Efficient Synthesis of Speed-Independent Combinational Logic Circuits

W. B. Toms, D. A. Edwards
Dept. of Computer Science,
The University of Manchester
Manchester, M13 9PL, UK
{wtoms, doug}@cs.man.ac.uk

Abstract - Speed-Independent synthesis of combinational logic datapath circuits using tools such as Petrify is often inefficient or infeasible because such circuits typically contain many concurrent inputs and independent outputs. This paper presents a practical method for generating arbitrary combinational logic circuits, using a sub-class of speed-independent circuits known as Strongly-Indicating circuits, without the need to verify the speed-independence of the implementation through construction of a state-graph or other method.

I. Introduction

Speed-Independent circuits assume unbounded delays in gate operators and zero delays in wires. The elimination of global-variables and most delay assumptions within SI circuits makes them robust and amenable to reuse as their interfaces are free from timing constraints.

Speed-Independent circuits compare badly with more conventional circuits in both the transmission of data and in the implementation of logic functions. The focus of this paper is a technique for synthesising arbitrary speed-independent combinational-logic (CL) functions. Much research has been undertaken in synthesising SI circuits, resulting in several excellent synthesis tools, most notably Petrify [1]. The general-purpose nature of the algorithms employed in these tools means that implementing logic functions is difficult because of the large size and concurrency inherent in CL designs.

We describe a simplified substitution technique, that allows sets of signals to be inserted into CL circuits without compromising the speed-independence of the result. We then describe a synthesis method which preserves the validity of the substitution method, by partitioning the input set of CL circuits into independent blocks. We present efficient algorithms for the extraction of both single-cube and multi-cube divisor sets, and the results of synthesising several circuits is presented. The technique incorporates elements of both conventional multi-level logic synthesis and speed-independent synthesis. It is assumed the reader is familiar with the algebraic extraction techniques of multi-level logic synthesis and in particular the rectangle covering algorithms of Rudell [2]. The next section describes existing speed-independent synthesis techniques developed by Cortadella et al. [1].

II. Speed Independent Circuit Synthesis

A. State Graphs and Speed-Independence

A state-graph (SG) is a labelled directed graph, where each node represents a state of the circuit. The states are determined by the values of all the signals in the circuit and so, in a binary circuit, may be encoded by a n-length binary vector, where each digit in the vector corresponds to a signal in the circuit. The arcs between states in the graph are labelled with transitions of signals: either rising ($x+$) or falling ($x-$). A transition in either direction is represented by the notation $x*$. A state-graph is *consistent* if, for each signal, rising and falling transitions alternate. In any state a signal is excited, denoted x' if in that state a transition on x may occur. The maximal set of connected states in which signal x is excited is called an excitation region and is denoted by $ER(x*)$. The maximal set of connected states in which x is stable (i.e. not excited) is called a quiescent region, $QR(x*)$. A *continuous function region* of x ($CFR(x*)$) is the maximum set of connected states where x has the same value (i.e. $ER(x*) \cap QR(x*)$).

The speed independence of a circuit is determined by the properties of the SG that describes the behaviour of the circuit; the SG of an SI circuit must

- be deterministic: for each state there must be at most one outward bound transition for each signal
- be commutative: for each state, *concurrent* transitions – transitions that may take place in any order – must lead to the same state.
- satisfy persistency: once a signal a is excited, it must not be possible to enter a state in which the signal is stable except through the firing of a. In circuit synthesis, the persistency constraint is only enforced on the output and internal signals of the circuit; this constraint is known as output persistency
- satisfy Complete State Coding (CSC). Each pair of states in the state graph must have a unique binary code, unless the set of output transitions of the pair are the same.

B. Monotonous Cover

Monotonous Cover Conditions allow speed-independent circuits to be implemented in a canonical architecture using unbounded input AND gates, bounded input OR gates and C-elements. Their standard architecture is shown in figure 1. Each AND gate represents a single Monotonous Cover (MC) cube C_j, which covers a CFR and upholds the following conditions:

- Cover Condition - $C(a_j*)$ covers all states of $ER(a_j*)$ (i.e. evaluates to 1 in all states of $ER(a_j*)$).
- One-hot Condition - $C(a_j*)$ covers no states outside $ER_j(a+) \cup QR_j(a*)$.

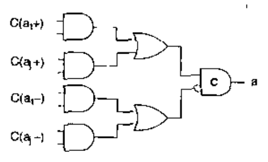

Figure 1: Standard Architecture for SI Circuits

- Monotonicity Condition - $C(a_j*)$ changes at most once along any state sequence of $QR(a_j*)$.

C. Event-Insertion

The process of inserting events into SGs is important as it allows CSC conflicts to be resolved, MC requirements to be fulfilled and circuit-implementations to be mapped into physical library cells.

Inserting an event, x, into a state graph results in each state in the excitation region of x being split into two states: one where x is excited and one where x has fired. The set of states that form the excitation region of x are called a *Speed-Independence Preserving* (SIP) set if the persistency and commutativity of the original state-graph is maintained. An excitation region of an inserted signal can be made into an SIP-set by including extra states in the region of any state diamond where a violation occurs. Adding states to an excitation region, changes the value of the function implementing the region, and so only states where the value of initial function is maintained may be added.

If a state graph is to be implemented as a binary-circuit, each event corresponds to a binary signal and hence complementary pairs of events relating to the rising and falling transitions of a signal must be inserted. These complementary events are inserted by means of an *I-partition.*

An I-partition consists of 4 sets of states, $\{S^0, S^1, S^+, S^-\}$, which relate to the transitions of the inserted signal. The states S^0 and S^1 are equivalent to the quiescent regions of signal x ($QR(x-)$ and $QR(x+)$ respectively), S^+ and S^- are equivalent to the excitation regions of x ($ER(x+)$ and $ER(x-)$ respectively). In order for the inserted signal to retain the consistency of the state-graph, the constructed I-partition must be *well-formed.* An I-partition is well-formed if there is no direct path from S^+ to S^0 and from S^- to S^1, hence the only permitted transitions are $S^0 \rightarrow S^+ \rightarrow S^1 \rightarrow S^-$.

D. Technology Mapping

Synthesised designs maybe technology-mapped in to existing libraries using event-insertion techniques. Valid algebraic decompositions of signals can be determined by checking whether the SI properties of the implementation are maintained by their insertion. The algorithm proceeds by selecting a divisor from the most complex cover in the network. An I-partition is constructed for the divisor and checked for well-formed SIP closure. Even if the divisor can be inserted into the network without violating the speed-independent properties of the circuit, progress conditions must be checked to ensure that the insertion reduces the cost of the implementation.

III. Speed-Independent Combinational Logic Circuits

A. Indication and Combinational-Logic

Varshavsky [5] defines a set of conditions *Self-Timed* logic blocks must uphold to be implemented and composed:

- Transitions must be *regular,* no nodes must change state more than once in a transition.
- Functions must be *isotonous* (or *antitonous*) i.e $f(x_i=1) > f(x_i=0)$ ($f(x_i=0) > f(x_i=1)$)
- Function inputs and outputs must be *unordered.*
- Each allowed input transition must be *translated* onto the outputs of a function by an output transition.

In order to eliminate the assumptions necessary in determining data-validity, both inputs and outputs in self-timed CL circuits are encoded in DI-codes [6]. DI-codes are *unordered* codes: no code word is contained within any other allowing for the unambiguous detection of valid data.

The most general class of codes are M-of-N codes, where M transitions on N wires represents the arrival of data. To increase the efficiency of such codes, large datapaths are formed by concatenating many small M-of-N code groups together.

Seitz [3] defined the operation of self-timed circuits in four-phases: inputs become valid; outputs become valid; inputs become invalid; outputs become invalid. This four-phase operation, also known as *Return-To-Zero Signalling,* is commonly implemented in asynchronous systems where all wires return to a quiescent state in between communication known as a *spacer.*

Seitz defined two possible modes of operation for self-timed CL circuits based on the order and overlapping of the four phases. In this paper we consider only Seitz's *Strong Conditions,* where the four phases are strictly sequential and no overlapping is permitted between any phase. Circuits adhering to the strong conditions are called *Strongly-Indicating Combinational Logic* (SI-CL) circuits. The other mode of operation defined by Seitz, *Weak Conditions,* allows restricted overlapping between phases and may lead to implementations with lower latency. Synthesis of *Weakly-Indicating* circuits is complex and is not considered in this paper.

B. Speed-Independent Combinational-Logic Circuits

Combinational logic circuits which deal with data arriving in some delay-insensitive code form an interesting sub-set of SI circuits. They are characterised by:

- many concurrent inputs
- many independent outputs which are often concurrent
- excitation and quiescent regions of the outputs that are often overlapped, although none is a proper subset of another.
- SGs that are distributive and have CSC.

Because of these characteristics, CL circuits can easily be implemented speed-independently using the SI standard architecture. However, because of the large numbers of inputs, these designs often need to be decomposed so that they may be mapped into gate-libraries. Robust techniques, such as those described in the previous section for decomposing general-purpose speed-independent circuits, have problems due to the large numbers of common inputs between signals. Divisors are selected by looking at signals individually and calculating the cost on the rest of the network of inserting a suitable signal in its current state. However progress conditions give no indication of how complicated the inserted signal will make decomposing other signals in the network. For this reason it is preferable to use simpler techniques, targeted specifically at SI-CL circuits, that select divisors by looking at all the network as a whole so that the full cost of the divisor can be evaluated. We present a method for synthesising SI-CL circuits based on traditional multi-level logic synthesis techniques.

C. Inserting Signals in SI-CL State-Graph

We will show that, because of the nature of strongly-indicating CL circuits, inserting a signal x with function f where:

$$f = f(sup(x))$$

$$\bar{f} = neg(x) = \wedge_{s \in sup(x)} \bar{s}$$

preserves the speed-independence of the implementation, providing that the signal x is inserted in the cover of every single function in which it is resident. *sup(x)* is the *support* of x and represents the set of input-literals of f. If f is a multi-cube function, each of the cubes of f must have exactly the same quotients in every function they are resident, to preserve the functional equivalence of the resulting circuit. For single cube functions this is automatically true and therefore any single-cube function can be inserted in this manner.

In a strongly-indicating CL function, each input transition is translated to an output transition. Therefore, if the function of x is substituted in every function in which it is resident, for each input set of the whole network that fulfils the cover of x $(C(x))$, at least one of the functions, a, into which x is substituted, must transition and the circuit enters $QR(x+)$. The strict ordering of phases within strongly-indicating systems means each phase must complete before the next phase can begin and the inputs in *sup(x)*, cannot become invalid until the output a has transitioned. It is not possible for the I-partition constructing x to transition from $S^{+} \rightarrow S^{0}$. The same is also true of the transition from $S^{-} \rightarrow S^{1}$ as all outputs must become invalid, before an input may become valid. Therefore the I-partiton constructed to insert signal x is wellformed.

An SIP set can always be generated from the input-border of a signal inserted in a SI-CL using the method described. As it is guaranteed that at least one signal, a, will transition as a result of x, all other input transitions where $f = 1$, are independent of x, i.e. they are inputs not in *sup(x)*, because all the inputs of x will have transitioned and may not do so again until all outputs of the function are valid. Therefore any states connected to $ER(x+)$ by an input-transition or an unrelated output transition may be added to $ER(x+)$, as $f = 1$, until $neg(x) = 1$, which is after a has transitioned and the circuit is in $QR(x+)$.

It can be shown [4] that both the cover functions that signal x is inserted into and the cover of x itself maintain the MC conditions necessary for speed-independent implementation, and hence can be implemented in the SI standard architecture.

Before being decomposed, a SI-CL circuit contains no internal signals, all the outputs signals are independent of each other. Therefore the cover function of each output signal can be described purely in terms of the input-signals of the network, and combinational logic techniques, such as the matrices of [2] can be used to determine suitable divisors. Substituting signals into function covers obscures the dependence of the functions on the original inputs. This means that matrices cannot be guaranteed to detect all instances of a cube within the functions of the network.

Simultaneously substituting a set of divisors (D) that completely cover a set of inputs (I) removes the input set completely from all other functions in the network and ensures that no dependencies can occur between the remaining functions of the network involving the input set I. The signals of D are also independent and hence CL synthesis methods can be used to select further sets of divisors. This means SI-CL circuits can be synthesised without the need to construct complex state-graphs for circuits, allowing a much larger range of SI-CL circuits to be synthesised.

IV. SI Combinational Logic Synthesis

In order to simplify the substitution process and prevent the need for state-graph analysis of the dependencies between signals, we would like to generate a set of divisors, D, where:

$$sup(D) = \bigcup_{d_i \in D} (sup(d_i))$$

which encapsulates the set of signals, *sup(D)*, in such a way that they are either removed from the cover functions of other signals of network, A, entirely, or remain only in terms with no common cubes containing literals of *sup(D)*. The selection of divisors is based on selecting common-cubes from functions. Therefore no divisor can be selected across the two sets of inputs caused by the partition. This is possible because SI-CL functions generally consist of several different code-groups, the terms of which are mutually exclusive and can be used for factoring logic in the same way as are binary variables in conventional logic synthesis.

Co-kernel-cube-matrices are the perfect tools for partitioning the inputs of A into two parts. The two-dimensional structures naturally represent a partition between common cubes and their remainders.

Property 1 *In a strongly-indicating CL network a complete sets of divisors can be generated from an intersecting rectangle of a Co-kernel-cube Matrix.*

To generate a complete set of divisors, all commonality in the input set must be explored. The entrants of a row (or column) of a matrix represent terms with common cubes. We only

Co-Kernels	b3 b1	b2 b1	b3 b0	b2 b0	a3 a2	b5 b1	b4 b1	b5 b0	b4 b0	a5 a3	a4 a3	a5 a2	a4 a2	a5 a4
a2a0	16		31			14	15	29	30					
a3a0		16		31		12	13	27	28					
a4a1	7	9	22	24		5		20						
a5a1	6	8	21	23			5		20					
b1b0					16					12	13	14	15	11
b2b0					31					27	28	29	30	26
b2b1					10					6	7	8	9	5

Figure 2: Intersecting Rectangles

consider Co-kernel-cube matrices with algebraic entries (all entries have disjoint co-kernel and kernel-cubes) and so two rows (columns) that share a common column (row) represent cubes of the same input set. If the input set of the rows and that of the columns is mutually-exclusive then the intersecting rectangle will form a perfect cube, where each row (column) contains the same set of column (row) entries. However it is useful, particularly for code groups where each code word cannot be represented by a single implementable cube, to use partitions where the row and column sets are not disjoint, such partitions form *dispersed* rectangles. In such rectangles, the input sets represent terms that overlap and several cubes must be inserted simultaneously into function terms to provide complete cover of the input set. Figure 2 shows the two different types of intersecting rectangle. A simple algorithm is shown in figure 3 for detecting intersecting rectangles from a matrix, based on the sparse matrix rectangle implementation of [2]. The process of synthesising a SI-CL network is described in the next two sections.

```
gen_intersecting_rectangles(M, index, rect) {
   new-rect = rect;
   col = M->columns[index];
   if col not in new-rect {add col to new_rect;}
   for-each element p in column {
      r = M->rows[p->rownum]
      if r not in new-rect {add r to new_rect;}
      for-each element p in r {
         i = p->colnum;
         if i >index {
            gen_intersecting_rectangles(M,i,new-rect);}
   }}
   return new-rect; }
```

Figure 3: Algorithm for Detecting Intersecting rectangles

A. SI Single-Cube Extraction

Single-Cube extraction is the simpler of the two stages. Single-cube extraction is concerned with extracting common cubes from the function terms of the whole network.

A Co-kernel-cube matrix is created from the set of unique cubes in the whole network. The algorithm of figure 3 is used to determine the sets of common-cube divisors and the co-kernel groups of the intersecting rectangles. Rectangles with disjoint support can be substituted independently of each other. To select between rectangles with overlapping support, a cost function is used which determines the cost, in literals, of adding dispersed rectangles to the network. The cost function reflects the fact that in dispersed rectangles, several co-kernels will be inserted into the same term. Therefore when dispersed rectangles are selected, the cost of implementing the network may increase. This is necessary as, in order to implement SI-CL functions independently, all common terms must be extracted from the network: this allows any large cubes to be decomposed into technology-specific gates arbitrarily.

Once the set of intersecting rectangles is selected, they are substituted simultaneously into all cover functions of the network. The process is then repeated, by constructing a new co-kernel-cube matrix for the new function terms of the network, until there are no multi-literal common cubes shared between any terms of the network. If no intersecting rectangles exist in the matrix but there are still shared cubes between terms, the network cannot be further partitioned and all remaining multi-literal co-kernels must be inserted simultaneously.

B. SI Multi-Cube Extraction

Multi-Cube extraction is more complex than single-cube extraction for two reasons. Firstly, the set of multi-cube divisors is constrained by the substitution process and so unsuitable divisors must be filtered. Secondly, unlike single-cube extraction, the co-kernel-cube-literal matrix for multi-cube extraction is generated with the kernels and co-kernels of each function; therefore, function terms that only appear once in a function may not be represented in the matrix, and hence the matrix cannot be used as a definitive guide to selecting correct divisors.

Initially candidate sets of divisors are generated from the matrix using the intersecting rectangle algorithm of figure 3. From these intersecting rectangles only a small proportion contain suitable kernel-intersections. The substitution process of C. only permits the use of multi-cube divisors that have the same quotient for each position they appear in the network. These divisors can be determined from the matrix as *maximal kernel-intersections*, columns with identical row entries. Intersecting rectangles that do not contain any maximum intersections cannot be substituted and are therefore discarded.

Any kernel-cubes in an intersecting rectangle that do not form maximum intersections can not be substituted and form the *co-rectangle* of the candidate divisors. In order to guarantee that removing a set of multi-cube divisors partitions the network correctly, we ensure that all intersecting column-terms of the co-rectangle are disjoint. If intersecting column-terms are not disjoint then there exists, for each co-kernel, a, of the intersecting rows, a common quotient, q, of the terms containing a and any common literals shared by the terms. This invalidates the principle of disjoint input sets as q is a common cube containing shared literals from both sets.

In order to ensure that only divisor sets that will maintain speed-independence are inserted, the position of the divisors in the whole network must be determined. As each entry of the kernel-cube matrix is labelled with the index of the network term it forms, the set of terms covered by an intersecting rectangle is easily determined. If an intersecting rectangle does not cover all of the terms of the network, it is possible that the rules of substitution maybe violated and so the relationship of the divisors to the uncovered terms must be evaluated. As the cubes of multi-cube divisors must share the same quotient for all network terms, any missing terms must form part of the co-rectangle of candidate divisors. If no term of the co-rectangle is contained in a missing term, the term forms a new column of the co-rectangle and hence must be disjoint with the other co-rectangle column terms.

Once the valid rectangles have been determined all non-overlapping, i.e. disjoint or contained, rectangles can be inserted into the network. This is done sequentially starting

```
gen_multicube_divisors(R, function-terms) {
  divisors = {}; co-divisors = {};
  co-kernels = R->rows; R1 = R;
  divisors = column terms with identical rows
  co-divisors = column terms with non-identical rows
  if empty divisors return false;
  else {
    covered-terms = 0;
    for-each element p in R {
      add p->term to covered-terms }
    remaining-terms =
      differ(function-terms,covered-terms);
    for-each t in remaining-terms {
      if (t contained in divisors) return false;
      else if (t not contained in co-divisors)
        for-each co-kernel c contained in t
          for-each co-divisor d {
          if not-disjoint(d,(t / c)) return false; }
  return R; }
```

Figure 4: Algorithm for determining multi-cube divisors

with the largest value rectangle. The process is then repeated until no multi-cube divisors remain. The algorithm for determing whether a intersecting rectangle forms a valid divisor group is shown in figure 4.

The method described is an efficient synthesis procedure for a large range of SI-CL circuits up to several thousand minterms. The method is vastly more efficient compared to general-purpose speed-independent synthesis and decomposition techniques. A large class of practical speed-independent circuits that were previously unsynthesisable can now be synthesised.

V. Results

Table 1 shows the results of the synthesis procedure on a small selection of SI-CL circuits. The two categories of circuits are representative of circuits used in combinational logic speed-independent designs. Each adder circuit is a full-adder constructed from two single groups of the code, with a dual-rail (1-of-2) carry. The completion detection functions are used to detect the arrival of valid code words within a single code group and are used extensively in speed-independent implementations.

The table details the number of minterms of the functions, and the cost in transistors of the implementation. Each circuit is implemented in two input gates, any large cubes being decomposed. 2-input OR and AND gates have 6 transistors and a 2-input C-element 10 transistors. We also show the cost of

Circuit Type	Circuit Code	Minterms	SI Standard Architecture	New Technique
Full Adder	Dual-Rail	8	376	154
	1-of-4	32	1364	450
	2-of-4	72	4116	1370
	3-of-6	800	128112	9094
	4-of-7	2450	529326	26486
Completion Detection	2-of-4	6	110	90
	2-of-7	21	350	330
	3-of-6	20	414	664
	4-of-7	35	854	2164
	5-of-7	21	602	4320

Table 1: Comparison of Synthesis Results

implementing the circuits in the SI standard architecture using unlimited fan-in gates. In this case, the costs were deduced from size of each cube in the design. We assume that each extra input to a gate results in two extra transistors, one on each transistor stack.

Completion detection circuits are particularly difficult to synthesise using this method. Completion Detection functions consist of a single code-group, and so, in general, the input sets can not be partitioned further. Where the number of active inputs is large, the size of implementations increases rapidly as each rectangle is dispersed and the cost of the network increases with each iteration. All strongly-indicating implementations will suffer the same problems and so it is necessary to implement such circuits using weakly-indicating methods: an area for further research.

VI. Conclusions

We have presented an efficient technique for the synthesis of strongly-indicating combinational logic circuits. The technique, based on conventional circuit synthesis methods allows a range of SI-CL circuits to be synthesised. Such circuits were previously unsynthesisable owing to the complexity of general purpose speed-independent synthesis methods which require the construction of infeasibly large state-graphs.

VII. References.

[1] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev. "Logic Synthesis for Asynchronous Controllers and Interfaces", *Springer-Verlag*, 2002.

[2] R. L. Rudell. "Logic Synthesis for VLSI Design", *PhD thesis*, University of California at Berkeley, 1989.

[3] C. Seitz. "System Timing", *C.A. Mead and L.A. Conway (eds), Introduction to VLSI systems*, Addison-Wesley, 1980.

[4] W B Toms "Synthesis of QDI Datapaths", *PhD thesis*, University of Manchester, 2004.

[5] V.I. Varshavsky, ed. "Self-Timed Control of Concurrent Processes: The Design of Aperiodic Logical Circuits in Computers and Discrete Systems", *Kluwer Academic Publishers*. 1990.

[6] T. Verhoeff, "Delay Insensitive Codes - an Overview", *Distributed Computing Vol. 3*. 1988.