QDI Decomposed DIMS Method Featuring Homogeneous/Heterogeneous Data Encoding

PADMANABHAN BALASUBRAMANIAN[¶], and NIKOS E. MASTORAKIS[§]
[¶] School of Computer Science
The University of Manchester
Oxford Road, Manchester M13 9PL
UNITED KINGDOM
padmanab@cs.man.ac.uk
[§] Division of Electrical Engineering and Computer Science
Military Institutions of University Education, Hellenic Naval Academy
Piraeus 18539
GREECE
mastor@hna.gr

Abstract: - The delay-insensitive minterm synthesis (DIMS) method facilitates a robust two-level asynchronous implementation of dual-rail encoded arbitrary combinational logic. However for multilevel realization, logic decomposition becomes indispensable. In this context, this paper describes an elegant technique of performing quasi-delay-insensitive logic decomposition based on set theoretic principles. This includes consideration of generic homogeneous and heterogeneous delay-insensitive data encoding schemes within the purview of the DIMS approach. Through the proposed set theory based logic decomposition rules, it is shown how any combinational logic specification that comprises any number of concurrent inputs can be synthesized in a multilevel fashion on the basis of the DIMS method without compromising on circuit robustness.

Key-Words: - Asynchronous logic design, Delay-insensitive codes, DIMS method, Quasi-delay-insensitivity, Combinational logic, Logic decomposition.

1 Introduction to DIMS Method

The delay-insensitive minterm synthesis (DIMS) technique [1], in its conventional format, is used for the robust two-level asynchronous realization of a combinational logic function. To achieve this, the DIMS approach requires enumeration of 2^n canonical products[1] of a combinational logic function, which is composed of '*n*' distinct primary inputs, with the literals comprising the unique canonical products replaced by their dual-rail encoded data equivalents. The products are then logically summed to synthesize the desired functionality. Due to an imminent input space explosion associated with the DIMS approach, this method is often preferred for robust asynchronous synthesis of small functionality such as discrete logic gates and arithmetic circuit blocks. The problem of input state-space explosion is compounded by the difficulty with logic decomposition. The NCL_D design method [2], asynchronous circuit synthesis based on partial acknowledgement [3], and asynchronous circuit optimization by local input completeness relaxation [4] are some of the popular asynchronous logic design methods that utilize the DIMS technique as part of their synthesis strategy. The DIMS approach differs from Seitz's method [5] in that logical conjunctions are perceived as achieved using Muller C-elements[2] rather than using simple AND gates. Hence the resulting physical realizations tend to properly indicate (acknowledge) the arrival of primary inputs on their primary outputs. Moreover, the primary outputs properly indicate completion of computation and attainment of steady-state within the 'function block', which represents the asynchronous equivalent of a synchronous combinational logic circuit.

[1] A product signifies a logical conjunction of distinct literals.

[2] The C-element outputs a 1(0) if all its inputs are 1(0); in other scenarios it retains its existing steady-state.

ISBN: 978-1-61804-030-5

A typical DIMS solution consists of C-elements in the first level and OR gates in the next level(s) which are used for disjunction of the appropriate product terms. The C-elements are assumed to have unbounded fan-in, while the OR gates can be decomposed arbitrarily as at this stage there is a one-hot code data representation. This is because for every function input only one C-element that synthesizes a unique canonical product term would be activated during the set phase. For physical realization of the DIMS based solution, the C-elements often have to be decomposed. The naive decomposition of a C-element based on the associative axiom could result in the generation of gate orphans (hazards) as illustrated in figure 1(a), while on the contrary merging could eliminate hazards as depicted by figure 1(b). Referring to figure 1(a), given the low-to-high going input transitions ($a\uparrow$, $b\uparrow$), only X↑ results while the steady-state of Y is maintained. Therefore, the transition on the intermediate node X is not reflected on the output Y, which is construed as a gate orphan. Considering figure 1(b), given the input sequence ($a\uparrow$, $b\uparrow$), the steady-state of Y is maintained and unless $c\uparrow$ occurs, Y↑ does not occur thereby avoiding the creation of gate orphans.

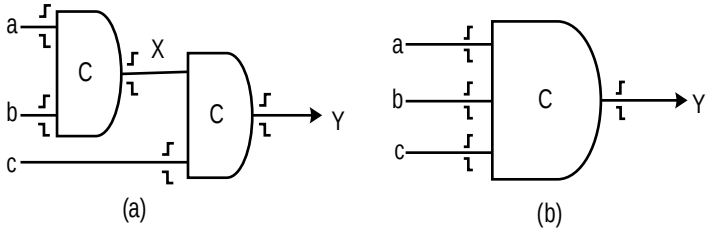

Fig. 1. Hazards due to naive decomposition of a C-element and elimination of hazards by merging

In general, reduction of Boolean equations is not allowed as part of the DIMS approach as it might violate the monotonic cover constraint [6], which implies that the product terms comprising a sum-of-products expression are mutually orthogonal. The function block constructed with its outputs expressed in disjunctive normal form by utilizing all the canonical products is termed strongly indicating, as none of the outputs would become defined/undefined until all the inputs have become defined/undefined. The DIMS approach is similar to an earlier work by Anantharaman [7], but has been subsequently extended into a standard technique for implementation of arbitrary multiple output function blocks in [1]. Let us consider a dual-rail full adder synthesized on the basis of DIMS method for an illustration. The fundamental equations governing the full adder with dual-rail inputs ($a1$, $a0$), ($b1$, $b0$), ($cin1$, $cin0$) and dual-rail outputs ($Sum1$, $Sum0$), ($Cout1$, $Cout0$) are given as follows:

$$Sum1 = a0b0cin1 + a0b1cin0 + a1b0cin0 + a1b1cin1 \quad (1)$$

$$Sum0 = a0b0cin0 + a0b1cin1 + a1b0cin1 + a1b1cin0 \quad (2)$$

$$Cout1 = a0b1cin1 + a1b0cin1 + a1b1cin0 + a1b1cin1 \quad (3)$$

$$Cout0 = a0b0cin0 + a0b0cin1 + a0b1cin0 + a1b0cin0 \quad (4)$$

The dual-rail full adder implementing the above equations is shown in figure 2. The C-element is represented by the AND gate symbol with the marking 'C' on its periphery. A delay-insensitive (DI) circuit is designed to operate correctly irrespective of component (gate) delays and delays encountered with the communicating signal wires i.e. unbounded (arbitrary, but positive and finite) gate delay and wire delay models are assumed. DI circuits feature the most robust unbounded delay model and such circuits are guaranteed to be correct by construction meaning they require no timing verification and can tolerate fluctuations in process parameters, temperature and noise and can also be ported between different technologies with ease, thus featuring excellent design modularity. It was shown in [8] and [9] that C-elements and inverters are the only DI elements and so unfortunately, the class of pure DI circuits would be very limited comprising these two elements. Since the class of pure DI circuits is highly restricted, a weakest compromise to delay-insensitivity was introduced known as the 'isochronic fork' assumption [8] [9]. The isochronic fork timing assumption has been defined by Martin in [9] as: “In an isochronic fork, when a transition on one output is acknowledged, and thus completed, the transitions on all outputs are acknowledged, and thus completed”. A fork refers to a node or junction from

ISBN: 978-1-61804-030-5

where signal wires branch out and an isochronic fork assumption implies that the signal value (say '0' or '1') on all the branching-out wires from the fork is similar at any time instant. Technically, the difference between the delays in the branches of the fork is considered to be negligible in comparison with the delays encountered in the gate elements, and also the switching thresholds in the different gates to which the fork is an input are nearly the same. Though both these assumptions appear to be difficult to realize in smaller geometries, the isochronic fork assumption is usually confined to very small circuit areas. A recent work by Martin et al. [10] shows that practical realization of the isochronicity assumption is feasible even in nano-CMOS technologies where stricter design rules and large parameter variations are commonplace. DI circuits with isochronic fork assumptions are referred to as quasi-delay-insensitive (QDI) circuits, but it is not necessary that every fork be named an isochronic fork in a QDI circuit [9]. In figure 2, isochronic fork assumptions are made with respect to dual-rail primary inputs as they feed many C-gates while forks which feed the OR gates are non-isochronic. Hence it is to be noted that DIMS based solutions are inherently QDI.

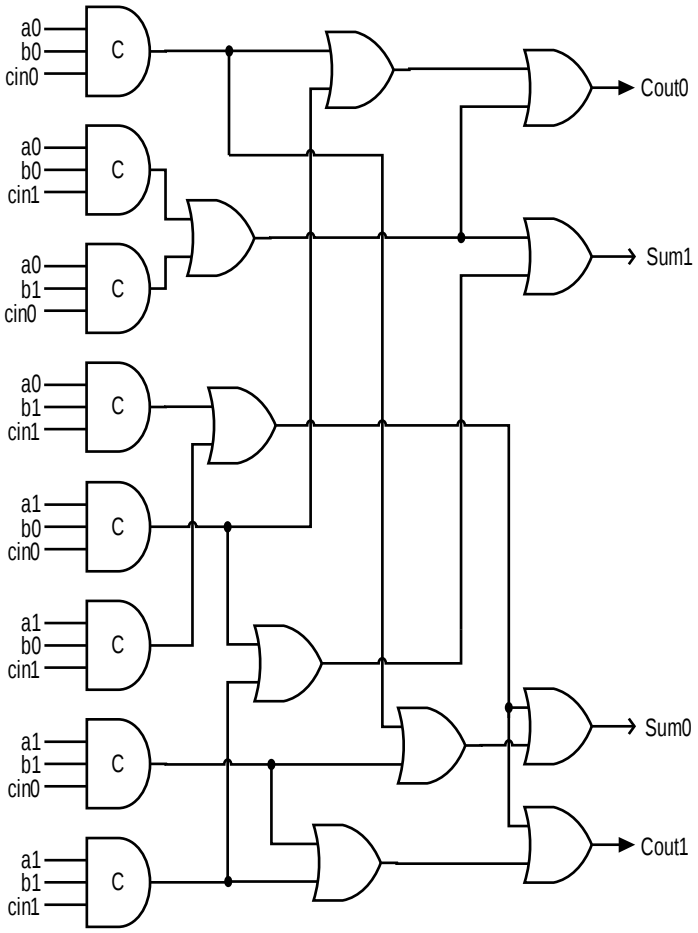

Fig. 2. Full adder realization based on DIMS approach

2 Set Theory Based Terminologies and Definitions

Terminologies governing set theory based logic operations are defined in this section, which are subsequently used to explain the process of QDI logic decomposition in Section 3. These are helpful in the development of multilevel synthesis models for robust asynchronous circuit designs. Unless otherwise stated, the following discussions pertain to dual-rail encoded data paths.

ISBN: 978-1-61804-030-5

2.1 Support Set and Dependency Set

The support[3] set of a product term P, denoted by S(P), enumerates the input variables that are a constituent of the product. On the other hand, a product's dependency set D(P) entails the listing of all the input literals that are a function of the product for its evaluation to logic 'high'. For a product term P specified by $a1b0c1d1$, its S(P) and D(P) are:

$$S(P) = \{a,b,c,d\} \quad (5)$$

$$D(P) = \{a1,b0,c1,d1\} \quad (6)$$

2.2 Products Support Intersection Set and Products Dependency Intersection Set

The products support (dependency) intersection set viz. PSI (PDI) set marks the intersection of the support (dependency) set of two product terms, characterized by the variables (literals) that are common to/shared between the support (dependency) set of both product terms. For example, with $D(P_1)$ and $D(P_2)$ specified by $\{a0,b0,c0,d0\}$ and $\{a0,b0,c0,d1\}$ respectively, their corresponding PSI and PDI sets are given by,

$$PSI\ [S(P_1), S(P_2)] = \{a,b,c,d\} \quad (7)$$

$$PDI\ [D(P_1), D(P_2)] = \{a0,b0,c0\} \quad (8)$$

2.3 Products Relativity Set

The products relativity (PR) set characterizing a product term with respect to another product term identifies or isolates the unique literals in the former compared to the latter. The PR set of product P_1 relative to product P_2 is obtained by computing the set-theoretic difference of the dependency set of product P_1 and the PDI set of both the product terms. It basically amounts to finding the relative complement of PDI set of both products in the dependency set of product term P_1.

$$PR\ [P_1, P_2] = D(P_1) \setminus PDI\ [D(P_1), D(P_2)] \quad (9)$$

Similarly, PR set of P_2 with respect to P_1 is given by,

$$PR\ [P_2, P_1] = D(P_2) \setminus PDI\ [D(P_1), D(P_2)] \quad (10)$$

For example, with $D(P_1) = \{a1,b1,c0,d1,e0,g1\}$ and $D(P_2) = \{b1,c1,d1,e0,g0\}$,

$$PR\ [P_1, P_2] = \{a1,c0,g1\},\ PR\ [P_2, P_1] = \{c1,g0\} \quad (11)$$

2.4 Variables Identification

Variables identification (VI) is an operation that is typically performed on the PR set yielding a set of variables as elements. It is equivalent to characterizing the support of a PR set. With respect to (11),

$$PR\_VI\ [P_1, P_2] = \{a,c,g\},\ PR\_VI\ [P_2, P_1] = \{c,g\} \quad (12)$$

3 Orthogonality and QDI Logic Decomposition

In this section, relationships that govern orthogonal products[4], sum-of-products expression, orthogonal sum-of-products expression, and QDI logic decomposition based on extraction of suitably shared logic functionality and substitution operation are described.

3.1 Mutual Orthogonality Set and Degree of Mutual Orthogonality

The essential relations to be satisfied in order that two product terms P_1 and P_2 may be said to be orthogonal to each other are given below.

$$|S(P_1)| \geq 1,\ |S(P_2)| \geq 1 \quad (13)$$

$$|PR\ [P_1, P_2]| \geq 1,\ |PR\ [P_2, P_1]| \geq 1 \quad (14)$$

The support set of any product term should consist of at least a single variable, which is specified by (13). According to (14), there should be at least one distinct element in $D(P_1)$ relative to $D(P_2)$ and vice-versa; otherwise there would not exist a possibility for P_1 and P_2 to exhibit mutual orthogonality. But the satisfying of

[3] In general, the term 'support' signifies a set. But the term 'set' has been additionally used as a suffix to ensure compatibility with the other set definitions to be introduced in this work. The support of a function (product term) is the set of variables appearing in the function (product term).

[4] Two product terms are said to be orthogonal to each other if their logical conjunction results in a null. For example, $x0y0$ and $x1y1$ are mutually orthogonal.

ISBN: 978-1-61804-030-5

conditions (13) and (14) alone cannot guarantee the existence of an orthogonal relationship between P_1 and P_2, since PR $[P_1, P_2]$ and PR $[P_2, P_1]$ may contain literals associated with different variable indices. For example, product terms P_i and P_j specified by dependency set elements $\{a1,b0,c1\}$ and $\{d1,e0\}$ respectively satisfy the inequalities given in (13) and (14) but are not orthogonal. Hence, if and only if the inequality mentioned in (15) is additionally satisfied, then can the two product terms P_1 and P_2 be labeled as mutually orthogonal, for it shall then be guaranteed that two different instances (literals) of the same support variable would be present in either of the sets being intersected.

$$PR\_VI\ [P_1, P_2]\ \cap\ PR\_VI\ [P_2, P_1] \neq \emptyset \quad (15)$$

The integer measure of the number of variables responsible for establishing the orthogonal relationship between two product terms is called the degree of mutual orthogonality (DMO). Given that (13) – (15) hold good for product terms P_1 and P_2, the mutual orthogonality (MO) set that comprises orthogonal variables and the DMO between P_1 and P_2 are given by,

$$MO = PR\_VI\ [P_1, P_2]\ \cap\ PR\_VI\ [P_2, P_1] \quad (16)$$

$$DMO = |PR\_VI\ [P_1, P_2]\ \cap\ PR\_VI\ [P_2, P_1]| \quad (17)$$

3.2 Sum-of-Products and Orthogonal Sum-of-Products Forms

A sum-of-products (SOP) expression represents a disjunction of product terms, each of which involves a conjunction of literals. An orthogonal sum-of-products (OSOP) form [11] consists of product terms that are all orthogonal to each other, i.e. the product terms do not overlap. Every product is mutually orthogonal to every other product in an OSOP form and therefore the OSOP implicitly satisfies the monotonic cover constraint. The SOP and OSOP expressions for the carry output of a dual-rail full adder are given by (18) – (21) below, where ($a1$,$a0$), ($b1$,$b0$) and ($cin1$,$cin0$) are the dual-rail inputs and ($Cout1$,$Cout0$) is the dual-rail output.

$$Cout1_{SOP} = a1b1 + b1cin1 + a1cin1 \quad (18)$$

$$Cout0_{SOP} = a0b0 + b0cin0 + a0cin0 \quad (19)$$

$$Cout1_{OSOP} = a1b1 + a0b1cin1 + a1b0cin1 \quad (20)$$

$$Cout0_{OSOP} = a0b0 + a1b0cin0 + a0b1cin0 \quad (21)$$

3.3 Criteria for QDI Logic Decomposition

The necessary criteria for performing QDI logic decomposition by way of extracting shared logic between two mutually orthogonal products P_1 and P_2 are listed below. These are also useful to effect decomposition up to a finer granularity in accordance with the elements specified in the base-function set (cell library).

$$|S(P_1)| > 1,\ |S(P_2)| > 1 \quad (22)$$

$$S(P_1) = S(P_2) \quad (23)$$

$$|PR\ [P_1, P_2]| = |PR\ [P_2, P_1]| = 1 \quad (24)$$

The first constraint implies that there should be at least two elements in the support set of both product terms, which is obviously mandatory for decomposition, as a product with a singleton support set reduces to a wire.

The second relation ensures that variables of both product terms are identical, which is an essential criterion for considering extraction of logic shared between them as products with disjoint supports cannot feature any commonality. Assuming that a function consists of a number of product terms, where the support set elements corresponding to all the products are distinct, the function could then be classified as purely read-once[5] and it would be implicitly expressed in its reduced SOP format [12]. In terms of dependency sets, the condition given by (23) conveys that $|D(P_1)| = |D(P_2)|$, i.e. P_1 and P_2 are said to be equipollent.

The third condition is vital as its fulfillment would point to an opportunity for performing QDI logic decomposition involving equipollent products which satisfy relations (22) and (23). It essentially means that there is only one unique literal in P_1 relative to P_2 and vice-versa. Given that the above conditions are upheld, it should be obvious that the equality relation $|PDI\ [D(P_1), D(P_2)]| = |D(P_1)|\text{-}1 = |D(P_2)|\text{-}1$ would hold good.

If two product terms P_a and P_b are not mutually orthogonal, and if they satisfy (22) with $|S(P_a)| > |S(P_b)|$, then a possibility for QDI logic decomposition could exist even though product terms P_a and P_b are not equipollent, provided $D(P_b) \subseteq D(P_a)$. To explain this, let us assume that P_a and P_b are given by $a1b0c0d0e1$ and $b0d0$

[5] A function is said to be 'read-once', if each variable appears only once in its factored form. For example, the Boolean function $F = x(y+z)$ is read-once.

respectively. Provided the activation of P_b would be certainly acknowledged by the next level logic, P_a can be expressed as the conjunction of products $a1c0e1$ and P_b. Indeed, P_b should belong to the cover of a function output different from that of the cover comprising P_a. However, both P_a and P_b cannot be present in the same function cover as P_b would be said to contain P_a, where P_a becomes the covered product term and P_b is the covering product term that can absorb P_a. Also, P_a and P_b cannot be present in different rails (of the dual-rail) of the same encoded function block output as then the system could enter into an illegal state (both 'set' and 'reset' functions would be asserted 'high' simultaneously). Nevertheless, this scenario of logic decomposition does not normally occur in case of indicating circuit synthesis models which consider the entire input space, and it is mentioned here only as a supplementary information addressing special situations.

3.4 Primary and Secondary Product Terms

The need for logic decomposition arises whenever larger product terms are present in a function which cannot be physically implemented due to fan-in restrictions of the cell library and therefore they have to be realized via smaller products. With the conditions mentioned in the above sections satisfied between mutually orthogonal products, say P_1 and P_2, a common product term can be extracted from among them, which shall be referred to as P_3. In this case, the following properties hold good: $D(P_3) \subseteq D(P_1)$ and $D(P_3) \subseteq D(P_2)$. In a similar fashion, $S(P_3) \subseteq S(P_1)$ and $S(P_3) \subseteq S(P_2)$. The size of the product term P_3 would be governed by (25), while its literals are specified by (26).

$$|S(P_3)| = |S(P_1)|-1 = |S(P_2)|-1 \quad (25)$$

$$D(P_3) = PDI\ [D(P_1), D(P_2)] \quad (26)$$

After the process of QDI logic decomposition, the variables and literals of the parent product terms P_1 and P_2 can be enumerated using (27) and (28), (29) respectively.

$$S(P_1) = S(P_2) = S(P_3) \cup PR\_VI\ [P_1, P_2] = S(P_3) \cup PR\_VI\ [P_2, P_1] \quad (27)$$

$$D(P_1) = D(P_3) \cup PR\ [P_1, P_2] \quad (28)$$

$$D(P_2) = D(P_3) \cup PR\ [P_2, P_1] \quad (29)$$

Product terms P_1 and P_2 can be called 'primary product terms' if their support sets are found to be a function of the primary input variables. Given this, P_3 can be referred to as the 'secondary product term' and it is usually substituted into the primary product terms P_1 and P_2 as an intermediate node. The primary product is basically a canonical product term comprising the primary input variables of the function block, while the secondary product is a standard product term formed from a subset of the support set variables of the function. From the preceding discussions, it can be intuitively observed that whenever two product terms qualify as candidates for QDI logic decomposition, they are certainly orthogonal but not vice-versa. For example, P_m and P_n represented by their dependency sets $D(P_m) = \{a0,b1,c1,d0\}$ and $D(P_n) = \{a0,b1,c0,d1\}$ are mutually orthogonal, yet no common logic can be extracted in a QDI fashion as (24) is not satisfied between them. This observation holds good for asynchronous data paths adopting any DI data encoding scheme.

3.5 Data Paths Employing 1-of-*n* DI Codes

The concepts described previously serve as a basis for the following discussion on QDI asynchronous data paths incorporating arbitrary one-hot codes. We shall first discuss this on the basis of 1-of-4 code to provide a specific illustration. Two single-rail inputs can be represented using a 1-of-4 code symbolically as shown in Table 1. It is to be noted that the 1-of-4 encoding shown represents only one of many possible assignments and an arbitrary data mapping is chosen here.

Table 1. Data representation in dual-rail and 1-of-4 encoding schemes

Single-rail inputs		**Dual-rail encoded data**		**1-of-4 encoded data**			
A	B	(A1 A0)	(B1 B0)	E0	E1	E2	E3
0	0	(0 1)	(0 1)	0	0	0	1
0	1	(0 1)	(1 0)	0	0	1	0
1	0	(1 0)	(0 1)	0	1	0	0
1	1	(1 0)	(1 0)	1	0	0	0

ISBN: 978-1-61804-030-5

To avoid the confusion that could result from similar symbolic variable assignments, a condition is imposed whereby the representation of each unique pair of single-rail inputs by a 1-of-4 code equivalent should be distinct in terms of the symbol variables used for corresponding mapping. Notwithstanding, the set definitions mentioned above would not be adequate to address QDI data paths that employ arbitrary combinations of generic 1-of-*n* codes. For example, four single-rail inputs (*m*,*n*,*o*,*p*) are converted into two-pairs and are mapped as highlighted in (30) and (31), which constitutes a valid representation. On the contrary, the mapping (*m*,*n*) ↔ (*q*0,*q*1,*q*2,*q*3) and (*o*,*p*) ↔ (*q*4,*q*5,*q*6,*q*7) is classified as non-permissible because the VI operation would yield the same element.

(*m*,*n*) ↔ (*q*0,*q*1,*q*2,*q*3) (30)

(*o*,*p*) ↔ (*r*0,*r*1,*r*2,*r*3) (31)

Let us assume that a function F is dependent on six input variables (*a*,*b*,*c*,*d*,*e*,*f*), and expressed by the disjunction of two product terms X and Y, which are specified as *a'bcd'e'f* and *a'b'c'd'ef* in single-rail format. With the pairs of input variables (*a*,*b*), (*c*,*d*) and (*e*,*f*) mapped to symbolic notations (*i*0,*i*1,*i*2,*i*3), (*j*0,*j*1,*j*2,*j*3) and (*k*0,*k*1,*k*2,*k*3) respectively, and assuming a similar encoding assignment as shown in Table 1, we have the dependency sets of products X and Y described as thus:

D(X) = {*i*2,*j*1,*k*2} (32)

D(Y) = {*i*3,*j*3,*k*0} (33)

Referring to relations (13) – (15), it can be seen that product terms X and Y jointly satisfy these and hence they are categorized as orthogonal product terms. But as they do not mutually satisfy (24), QDI decomposition is not feasible.

Let us now consider a data path of odd width and let (*a*,*b*,*c*,*d*,*e*,*f*,*g*,*h*,*i*) be the single-rail inputs. Let us have the following permissible mappings utilizing 1-of-2, 1-of-4 and 1-of-8 DI codes for a QDI data path.

a ↔ (*a*0,*a*1) (34)

(*b*,*c*,*d*) ↔ (*m*0,*m*1,*m*2,*m*3,*m*4,*m*5,*m*6,*m*7,*m*8) (35)

(*e*,*f*) ↔ (*n*0,*n*1,*n*2,*n*3) (36)

g ↔ (*g*0,*g*1) (37)

(*h*,*i*) ↔ (*p*0,*p*1,*p*2,*p*3) (38)

The mapped representation *b'c'd'* ↔ *m*8 and *bcd* ↔ *m*0 is used. Let two product terms Z_1 and Z_2 be specified as *a'b'c'd'e'fgh'i'* and *a'bcde'fgh'i'* respectively in their single-rail format. Referring to Table 1 again for the 1-of-4 coded value assignments corresponding to a pair of single-rail data inputs, the dependency sets of the encoded product terms are given as,

$D(Z_1)$ = {*a*0,*m*8,*n*2,*g*1,*p*3} (39)

$D(Z_2)$ = {*a*0,*m*0,*n*2,*g*1,*p*3} (40)

It can be seen that after encoding the product terms Z_1 and Z_2 satisfy (13) – (15). Also, (24) is satisfied between them as their MO set is singleton. Hence, it can be inferred that product terms Z_1 and Z_2 are not only orthogonal to each other but also qualify as potential candidates for QDI decomposition, thus paving the way for logic extraction between them.

4 Example Problem

QDI decomposed DIMS implementations of a combinational benchmark function, *check*, which consists of four single-rail inputs (*a*,*b*,*c*,*d*) and produces a single output (F) are shown in figures 3 and 4 for the case of homogeneous encoding (only dual-rail) and heterogeneous encoding (dual-rail and 1-of-4). The encoding and decoding circuitry is shown in figure 5 that is used to translate data between dual-rail and 1-of-4 encoding protocols. It is assumed for this case study that the base-function set is merely composed of 2-input and 3-input C-elements (CE2s and CE3s respectively). Secondary products correspond to the outputs of first-level C-gates, while primary product terms are obtained as outputs of the second-level C-gates. The conventional DIMS method would require sixteen 4-input C-elements for the homogeneous data encoding convention, which eventually renders the solution not physically realizable given the library constraints. Isochronicity assumption is imposed on primary inputs and intermediate output forks of the first level logic of both QDI circuits.

ISBN: 978-1-61804-030-5

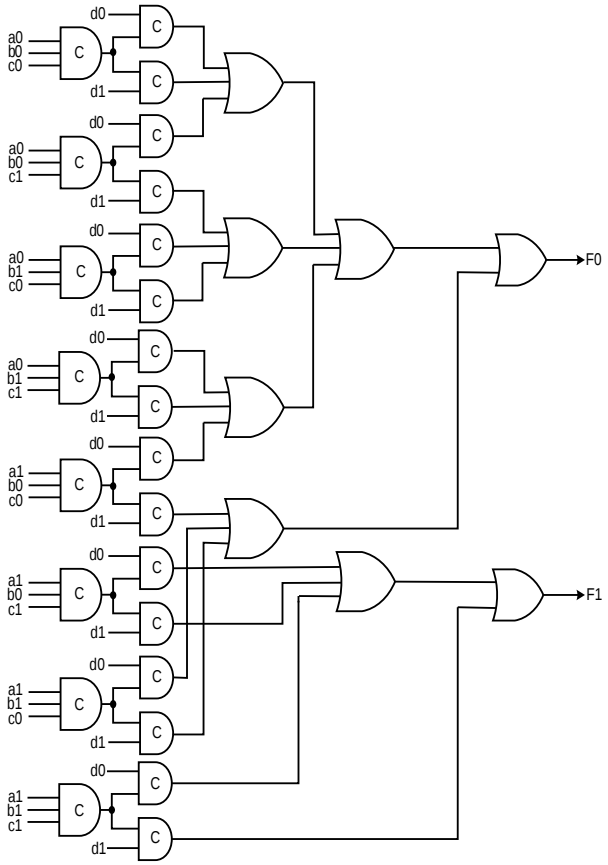

Fig. 3. QDI decomposed realization of *check* function block employing homogeneous data encoding

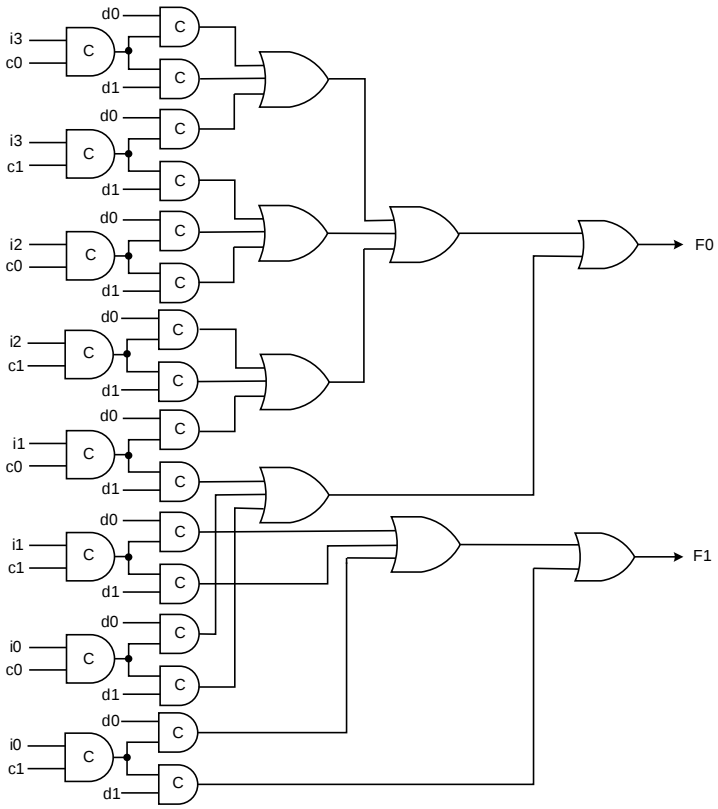

Fig. 4. QDI decomposed synthesis of *check* function block incorporating heterogeneous data encoding

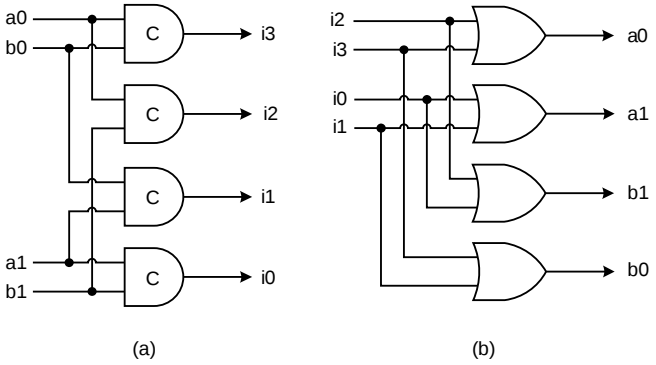

Fig. 5. (a) Dual-rail to 1-of-4 encoder, and (b) 1-of-4 to dual-rail decoder

4 Conclusion

A technique to introduce QDI logic decomposition into the standard DIMS method was discussed in this paper. The proposed technique is based upon set theoretic principles and considers generic homogeneous and heterogeneous delay-insensitive data encoding schemes. The problem associated with physical implementation of a traditional DIMS based solution grows by $O(2^n)$ for increase in inputs by $O(n)$. The proposed QDI logic decomposition procedure has provided a solution to this problem by suggesting a feasible logic optimization strategy as an extension to the DIMS asynchronous logic synthesis method.

References:

1. J. Sparso, and J. Staunstrup, "Delay-insensitive multi-ring structures," *Integration, the VLSI Journal*, vol. 15, pp. 313-340, 1993.

2. Ligthart, K. Fant, R. Smith, A. Taubin and A. Kondratyev, "Asynchronous design using commercial HDL synthesis tools," *Proc. 6th International Symp. on Advanced Research in Asynchronous Circuits and Systems*, pp. 114-125, 2000.

3. Y. Zhou, D. Sokolov and A. Yakovlev, "Cost-aware synthesis of asynchronous circuits based on partial acknowledgement," *Proc. IEEE/ACM International Conf. on Computer-Aided Design*, pp. 158-163, 2006.

4. Jeong and S.M. Nowick, "Optimization of robust asynchronous circuits by local input completeness relaxation," *Proc. Asia and South Pacific Design Automation Conference*, pp. 622-627, 2007.

5. C.L Seitz, "System Timing," in *Introduction to VLSI Systems*, C. Mead and L. Conway (Eds.), pp. 218-262, Addison-Wesley, Reading, MA, 1980.

6. A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen and A. Yakovlev, "Basic gate implementation of speed-independent circuits," *Proc. 31st ACM/IEEE Design Automation Conference*, pp. 56-62, 1994.

7. T.S. Anantharaman, "A delay insensitive regular expression recognizer," *IEEE VLSI Technical Bulletin*, vol. 1, no. 2, pp. 3-15, 1986.

8. A.J. Martin, "Compiling communicating processes into delay-insensitive VLSI circuits," *Distributed Computing*, vol. 1, no. 4, pp. 226-234, December 1986.

9. A.J. Martin, "The limitation to delay-insensitivity in asynchronous circuits," *Proc. 6th MIT Conf. on Advanced Research in VLSI*, pp. 263-278, MIT Press, 1990.

10. A.J. Martin and P. Prakash, "Asynchronous nano-electronics: preliminary investigation," *Proc. 14th IEEE International Symp. on Asynchronous Circuits and Systems*, pp. 58-68, 2008.

11. V.I. Varshavsky (Ed.), *Self-Timed Control of Concurrent Processes: The Design of Aperiodic Logical Circuits in Computers and Discrete Systems*, Chapter 4: Aperiodic Circuits, pp. 77-85, (Translated from the Russian by Alexandre V. Yakovlev), Kluwer Academic Publishers, 1990.

12. I. Newman, "On read-once Boolean functions," in M.S. Paterson (Ed.), *Boolean Function Complexity*, pp. 25-34, London Mathematical Society Lecture Note Series 169, Cambridge University Press, 1992.