tested on industry grade designs. In addition, few designers are aware of these alternative methodologies because they remain primarily in the purview of research labs and universities, and as such, have seen little exposure to market driven pressures.

Our work attempts to bridge the gap between synchronous technology mapping techniques and the technology mapping of timed asynchronous circuits. Our goal is to complete the design flow for timed asynchronous circuits by developing an efficient and verifiable methodology for identifying and eliminating hazards created during technology mapping.

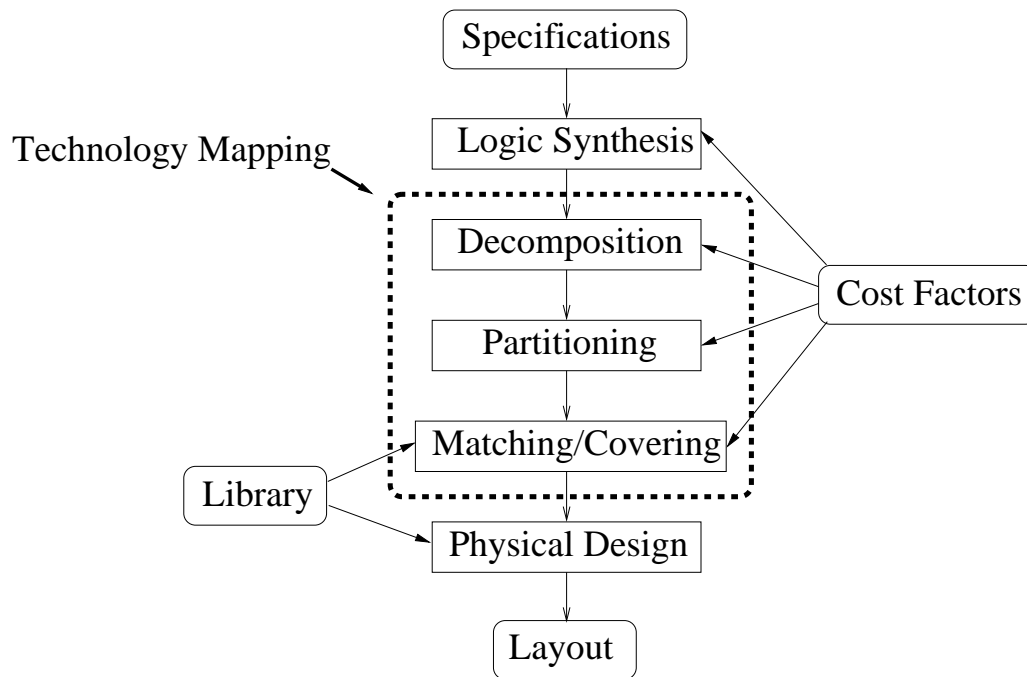


Figure 1.1. Synchronous design flow.

1.1 Synchronous Design Flow

The design flow for synchronous systems is shown in Figure 1.1. The process of automatic synthesis starts with a technology independent description of the design and ends with a technology dependent transistor level implementation.

The subject of fully automated systems for synchronous design has been an active area of research for the past 20 years [6, 7, 13, 22]. The theory and tools are mature and a number of companies such as Synopsys, Cadence, and Mentor Graphics currently produce complete tool sets allowing the designer an automated means of producing fully functional and verifiable designs. The design process typically starts with an abstract high-level behavioral description of the circuit. Over the years, this has migrated from schematics or RTL descriptions to high-level languages such as VHDL [54] or Verilog [55]. This allows for better maintenance, portability, and reuse of existing design efforts.

The logic synthesizer takes this high level description and creates a logical

representation of the circuit implementation. This process is often guided by a cost metric such as area, power, or delay and the resulting logic equations reflect this metric. For instance, a one-hot encoding scheme may be used in a synchronous state machine to reduce the next-state logic which in turn reduces the combinational path delay. This allows for a faster clock but at the cost of increased area. The synthesis engine works best with few (or no) constraints and the resulting logic equations are seldom optimized. An optimal set of logic equations can then be created from a logic minimization program such as SIS [49].

1.2 Asynchronous Design Flow

Synchronous design flows cannot be used *as is* for asynchronous circuits because of the possibility of introducing hazards on new internal wires. Nor can timing information be used easily since causality is not expressed directly between outputs and inputs in sequential machines. As a result, the design flow shown in Figure 1.1 must

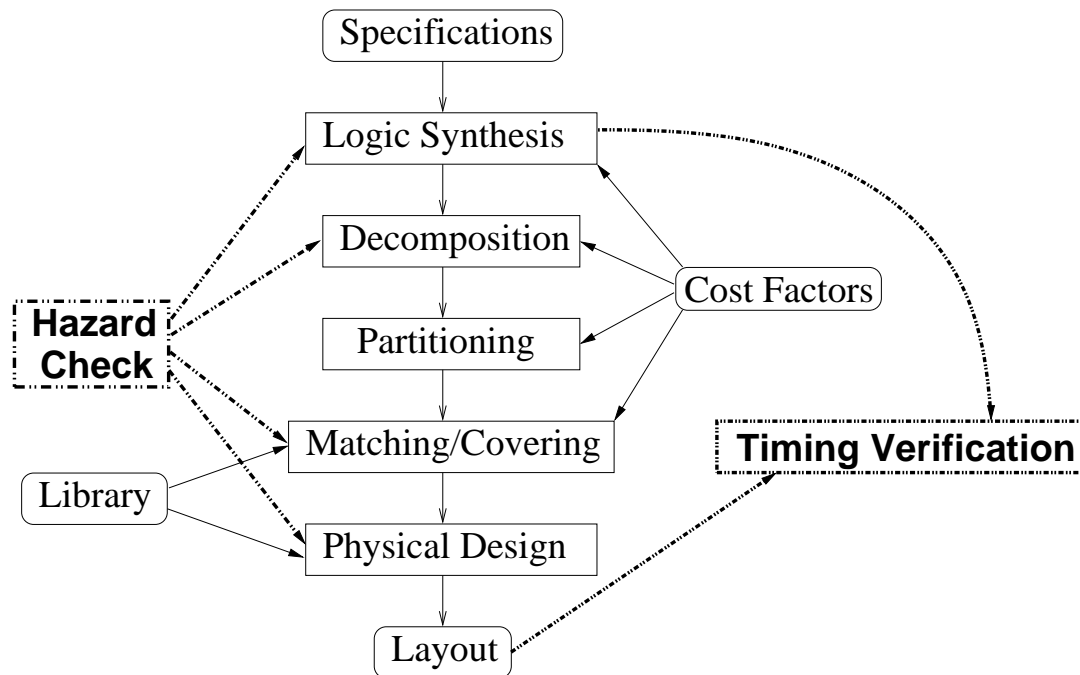


Figure 1.2. Asynchronous design flow.

be modified to accommodate the added restrictions placed on asynchronous design. Figure 1.2 shows the additional complexity associated with these modifications.

Hazard checking must begin in the synthesis stage of the design process. The particular asynchronous design style chosen determines the algorithms used during synthesis to generate a set of hazard-free logic equations. We assume that these logic expressions and supporting circuit descriptions (state graphs, time Petri nets, netlists, etc.) passed to the technology mapper are hazard-free under the constraints specified with the design.

Depending on the particular asynchronous design style, hazards can be introduced during the decomposition stage because the structure of the circuit changes. Each node in the new decomposed netlist must be annotated with hazard properties. This information is then passed along to the matching/covering stage where every attempt is made to eliminate the hazards by affecting the timing properties of the covered netlist or using the *atomic gate assumption* to encapsulate hazardous nodes. An atomic gate is modeled by a single delay element connected to its output and it is assumed that its operation is equivalent to a single-stack cell. Since physical design and layout can affect the timing behavior of the final circuit, timing verification must be performed after the final design is implemented in a given technology to ensure the mapped circuit meets the initial timing constraints. Failure to do so may result in circuits whose behavior is not consistent with the initial specifications.

1.3 Verification

While timed asynchronous circuits offer potential advantages over synchronous circuits such as faster operation and lower power, these advantages are often offset by the expense of the circuit overhead needed to eliminate *hazards*. Hazards are conditions generated by the structure of the circuit or timing relationships between inputs and propagation delays that can cause incorrect behavior. As synthesized hazard-free logic equations are mapped to a given gate library, new internal nodes are introduced in the circuit netlist. Each new internal node as well as the outputs of the circuit must be verified for hazard-freedom to ensure correct operation of

the mapped circuit. This verification must be extremely efficient to allow for many alternative designs to be considered during technology mapping. Current timing verification algorithms [46, 5, 43, 36, 32] often suffer from state explosion problems because each node in the circuit netlist is treated as a new state variable, potentially doubling the number of states.

There have been numerous methods developed for the verification of gate-level *speed-independent* asynchronous circuits [16, 17, 4, 27, 21, 44, 45]. In speed-independent circuits, no timing assumptions are made about gates or the environment. In [4], an efficient verification method for determinate speed-independent circuits is proposed. This work reduces the state space explosion problem found in earlier methods by examining individual behavior at each internal node and approximating this behavior for each state in the specification. The hazard-freedom of the circuit is then verified by examining this *cube approximation*. When the number of internal signals is high as compared with the number of primary inputs and outputs (a feature common of many circuit design styles), this cube approximation technique has the potential to substantially reduce the complexity of verification as demonstrated in the results shown in [4].

1.4 Synchronous Technology Mapping

As previously mentioned, *technology mapping* is the process of binding the technology independent design to a dependent technology. The mapper takes as inputs a technology independent set of logic equations and a library of cells, and produces a technology dependent netlist implementing the circuit from cells found in the library. The process is often optimized for a particular cost metric such as area, speed, power, or in our case, hazard-freedom. Solving this problem exactly is intractable [20] so heuristics have been created to break the problem up into smaller parts to get as good an approximation as possible.

An excellent overview of synchronous technology mapping algorithms is given in [14, 23]. Algorithmic mappers [24, 34, 48] generally break the process up into three phases: decomposition, partitioning, and matching/covering.

The synthesized logic expressions from the initial network are represented as a directed acyclic graph (DAG). These expressions are decomposed into a multi-level circuit composed of simple gates called *base functions*. Base functions are typically two-input AND, OR, NAND, or NOR gates and possibly inverters. The implementation library must include these base functions to guarantee a solution.

Decomposition serves two purposes. First, it guarantees that a solution can be found, that is, a netlist of the logic equations can always be created from the library elements. Secondly, the finer granularity of a decomposed network allows for more matching options and a chance of a higher quality solution. This is particularly important since different matchings may be needed depending on the chosen cost metric.

As an example, Figure 1.3 shows two possible decomposition architectures for a 4-input AND gate. Which architecture is chosen depends on the cost metric. For instance, the decomposition shown in Figure 1.3(b) has a longer worst case path delay than the decomposition shown in Figure 1.3(c) so may not be desirable if delay is the primary cost metric. For our work, explicit timing information is available so it may be possible to order the later arriving inputs closer to the output to mitigate the longer delay or at least affect the hazard properties. The decomposition in Figure 1.3(b) may also prove to be attractive because the architecture is consistent for N-inputs whereas the architecture in Figure 1.3(c) will change depending on the number of inputs.

During partitioning [33], the network is split into single-output *cones* of logic where each cone represents a partition of the circuit obtained by creating a cut-set at points of multi-fanout. Each of these sub-circuits is called a *subject graph* which is matched to the various library cells, called *pattern graphs* in the matching/covering step. The purpose of partitioning then, is to divide the size of the circuit to make the covering function more tractable. In its simplest form, the circuit is split at all points of multi-fanout and at the inputs to sequential elements. Each sub-partition is then covered locally. While this certainly makes the covering problem more tractable, the efficiency obtained by covering across partition boundaries

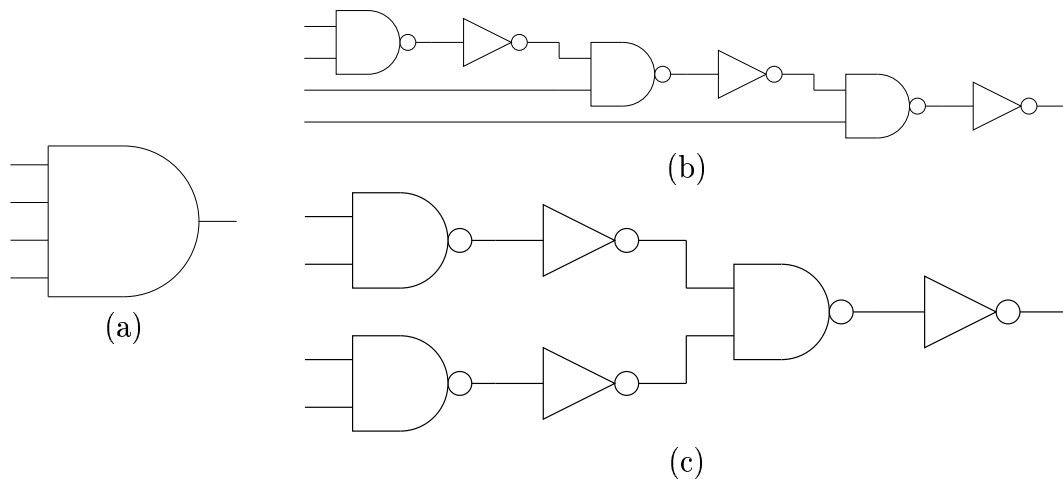


Figure 1.3. (a) 4-input AND gate. (b) A non-symmetric decomposition. (c) A symmetric decomposition.

is completely missed and the result is less optimum. Programs such as MIS [7] optimize the partitioning step to include the possibility of covering across partition boundaries. The final result of this optimization is likely to be a smaller circuit with a corresponding penalty paid in computation time.

The matching/covering step [33] finds all possible matches between the subject graph and pattern graph for each node in the sub-network. A set of these matches is then selected, as best conforms to the cost factors. Matching algorithms are classified as either *structural* or *boolean* and both methods are discussed in more detail below. Both methods fit into the design flow shown in Figure 1.1 and are differentiated primarily by the process which the subject graph, representing the synthesized design, is matched to the pattern graph, representing the available library elements.

Structural mapping is straight-forward and is the simplest to implement [24]. The decomposed functions can be viewed as trees, with roots at the outputs and the primary inputs as leaves. Beginning at the leaves, the decomposed subject graph is then compared node-by-node to the available library gates. Because of the semi-canonical nature of the decomposition, each node may have several variations

that need to be examined. If a library gate can implement the tree to that point, the node is annotated with that information in addition to the cost up to that point. The cost includes the cost of generating the inputs and the cost of the gate. For example, in Figure 1.4(b), the cost at node e would be the cost of a 2-input NAND gate where the cost of node g could either be the cost of an inverter in series with a 2-input NAND gate or the cost of a 2-input AND gate.

When the root is reached, the lowest cost for implementing that portion of the circuit is chosen. If there are higher level functions available in the library such as 3-input NAND gates or AND-OR-INVERT gates, it is typically advantageous to enclose as many gates in the subject graph as possible. The complexity of the matching algorithm increases as the base functions used increase in complexity. The additional search space needed for complete matching makes base functions beyond 2-input NAND gates and inverters computationally expensive.

The structural mapping method usually gives good results but the quality can be affected by how the decomposition is done and even by the form of the equations derived to describe the design. Because of this, structural mapping does not find all possible covers for a design.

Boolean mapping uses a process similar to structural matching except the nodes are labeled with the Boolean functions they represent rather than the cost. In [33], Mailhot proposes to use a list of possible matching functions at all nodes. This list represents all combinations of intermediate nodes available. For instance, given the 4-input AND function $f = (abcd)$ shown decomposed into 2-input NAND gates and inverters in Figure 1.4(b), the network consists of the following nodes: $f = \neg j$, $j = \neg(id)$, $i = \neg h$, $h = \neg(gc)$, $g = \neg e$, $e = \neg(ab)$. The library will be exhaustively checked for boolean functions matching each node. For instance, node h would be checked for gates implementing the following functions: $h = \neg(gc)$, $h = \neg(\neg e(c))$, $h = \neg(\neg(\neg(ab))(c))$. Unlike the structural method, these gates here need not match exactly; they need only be equivalent in a Boolean manner. For instance, in boolean matching, the gate $g = a(b + c)$ matches the function $f = (ab + ac)$. A match of this type would not be found by the structural mapper unless both gates

were structurally represented in the library. This boolean matching method can check for input permutations, inversions, and use *don't care* conditions to further optimize the solution. This method tends to be computationally prohibitive for larger networks but work by Burch [8] shows implicit methods can be used to produce more reasonable run times.

Looking again at Figures 1.4(b) and (c), we see two different structural decompositions of the 4-input AND gate shown in Figure 1.4(a). A boolean matching procedure at node f would find both of these decompositions equivalent. However, a structural mapper would find both of these a match to a 4-input AND gate only if their unique structural representations were entered as two separate library cells. Thus a library for a structural mapper must contain every possible structural representation of the function represented. Because of this, structural libraries tend to be limited to a smaller number of functions but the structural matching technique is extremely efficient computationally and the choice of which mapping technique to use depends as much on reasonable run times as it does on the quality of the final solution.

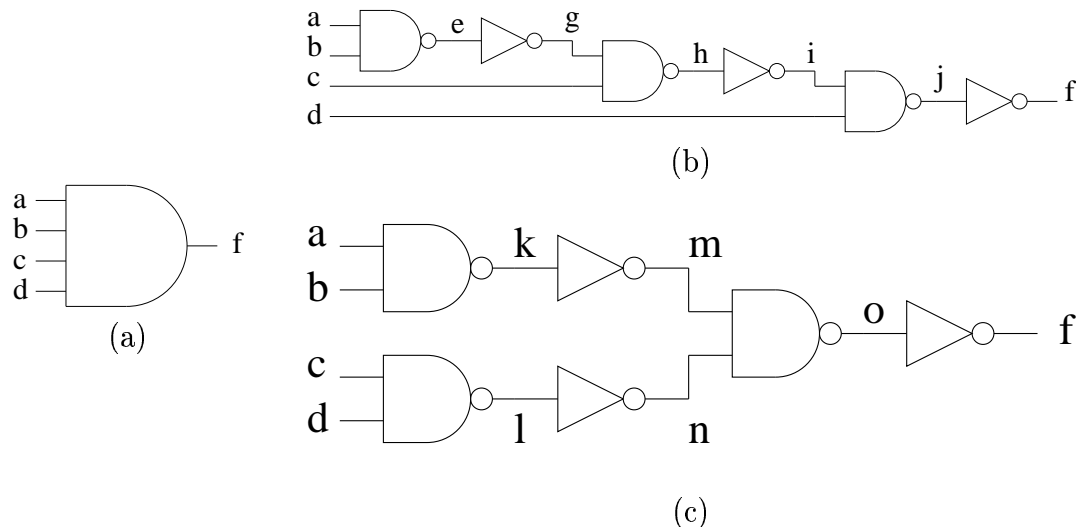


Figure 1.4. (a) 4-input AND gate. (b) A non-symmetric decomposition. (c) A symmetric decomposition.

1.5 Asynchronous Technology Mapping

The particular asynchronous design style chosen determines the types of hazards generated and the algorithms used to synthesize the design. Four main classes of asynchronous design styles avoid or reduce hazards during synthesis and technology mapping by simplifying the timing assumptions.

1.5.1 Delay Insensitive Circuits

Delay-insensitive circuits [47] make no assumptions on the delays of the logic or routing wires. This type of design is quite restrictive and few designs can be implemented by direct synthesis methods. Often, syntax-directed methods are employed which use non-standard libraries [18]. This approach renders technology mapping as unnecessary because the language compiler determines the structure and selects the library elements directly.

1.5.2 Speed Independent Circuits

Speed-Independent (SI) circuits [38] assume that the time skew of output branches is less than that of the logic gate delays. This timing assumption increases the potential for higher circuit performance by exploiting concurrency although not without its added design challenges. Siegel showed in [50] that the partitioning and matching/covering algorithms that she originally described in [51] as pertaining to burst-mode circuits (discussed in section 1.5.3) can be used without modification as applied to SI technology mapping. For decomposition, she stated that OR functions can be decomposed according to the associative law independent of signal ordering and the resulting network is still hazard-free. However, AND functions must be dealt with in a completely different manner. She suggests adding acknowledgment wire forks defined in [1] and proposed for use during decomposition in [3] for those gates where a hazard-free decomposition does not exist. These cases occur when sequential hazards are generated, which happens whenever a monotonic transition on an intermediate gate is not acknowledged somewhere else in the circuit.

Work addressing the decomposition of speed-independent designs was first addressed by Kimura in [25, 26]. Kimura's work involved analysis of circuit delays

when buffers were added to wires. In [29, 31, 30], Lavagno et al. leverages synchronous technology mapping methods by using the atomic gate assumption to produce hazard-free circuits but did so by introducing delay elements where needed to remove hazards. Myers et al. used a decomposition and re-synthesis approach applied to large-fanin generalized C-element gates [40]. A generalized C-element (gC) is a state holding component that also contains some random logic. Some of the most important work in this area is that done by Burns [9] where he addressed the decomposition of gC gates by using implicit methods to create and analyze large numbers of potential decompositions. Finally, Cortadella [12] and Kondratyev [28] more recently expanded the work of Burns to allow for a larger number of decompositions.

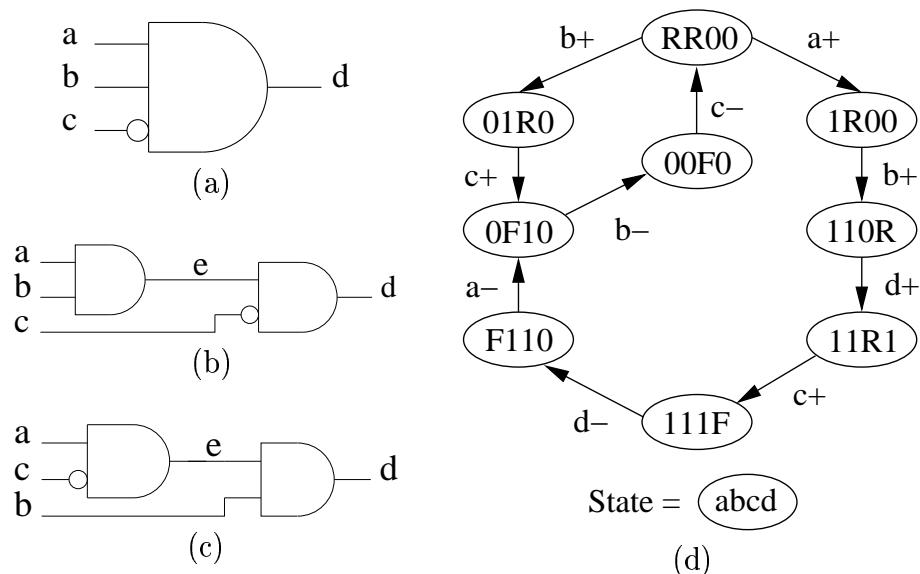


Figure 1.5. (a) 3-input AND function. (b) A circuit netlist that is hazardous under the speed-independent model. (c) A circuit netlist that is hazard-free under the speed-independent model. (d) State graph for 3-input AND function.

Figure 1.5 illustrates how the decomposition of an AND function may or may not be hazard-free under SI assumptions. The example shown is taken directly from [41] and utilizes a state graph (defined formally in chapter 2) shown in (d) to indicate a

sequence of transitions. The implementation for output d whose behavior is shown in the state graph, is synthesized using a CAD tool and the results are shown in Figure 1.5(a). Suppose this 3-input AND function is not available in the library. Decomposition must then be done in order for the circuit to be implemented. Figures 1.5(b) and 1.5(c) show two logically equivalent decompositions. The new internal signal e must now be checked for potential hazards introduced during decomposition. For the decomposition in 1.5(b), in state (F110), inputs a , b , c and internal signal e are high while the output d is low. After a falls, we move to state (0F10), and e becomes excited to go low. Assume the AND gate generating e is slow. When b falls, we move to state (00F0). If at this point c falls before e falls, d can become excited to rise prematurely. The result is a hazard on the signal d , and there is a potential for circuit failure.

Next, consider the decomposition shown in Figure 1.5(c), beginning in state (F110). This time e begins low, and it does not become excited to change until after a falls, b falls, c falls, and a rises again. At this point, b is already low which maintains d in its low state until b rises again. There is no sequence of transitions that can cause this circuit to experience a hazard on the output d .

This example illustrates the need to take special care during decomposition of SI circuits. It also hints at how explicit timing information may be used to help determine whether a hazard condition actually manifests or not. This topic is explored fully in Chapter 3.

For most of the work published in the SI domain, the implementation is done completely in the decomposition stage, making partitioning and matching/covering unnecessary. The goal of this work is to eliminate *all* hazards through a decomposition and re-synthesis loop.

1.5.3 Fundamental Mode Circuits

Fundamental mode circuits [57, 42, 59] further constrain the inputs by assuming the environment is too slow to respond to new input changes before the previous change has stabilized. This type of circuit acts much like a synchronous state machine where the timing on the input transitions is like the clock in a synchronous

system. This allows the designer to apply basic synchronous techniques. An expansion to the fundamental mode model, called burst-mode, allows a burst of inputs to change rather than a single input. A further extension, termed extended burst-mode (XBM), allows for the use of directed don't cares to specify that an input *may or may not occur* in a given burst. Decomposition can be performed by recursively applying De-Morgan's theorem and the associative boolean law to the network. Both operations have been shown to be hazard preserving [57, 51, 52], that is, if the original circuit was hazard-free, than the decomposed circuit is also hazard-free. More recently, technology mapping techniques for burst mode machines were developed in [10]. While these results were significant for the XBM type of design, they do not take into account possible optimization's due to known timing relationships. Figure 1.6 illustrates how the hazard problem encountered in Figure 1.5(b) for SI circuits cannot occur under fundamental mode assumptions.

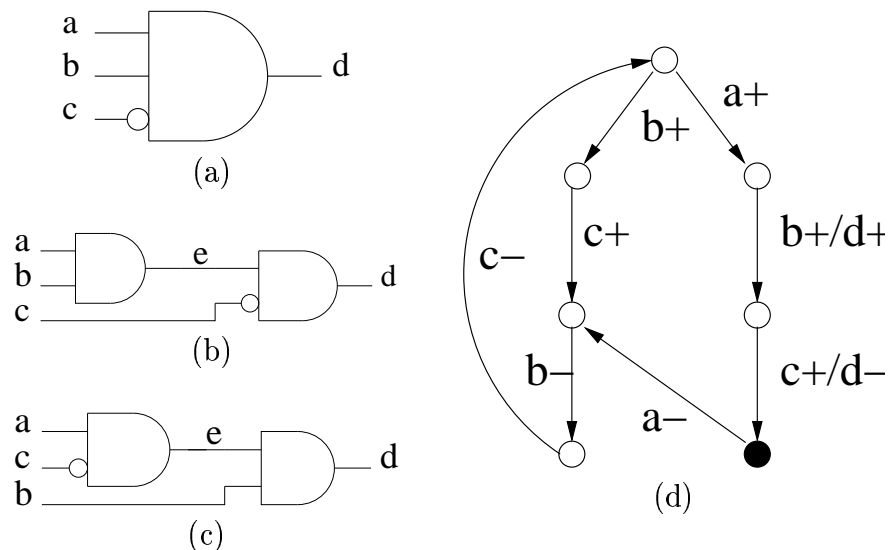


Figure 1.6. (a) 3-input AND function. (b) A circuit netlist that is hazardous under the speed-independent model. (c) A circuit netlist that is hazard-free under the speed-independent model. (d) Petri net for the 3-input AND function.

Figure 1.6(d) shows a burst-mode state machine represented as a Petri net

(defined formally in chapter 2) with inputs a , b , c and output d . The output d has again been synthesized with a CAD tool and the results shown in Figure 1.6(a). Starting with the dark circle in Figure 1.6(d), the circuit state is a , b , c , and e all high and d low. At this point, a goes low causing the new internal node e to go low. Then b goes low followed by c going low. The output cannot glitch because e is already low well before c goes low. In fact, there is no sequence of events that can cause a hazard, under fundamental mode assumptions, in either of the decompositions shown in Figures 1.6(b) and 1.6(c). This is a simple example of the hazard-preserving nature of fundamental mode decompositions.

This result should not be surprising; once any input or burst of inputs change, *all* internal signals and outputs must be allowed to evaluate before another burst of inputs can occur. The process for mapping fundamental mode circuits is virtually the same as that for mapping synchronous circuits with the restrictions that decomposition be done using only De-Morgan's theorem and associative laws, and the library elements used during the matching stage be hazard-free. Fundamental mode circuits suffer from the same limitation that synchronous circuits do; the reduced ability to exploit concurrency.

1.5.4 Timed Circuits

Timed circuits add a further restriction by placing two-sided timing constraints on the inputs and gates. There is little published work regarding technology mapping of timed circuits except for that done by Myers et al. in [40] where they used decomposition and re-synthesis to break up high fan-in gC gates.

Considerable work has been done on the synthesis stages of the timed circuit design process [39]. Since exhaustive state space methods quickly become intractable, much effort has gone into developing new algorithms that examine the state space and generate a reduced state graph (RSG) that factors in the initial timing constraints. The RSG is then analyzed to determine a set of valid solutions for the outputs. These solutions are further analyzed, and an optimum solution is chosen based on a set of constraints.

The output of the synthesis stage is a set of Boolean equations that consist

of combinational and sequential components. The sequential components contain state-holding elements and the combinational components contain standard logic expressions. These logic expressions have no fanin/fanout constraints and must be decomposed into smaller expressions that can be mapped directly to library cells.

It is often the case that hazard conditions found in SI circuits do not manifest as glitches in the real circuit implementation due to the actual timing behavior. The reason for this is that internal signals once enabled certainly do fire in some finite time. If we can track the time evolution in the state space, then it may be possible to identify the stability of all internal signals. Using this *timed cube approximation*, we can rapidly analyze a gate-level timed circuit for hazards.

The goal of our research is to apply the technology mapping process of Figure 1.2 to timed circuits. We plan to do this by extending the verification work done in [4] to perform efficient verification of our decomposed netlist. Verification will also be done on the final covered netlist to ensure that any timing changes incorporated during the covering process did not introduce new hazards. This new method that we present has the potential to greatly increase the size of circuits that can be technology mapped in an efficient and hazard-free manner.

1.6 Contributions

This dissertation makes five main contributions in the area of technology mapping of timed asynchronous circuits: applying the synchronous technology mapping flow to timed asynchronous circuits, utilizing explicit timing information to assist in the detection and elimination of hazards, driving the design flow to minimize rather than eliminate hazards, using the atomic gate model to encapsulate hazards during covering, and using standard libraries to implement hazard-free timed asynchronous circuits.

The first contribution is to utilize the synchronous approach to technology mapping. This allows us to leverage the existing research that is plentiful and mature and apply it to our design flow for timed asynchronous circuits. The synchronous methods of design decomposition, partitioning, matching, and covering

can work in our favor if we can address the issue of hazard elimination. Our approach requires us to use hazard-freedom as our primary cost metric.

The second contribution is the use of explicit timing information to aid the verification process. Not all hazard conditions result in actual glitches due to the finite time that it takes gates to evaluate. The identification and determination of which conditions *do* cause problems is a pivotal piece of the work we are attempting to accomplish.

The third contribution is the development of algorithms that minimize the number of hazards present in the netlist after decomposition rather than eliminate all hazards. Previous work has focused on the removal of *all* hazards prior to mapping the design to a library. Certainly it is easier to map a hazard free netlist to the library but this isn't always practical without numerous iterations or significant computational expense. Our algorithm allows for the existence of hazardous nodes after decomposition with the expectation that the matching/covering algorithm can eliminate the remainder.

The fourth contribution is the development of algorithms that identify the remaining hazardous nodes during the matching step and then eliminates them by encapsulating the hazardous nodes within library elements. Also, in conjunction with the timing algorithms, it may be possible to show that the new netlist timing from the covered circuit has altered the timing of the netlist such that nodes previously determined to be hazardous no longer exhibit hazardous properties.

The fifth and final contribution is the ability to utilize standard libraries to implement the timed circuit. Typical asynchronous approaches often require the use of specialized libraries to do a hazard-free implementation of the design. We are planning to push as far as we can the use of standard libraries to complete a hazard-free implementation. The quality of the final netlist, of course, depends on the available library components, but this is true for other types of asynchronous design as well as for synchronous design.

1.7 Dissertation Overview

Chapter 2 develops the necessary mathematical descriptions used throughout this thesis. Examples are used to help illustrate these descriptions. Formalism is developed to support the design description methods including time Petri nets, state graphs, and netlists.

Chapter 3 presents the bulk of the work focusing on methods to verify hazard freedom in a decomposed netlist. Since the decomposition can result in a large netlist, special attention is given to efficient verification techniques. Each node is checked for hazard behavior independently first using speed independent techniques then using explicit timing information. The netlist is then annotated with any nodes found to be hazardous.

Chapter 4 presents an efficient method for the decomposition of the netlist, particularly as it relates to the synchronous world. Since many of the existing algorithms for asynchronous technology mapping decompose directly to library gates, this chapter follows more of the synchronous approach, discussing enhancements applicable to asynchronous designs and in particular, timed methods. It finishes with a discussion of optimization techniques used to extract a more optimal solution throughout the entire mapping process.

Chapter 5 discusses the methods used to match the decomposed netlist to a general purpose library and then do a covering based upon hazard elimination. The netlist is annotated with information regarding any remaining hazardous properties. A final netlist is then written, containing only elements from the available library. This netlist is re-verified for hazard freedom and iterations are done as needed.

Chapter 6 tabulates and discusses the results from running numerous examples through the supporting software. Particular emphasis is given to the methods used to optimize the final netlists. The results are compared against other technology mapping tools. The cost of using hazard-freedom as the primary cost metric driving the final netlist creation is presented versus using the more traditional metrics of area and/or delay.

Chapter 7 summarizes the results of the work and discusses the successes and

limitations of this technology mapping approach. A future work section presents a number of ideas that we think could enhance or extend this work to produce more robust and/or correct circuits.