QDI Implementations of Boolean Graphs

W.B. Toms
Dept. of Computer Science, The University of Manchester,
Oxford Road, Manchester M13 9PL, UK.
tomsw@cs.man.ac.uk

Abstract

Implementing combinational logic efficiently in QDI designs is hard, due to the need to acknowledge every transition, and naive approaches are often used. The paper demonstrates the problem of using Delay-Insensitive Minterm Synthesis on anything but the smallest designs.These problems can be overcome by employing techniques from Multi-Level Logic synthesis. An initial method using Roth-Karp decomposition is described.

1. Delay-Insensitive Circuits

Delay-insensitive circuits make no assumptions about the delays within either the circuit or its environment, except that they are finite and positive. As such they are more robust than other circuit styles whose operation is based on (and optimised for) worst case constraints. Apart from this increased operating ability, delay insensitivity has many other advantages:

- DI circuits need no timing validation once they are designed. Circuit styles such as single-rail can only be validated by comprehensive simulation.
- DI circuits can be scaled to any process easily and are not constrained by layout timing issues.

These factors make delay insensitivity particularly desirable in a synthesis environment as it allows designers to produce designs that need little post-layout verification, can be ported between technologies with ease and surrendered to automatic place and route software without any danger of it breaking delay assumptions.

However, eliminating the assumptions made in a design also has its costs. Delay-insensitive circuits:

- generally face extra overheads in area due to the need to disseminate timing information throughout the circuit.
- usually have higher power consumption due to the need to transmit data validity explicitly
- may suffer a speed penalty due to the extra logic and switching involved in their operation.

In order for a circuit to be DI certain conditions must be upheld [4]:

- *Stability* - Once the conditions that allow a gate to transition (*guards*) are met, they cannot be falsified before the gate has transitioned
- *Non-Interference* - The two guards of any gate must be mutually-exclusive.

In order to uphold these conditions Martin defined two properties all DI circuits must adhere to:

- *Acknowledgement Theorem* - Each non-final transition in a circuit must be *acknowledged* by a subsequent transition.
- *Unique Successor Set Theorem* - The set of nodes that transition as a result of a transition on a node, *x*, must be unique and the same for both up-going and down-going transitions on that node.

In practice these properties are so restrictive that no practical DI circuits can be built. So Martin suggested the weakest compromise to delay-insensitivity, the *isochronic fork*. An isochronic fork is a fork where only one transition need be explicitly acknowledged, with the assumption that if a transition is seen on one end of the fork, it will also have appeared on the other end. Circuits implementing isochronic forks are called Quasi-Delay-Insensitive (QDI) circuits. Research [11] suggests that isochronic fork assumptions need careful verification and can be violated by mismatched threshold voltages or gate capacitances on the receiving gates.

Isochronic forks can be extended through gates, resulting in Q^nDI circuits, where *n* is the number of gates through which an unacknowledged signal travels. Q^nDI circuits often employ asymmetric forks, where a path a deemed to be slower as it travels through less gates. Again [11] showed that these assumptions need careful verification as their operation is not always predictable.

Theseus Logic Inc. [3] extended the notion of isochronic forks and extended isochronic forks, and traded the unacknowledged path against the cycle time of the circuit. These assumptions are known as Orphans.

Data in QDI systems is encoded in a delay-insensitive, or unordered, code[12]. This encodes validity directly into the data, eliminating the assumptions necessary in transmitting a *valid* signal separately. There are many DI codes, the simplest are the one-hot codes, where a single transition on a wire represents the arrival of valid data, with a set of *n* wires representing *n* values.The efficiency of such a code can be increased by

concatenating many such codes together into a larger code called a 1-of-*n*. This idea can be generalised to an *m*-of-*n* code where transitions upon *m* wires from a set of *n* represents the arrival of valid data. Often the *Rate* (number of data symbols per wire) of a datapath can be increased by employing *m*-of-*n* codes (m>1), but the logic is generally more complicated due to the encoding complexity.

2. Combinational Logic in QDI Systems

Implementing QDI logic is generally more complicated than in other design styles as traditional optimisation techniques cannot, in general, be used, as these techniques make use of don't care values to try and reduce the size of implementations. In a QDI circuit, if a transition has no part to play in the output of a function, it becomes unacknowledged and therefore introduces extra assumptions into the circuit.

2.1. Delay-Insensitive Minterm Synthesis

A very simple method for performing logic in QDI circuits is called *Delay-Insensitive Minterm Synthesis* (DIMS)[10]. Here the function is implemented by a simple S-O-P of all the valid products. It should be noted that in Return-To-Zero DI systems, the spacer (zero) value of a wire represents an absence of data on that wire, so all products are generated with only the active literals in that value, and there are no inverters. To correctly acknowledge the return-to-zero phase of the circuit, each product is instantiated with a c-element. Because each product is unique only one c-element will be activated in any cycle and will be properly acknowledged by the or-gate network. The circuit is QDI, the only assumption being the fan-out of the inputs to multiple gates. This method, because of its simplicity, is very easy to use for the synthesis of general function and has become popular[]. It's main drawback is its size, which increases exponentially with function size.

However, the DIMS technique can become invalid for QDI networks employing complex codes or a number of input code groups. In a DIMS function each product is unique as it is implemented by a single gate element. However, for anything but the simplest functions, fan-in restrictions mean the products have to be decomposed over several gates (fig1). When the products are decomposed, the products cease to be unique as the partial products may be shared amongst several products as in fig 1. This means when an input set is applied to the circuit several intermediate gates will be activated, but only one product will actually fire, leaving several unacknowledged transitions on the gates of the unactivated products, resulting in Q^nDI circuits. Clearly this problem increases with the use of *m*-of-*n* codes and increased inputs, causing further decompositions, and possibly shared trees of c-elements. This results in dangerous Orphan assumptions, particularly in synthesis systems where the amount of sharing is not known, and may go unchecked.

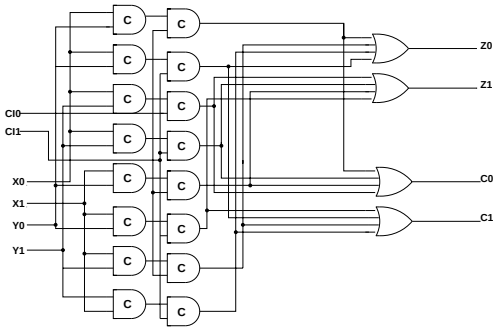

Figure 1: Decomposed DIMS Dual-Rail Full Adder

The problem can be solved by sharing common partial products between all the products. This problem is similar to extracting common divisors in multi-level logic synthesis, with the extra constraint that all common products must be shared. A simple example of how traditional synthesis techniques may be used to implement QDI circuits is given below.

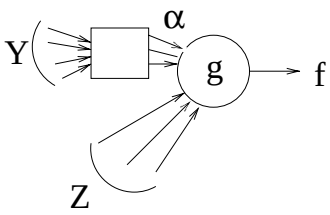

Figure 2: General Function Decomposition

3. Functional Decomposition

Functional decomposition was first suggested by R.L. Ashenhurst in the 1950's [1], and can be thought of as transforming a function, *f(X)*, to the form

$$g(\alpha(Y), Z)$$

(fig) where X is as set of variables $\{X_1,X_2,...,X_n\}$ and Y and Z are subsets of X. Y is known as the *bound* set of X and *Z* the *free* set. The function *g* is known as the image. The idea behind the decomposition is to calculate a set of compatible classes from the input vectors of *Y,* which are encoded in the outputs of the function α.

Ashenhurst describes a simple, disjoint decomposition where *Y* and *Z* are disjoint and the α function only yields a single binary value. Ashenhurst's method involved generating a *partition matrix*, which contains an entry for all possible values of the *Y* and *Z* sets. The matrix is reduced to remove all duplicate rows and columns. Ashenhurst stated that for a simple decomposition to exist a partition matrix must have a most two distinct columns.

The notion of decomposition was extended by J. P. Roth &

R. M. Karp[6] to include non-disjoint complex decomposition, involving multiple α functions.

$$f(X) = g(\alpha_1(Y), \alpha_2(Y), \ldots, \alpha_n(Y), Z)$$

The method described by Roth & Karp is preferred as it avoids creating a large partition matrix.

In more recent times, decomposition has been used to implement functions on Field Programmable Gate Arrays (FPGA). FPGA's are built up from small lookup tables (LUT) that can implement any function of *m* inputs (usually around 5). Roth-Karp decomposition was suggested as a way of optimising functions for FPGAs by decomposing them into an *m-feasible* network i.e. a network where each node has a *support* of *m* and can be implemented by a single LUT.

Murgai et al. [5] suggest a method using recursive decomposition of the image, *g*, to achieve an *m*-feasible network, where the logic of the image function is minimised by selecting a suitable encoding for each of the compatibility classes using existing input/output state encoding techniques. Murgai simplified the problem by allowing the values in a compatibility class to be encoded to different values, if necessary, and, because the procedure was targeting FPGAs (which can implement any function of up to *m* variables in a single LUT), ignoring the complexity of the functions generating the α variables.

4. QDI Decomposition

In essence, the problem of generating QDI networks can be thought of as being similar to that of implementing efficient FPGA *m*-feasible networks. In a QDI network each node is implemented by of up to *m* inputs, where *m* is the maximum fan-in of the gate. For the network to correctly acknowledge each transition all common nodes must be shared throughout the network.

It is hypothesised that a QDI network can be built using Roth-Karp decomposition by applying some constraints to the α-functions and the encoding of compatible classes.

A lot of previous research has been done on the decomposition of Speed-Independent networks (for library binding). The technique described here differs slightly from previous work by only considering Return-To-Zero combinational networks, the transitions involved are always monotonic and unate, so can be thought of as a simplification of previous work. The decomposition differs from [9] in looking at multiple outputs and trying to perform common decompositions among them rather than trying to decompose single gates independently. It is believed that the decomposition is similar to the special cases of [2]: *unate second operator, always acknowledging second operator* and clearly the *generalised c-element decomposition*. It remains to be determined whether this technique fulfils the tautologies described and how Montage would fair on a QDI combinational network, as it too appears to decompose single nodes at a time.

5. Assigning α-Functions

During decomposition the legal input vectors of a set of input variables (*Y*) are assigned to compatibility classes, two values y_1 and y_2 are compatible if for all (legal) values of the free set, *Z*, $f(y_1) = f(y_2)$. Once all the compatibility classes been determined they must be encoded into a set of boolean variables, α, which form inputs to the image function. How the α-variables are implemented determines whether or not the network will be QDI.

As all QDI combinational networks, apart from completion detectors, have multiple outputs; each decomposition stage consists of calculating compatibility classes for all outputs and encoding all of these in the α-variables. As the set of values each class represents may not be disjoint and maybe ordered (i.e. some classes may be contained within others) certain constraints are necessary to ensure no unacknowledged nodes are created in the network.

Firstly all values for all classes have to be represented in full, don't care values among classes cannot be exploited. The values are represented by a set of δ-variables. Each δ-variable represents a product of the initial input vectors of up to *m* literals. i.e. the literal 0110010 from a set of variables, $x_1 \ldots x_6$, could be represented by the δ-variables δ_1, δ_2 where δ_1 represents the literal x_2x_3 and δ_2 represents x_5.

A set, Δ, of δ-variables may be combined, by a c-element tree, if no element of Δ occurs in any value where the whole of Δ is not present. The values, now represented by δ-variables, are assigned to compatibility classes. Values may be reduced to a single encoding, by an or-tree, only if they are single δ-variables, i.e. they do not component nodes with any other value, and as before the set is instantiated in every class in its entirety.

Building up α-variables in this way ensures that all the logic will be implemented in a QDI manner, where common components shared between compatibility classes will be visible in the α-variables. If the decomposition is continued until the image function is realisable by an *m*-feasible network, all common products will be combined and the resultant network will be QDI.

The decomposition recursion terminates when the maximum number of literals in the products of the image are equal or greater to the number before the decomposition took place. If the image is not immediately *m*-feasible when recursion terminates, the products can be implemented from the α-variables and the free-set by decomposing them into δ-variables and reducing as before. Here, however, the products are unique and so a QDI realisation can be achieved.

6. Example

Fig shows a 1-of-4/Dual-Rail full-adder. This has a single 1-of-4 input (a_0, a_1, a_2, a_3)and two Dual-Rail data inputs($b0_0$, $b0_0$ and $b1_0$, $b1_1$) as well as a Dual-Rail carry input (ci_0,ci_1). The DIMS implementation requires 32 4-input c-elements and 12 4-input and 6 2-input or-gates. This can be implemented in

4 levels of logic by 16 3-input and 32 2-input c-elements and 18 3-input and 6 2-input or-gates using only gates of up to 3 inputs. The decomposed implementation uses, 32 2-input c-elements and 18 3-input and 21 2-input or-gates, and has between 4 and 6 logic levels.

The design was initially partitioned:

$$Y = \{a_0, a_1, a_2, a_3, b0_0, b1_1\}$$

$$Z = \{b0_0, b1_1, ci_0, ci_1\}$$

The compatibility classes for this partition where:
Z_o,Z_1 ={(001001, 010001, 001010, 010010),
(000101, 100001, 000110, 100010)}
Z_2,Z_3 ={(001010, 010010),(000110, 100001),
(001001, 010001), (000101, 100010)}
Z_4,Z_5 ={(001010, 010010),(000110, 100001),
(001001, 010001), (000101), (100010)}
Resulting in 5 alpha variables:
Z_o,Z_1 ={(10000,01000),
(00001, 00010, 00100)}
Z_2,Z_3 ={(01000),(00100),
(10000), (00010, 00001)}
Z_4,Z_5 ={(01000),(00100),
(10000), (00010), (00001)}
Next the design was initially partitioned:

$$Y = \{b0_0, b1_1, ci_0, ci_1\}$$

$$Z = \{\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4\}$$

With the compatibility classes:
Z_o,Z_1 ={(1001, 0101),(1010, 0110)}
Z_2,Z_3 ={(1001),(0101),(1010), (0110)}
Z_4,Z_5 ={(1001),(0101),(1010), (0110)}
Resulting in 5 alpha variables:
Z_o,Z_1 ={(1000, 0100), (0010, 0001)}
Z_2,Z_3 ={(1000),(0100),(0010), (0001)}
Z_4,Z_5 ={(1000),(0100),(0010), (0001)}

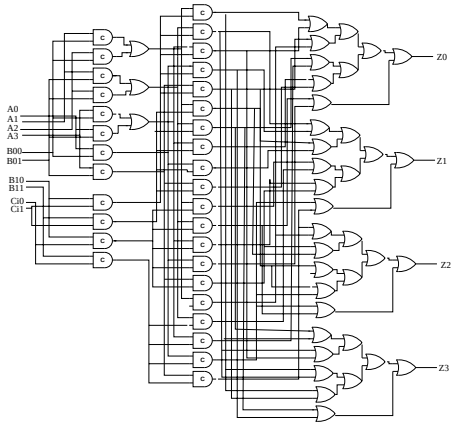

Figure 3: Decomposed 1-of-4/Dual-Rail Full Adder (Sum Function Only)

A final partition was attempted but this lead to an increase in the number of classes and literal count of the image products. The image products were formed by 20 δ-variables and the or-tree calculated for each output separately.

7. Conclusion

An initial method was presented for performing technology-dependent decomposition in order to implement a QDI circuit in a cell-library. This approach presents many problems, boolean decomposition has a high computational complexity and is restrictive for large circuits, calculating the α-variables is complex and library dependent. A better method based on algebraic decomposition and extraction, that is technology independent has been postulated and will be presented.

8. References

[1] Ashenhurst, R.L., The Decomposition of Switching Functions. *Proc. Int. Symp. on the Theory of Switching.* 1959

[2] Burns, S. General Conditions for the Decomposition of State Holding Elements. *Proc 2nd Int. Symp. Asynchronous Circuits and Systems* 1996.

[3] Fant K.M., Stephani R., Smith R., Jorgenson R. The Orphan in 2 Value NULL Convention Logic. Tech. Rep. Theseus Logic Inc. 140, 485 N. Keller Rd. Maitland, FL 32751

[4] Martin, A.J, The Limitations to Delay-Insensitivity in Asynchronous Circuits. *6th MIT Conference on Advanced Research in VLSI Processes*, 1990.

[5] Murgai, R., Brayton, R. K., Sangiovanni-Vicentelli, A. Optimum Functional Decomposition Using Encoding *Proc 31st ACM/IEEE Design Automation Conference*, June 1994.

[6] Roth, J.P., Karp, R. M., Minimisation Over Boolean Graphs, *IBM J. Res. Develop*, 1962.

[7] Plana, L.A, Riocreux, P.A, Bainbridge, W.J, Bardsley, A, Garside, J.D, Temple, S. "Spa - A Synthesisable Amulet Core for Smartcard Applications" *Proc 8th Int. Symp. Asynchronous Circuits and Systems* 2002.

[8] Saldanha, A., Villa, T., Brayton, R. K., Sangiovanni-Vicentelli, A. Satisfaction of Input and Output Encoding Constraints. *IEEE Trans. Computer-Aided Design Vol. 13 No. 5.* 1994.

[9] Siegel, P. Micheli, G. D. Decomposition Methods For Library Binding of Speed-Independent Asynchronous Designs. *Proc. IEEE. Int. Conf. Computer-Aided Design.* 1994.

[10] Sparsø, J.Staunstrup. J., Delay Insensitive Multi Ring Structures. *Integration, the VLSI Journal*. Vol. 15. 1993.

[11] van Berkel, C.H. Beware the isochronic fork. *Tech. Rep. UR 003/91*, Philips Research Laboratories 1991

[12] Verhoeff, T. Delay-insensitive codes- an overview. *Distributed Computing*, 3(1):1-8, 1988.