A New Design Technique for Weakly Indicating Function Blocks

P. Balasubramanian, D.A. Edwards
School of Computer Science, The University of Manchester,
Oxford Road, Manchester M13 9PL, United Kingdom.
E-mail: (padmanab, doug)@cs.man.ac.uk.

Abstract-**This paper presents a novel technique for gate-level design of combinatorial logic as weakly indicating function blocks. The input state space associated with a function block expands exponentially with a gradual increase in the number of inputs. As a result, large area overhead would incur for an asynchronous realization. Hence, a novel design methodology for realizing combinational logic as a function block is developed under the discipline of quasi-delay-insensitivity with four-phase handshaking and dual-rail encoding, whilst trying to mitigate the area overhead. The focus is on design adhering to the weakly indicating timing regime. Based on analysis with some combinational benchmarks and widely used logic circuit functionality, the proposed method is found to enable compact realizations and appears to be promising for weakly indicating function block design comprising many inputs and outputs.**

I. INTRODUCTION

Digital logic design has been dominated by synchronous solutions for the past several decades. The renewal of interest in and requirement of asynchronous design is largely motivated by the need to overcome the problems associated with clocking. Future deep sub-micron technologies would be characterized by irregular parameter variations (voltage and temperature variations, process changes and noise), which could make asynchronous design an attractive solution. It is expected that parametric variance of device delay might reach 35% by 2020 [1]. Since asynchronous circuits are self-timed, they tend to absorb the deviations of device characteristics. Self-timed design is a method for designing asynchronous circuits such that their correct operation depends neither on the speed of their components nor on the delays of the communication wires.

The scope for asynchronous design techniques are increasing due to a number of potential advantages:

- Enhanced modularity (permitting design reuse)
- No problems with clock signal skew
- Operational scalability (system may slow down but does not fail)
- Robustness to parametric variations, for example, temperature and supply voltage [2]
- Reduced susceptibility to electro-magnetic interference
- Minimization of power consumption due to the absence of clock activity and
- the portion of the circuit dissipating power is that involved in computation [3] paving way for automatic power-down of unused circuitry.

Unlike synchronous circuits, where a global clock is used to control when the outputs of the combinational logic are read, asynchronous circuits must operate correctly independent of any delays in circuit elements and as such, they dispense with the need for a global clock. The vast majority of existing design automation flows target synchronous circuits. Even when asynchronous designs leverage existing tool flows, they introduce large area overheads. As a result, asynchronous function block implementation requires special techniques in comparison with the synthesis of combinational logic for synchronous digital circuits.

The remainder of this paper is organized as follows. Section 2 describes the function block. In section 3, some relevant works have been discussed. Section 4 describes the novel weak-indication function block design technique with the help of a block diagram and also lists the logic decomposition constraints necessary for an asynchronous style implementation. The proposed method is illustrated by means of a case study in section 5. Results corresponding to the different functionalities considered are given in section 6. In the next section, we make the concluding remarks and highlight directions of our further work.

II. FUNCTION BLOCK – CHARACTERIZATION

A function block is the asynchronous equivalent of a digital combinatorial circuit [4]: it computes one or more outputs from a set of input signals. Apart from computing the desired function outputs based on the function inputs, a function block must also be transparent to handshaking that is implemented by its surrounding latches. The transparency to handshaking is what makes function blocks different from combinational logic circuits. Besides it should have a valid combinatorial structure meaning that it should not have dangling inputs or outputs and no feed-back signal paths.

Function blocks can be strongly indicating (SI) or weakly indicating (WI) depending on how they behave with respect to the signaling transparency. In this work, we restrict ourselves to the WI timing regime and discuss how to arrive at realizations suitable for implementation with standard cells along with custom designed C-elements, forming part of the cell library. A function block is categorized as WI if it starts to compute and produce valid (empty) outputs as soon as possible, i.e. when some but not all its input signals might have become valid (empty). Such a function block never produces all valid (all empty) outputs until after all inputs have become valid (empty). This behavior is defined by Seitz's weak conditions [10]. It has also been proved in [10] that if independent function blocks satisfy the weak-indication constraints, then they can be combined to form

978-1-4244-2277-7/08/$25.00 ©2008 IEEE

larger function blocks, suitable for design of cascaded arithmetic circuits.

In the case of circuits using dual-rail (DR) signals, function blocks implicitly indicate completion. In contrast to bundled-data encoding, DR encoded data does not require a separate request wire; instead it is embedded within the data wires. Moreover, each data wire x is now represented by two data wires $x0$ and $x1$. A transition on the $x0$ wire indicates that a zero has been transmitted, while a transition on the $x1$ wire indicates that a one has been transmitted. But simultaneous transitions on both $x0$ and $x1$ is not allowed and hence considered to be invalid. Since the request wire is embedded within the data wires, a transition on either $x0$ or $x1$ informs the receiver about the validity of the data. In other words, they will always produce functionally correct outputs independent of the wire delays whereas bundled-data protocol is not delay-insensitive (DI). The 4-phase protocol, also known as return-to-zero (RTZ) protocol, is similar to the 2-phase protocol except for the fact that it is level-sensitive. Therefore, the request wire has to go low before it can go high again leading to an intermediate RTZ phase between two successive data values. The advantage of 4-phase protocol is that logic processing elements can be made much simpler and familiar logic gates can be used, whereas designs based on 2-phase protocol would require complex control circuits to process transitions.

Circuits designed following the four-phase protocol DR approach are generally quasi-delay-insensitive (QDI), since the class of DI circuits is rather small [5]. QDI is as robust as the DI class to variable operating conditions and transistor variations [7]. A circuit is QDI if and only if the production rule set describing it is stable and non-interfering [8]. It is also an attractive design style mainly for the simple timing closure and analysis that it permits. QDI circuit design assumes that both operators and wires can take an arbitrary time (finite and positive) to switch, except for certain wires that form isochronic forks [6] [9] (weakest compromise to DI). If the delays to all the end-points of a forking wire are identical, then the wire-fork is called isochronic. The isochronic fork assumption has been defined in terms of the acknowledgement by Martin in [5] 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".

III. Previous Work

DI (also include QDI) circuits can be identified as either pertaining to control circuits or datapaths. While control circuits are those that realize the sequencing of actions of a computation, datapaths are those that deal with the manipulation and transmission of data. Datapath design, in general, is more difficult than control circuit design and relatively much of the published research has concentrated on the design of control circuitry. Though a generalized method for design of delay-insensitive datapath circuits has been discussed in [11], it largely corresponds to the design of an adder circuitry. Infact, the transistor level implementation of an adder circuit using CMOS technology in [11] represents the ultimate for the case of a WI adder design. Recent works have also primarily targeted implementation of arithmetic circuits, especially integer addition [12] [13] [14]. The DIMS approach [12], widely adopted for strongly indicating adder designs can be modified as in [4] to suit the weak-indication timing model. But it is not suitable for synthesizing combinational logic with many inputs, as it severely exacerbates the area overhead, besides giving rise to many unacknowledged transitions in the circuit (circuit orphans) for naïve C-element decomposition. The notion of a quad-gate has been introduced in [14]. Based on published results, it can be observed that the quad-gate based implementation of a DI carry-lookahead adder with speedup (DICLASP) [13], mentioned as quad-DICLASP in [14], is found to consume more transistors and significantly more power dissipating than the original DICLASP of [13], while reporting a marginal improvement in delay for the best case addition. For the worst case 32-bit addition, there is degradation in speed. The merits/demerits of quad-gate based implementation for realization of arbitrary combinational logic has not yet been reported in current literature. In [15], early output logic has been proposed, which has a flavor of the weak-indication phenomenon in that some/all valid outputs may be produced without requiring the arrival of all valid inputs. However the converse does not hold well, i.e. weak-indication is not strictly synonymous with the early output logic of [15]. Though theoretical models based on Petri nets for the WI timing regime were made available in [16], general purpose datapath synthesis algorithms for this timing model have not yet been developed [21]. Further, as mentioned in [20], weak-indication logic synthesis involves relatively more complexity and is difficult. This paper tends to precisely address this issue.

IV. WI Function Block Design

As mentioned in the previous section, much of the previous work have addressed the design of arithmetic circuits, paying little attention to realization of an arbitrary non-regenerative logic functionality. An asynchronous implementation of an arbitrary combinational logic usually requires the generation of all minterms (standard product terms), which is $O[2^n]$ for 'n' inputs, causing an input state space explosion. Hence the main issue dealt with in this paper is the efficient realization of any arbitrary combinational logic as a QDI weak-indication function block pertaining to the discipline of 4-phase handshaking with DR encoding using standard library cells (including 2-input and 3-input C-elements), which poses a real and significant challenge for asynchronous logic design, as in general, the area overhead becomes massive in comparison with synchronous combinational logic synthesis solutions.

In contrast with a SI function block design, which exhibits worst case latency as in a synchronous circuit, actual case latency can be expected from a WI function block. In order to facilitate good area optimized implementations (measured in terms of two-input gate count [19] and transistor cost [20]), we bank on widely used conventional multiple-output minimization strategies available [18] and utilize them for translation to asynchronous logic while complying with

weak-indication property by incorporating novel logic transformations and decomposition techniques (which guarantee speed-independence), while simultaneously satisfying the monotonic cover constraint (MCC) [23].

A. Terminologies and Definitions

Before we mention the main issues and constraints involved in synthesis and decomposition, new terminologies are elucidated alongside existing ones for clarity.

1) Support set and Dependency set of a Boolean cube

A literal is an variable or its inversion (say, k or k'). A Boolean cube, C is a conjunction of literals. The support set of a cube, $S(C)$ entails the enumeration of all the literals that are a function of the cube, while a cube dependency set, $D(C)$ entails enumeration of all the support set literals in their actual form that the cube depends upon for its evaluation to a logic '1'.

For a cube C assigned with $ef'g'h$, its $S(C) = \{e,f,g,h\}$ and $D(C) = \{e,f',g',h\}$.

2) Cubes Support Intersection set and Cubes Dependency Intersection set

The intersection of the support set of two cubes (dependency set of two cubes) is characterized by the literals that are common to both the cubes. This is referred to by an intersection set, *CSI* (*CDI*). For example, with $D(C_1)$ and $D(C_2)$ described by $\{a,b,c',d\}$ and $\{a,b',c',f\}$ respectively, $CSI[S(C_1), S(C_2)] = \{a,b,c\}$ and $CDI[D(C_1), D(C_2)] = \{a,c'\}$ and hence their respective cardinalities are: $|CSI[S(C_1), S(C_2)]| = 3$ and $|CDI[D(C_1), D(C_2)]| = 2$.

3) Covering and Covered cubes [17], Cover extent, CE

We identify a cube C_1 as fully covering another cube C_2, if $D(C_2) \subseteq D(C_1)$ and hence the following equality holds: $|CDI[D(C_1), D(C_2)]| = |D(C_2)|$ and $CE = |D(C_2)|$.

4) DNF and MODNF

A Boolean formula is said to be in disjunctive normal form (DNF) if it is a disjunction of clauses, each of which is a conjunction of literals. The mutually orthogonal DNF (MODNF) of a logic function consists of a set of conjunctions which are mutually orthogonal cubes [24], i.e. the cubes do not overlap. Every Boolean cube is mutually orthogonal to every other Boolean cube in a MODNF. Formally, when two Boolean cubes C_1 and C_2 are mutually orthogonal, then the following inequalities would hold good: $|CSI[S(C_1), S(C_2)]| \geq 1$ and $|CDI[D(C_1), D(C_2)]| \geq 0$.

B. Synthesis issues and Decomposition constraints

- Obtain a minimized solution for all the m function outputs (in positive phase) of the multiple-input multiple-output functionality description, originally in single-rail format. These correspond to the expressions for true function outputs, after DR encoding.
- Obtain minimized expressions for all the m function outputs (in negative phase). These correspond to the Boolean equations for false outputs after DR encoding.
- Transform both the true and false function output expressions into MODNF. This is accomplished through redundancy insertion using identity law and also by recursive application of absorption and distributive axioms of Boolean algebra.
- Where C_1 and C_2 are not mutually orthogonal and $|CSI[S(C_1), S(C_2)]| = 0$ and $S(C_1)$ is singleton ($S(C_2)$ could also be singleton); by absorption law, we therefore get: $|D(C_2)| = |S(C_2)| = |S(C_1)| + 1 = |D(C_1)| + 1$.
- Where C_1 and C_2 are not mutually orthogonal and if $|CSI[S(C_1), S(C_2)]| = 0$ and $S(C_1)$ is not singleton ($S(C_2)$ is also not singleton), then redundancy insertion through identity axiom is followed by application of distributive and absorption laws. Subsequently, additional cube(s) get introduced into the logic network.
- Consider the whole functionality as a global combinatorial network with DR inputs and outputs.
- For k cubes, perform cube dependency intersection with each of the remaining (k-1) cubes in the network.
- If for two cubes C_1 and C_2, the equality relationship $|CDI[D(C_1), D(C_2)]| = |D(C_1)| = |D(C_2)|$ holds good, then both C_1 and C_2 are replaced by an intermediate node symbol and it is substituted back into its parent nodes. This analogy can be extended to any number of cubes in the network.
- In the above scenario, $CE = |D(C_1)| = |D(C_2)|$. If the equality is not satisfied, then opt for the case, where $|CDI[D(C_1), D(C_2)]| = |D(C_1)| - 1 = |D(C_2)| - 1$. Then replace the shared literals by an intermediate node and substitute it back into its parent nodes via, a conjunction. However, this is permissible only when $HD(C_1, C_2) = 1$ in order to preserve speed-independence, where *HD* signifies the Hamming distance between the cube vectors.
- A cube in the transformed Boolean network is to be chosen only once, also an intermediate node (cube) resulting from the network should be chosen only once, whether for substitution or for covering.
- The covering cube absorbs the covered cube.
- If a cube covers more than one cube in the network, then priority for a particular cube to be covered would be based upon a maximal value of *CE* for the covered cube.
- A covered cube can be the covering cube for another cube and a covered cube can be substituted into as many covering cubes as possible apart from its actual covering cube provided *CE* for the other covering cubes with respect to this covered cube is equal to the cardinality of the dependency set of the covered cube.
- Uncovered smaller cubes (bound dictated by maximum fan-in) might exist in the final network.
- Synchronization of the DR encoded inputs (for both data and spacer values) is achieved through block I1 (shown in figure 1), which has $2n$ DR inputs and a single output.
- Decomposed multi-level realization of the function outputs, which satisfy the MCC is represented by block I2. The DR encoded inputs to the block may be only a subset of the actual inputs, whereas there would certainly be $2m$ DR outputs.
- Block I3 basically ensures that the weak-indication criteria are satisfied by ensuring synchronization between the output of block I1 and any two outputs of block I2, enabling self-timed operation. But it is recommended not to combine those two complementary data rails of a function output which are already associated with the longest path

delay as it would further increase their delay. The logical values of the inputs from block I2 to block I3 are indeed preserved on its output side. The number of actual function outputs extracted from block I2 is (2*m*-2).

The above logic decomposition constraints facilitate weak-indication function block realization using only C-elements and OR gates. The implementation is partly technology-dependent in the sense that the synthesis and decomposition process facilitates incorporating 3-input C-elements besides 2-input Muller elements. This is preferable as the depth of the synthesized network would be reduced.

The block diagram of the proposed method is shown in figure 1.

V. Illustrative Example

The weak-indication circuit implementation of a 4-to-2 encoder with 8 DR inputs (*i*11, *i*10, *i*21, *i*20, *i*31, *i*30, *i*41, *i*40) and 4 DR outputs (***y*11**, ***y*10**, ***y*21**, ***y*20**) is portrayed by figure 2 as an illustration of the proposed method. Of these, (*i*11,*i*10) correspond to the most significant DR inputs, while (***y*11**,***y*10**) represent the most significant DR outputs. The portions of the circuit corresponding to the three functional blocks of figure 1 have been clearly marked and identified as blocks I1, I2 and I3 in figure 2.

To comment on the area complexity of the resultant multi-level speed-independent network; for the sake of simplicity, the 2-input gate count metric used in [19] and the transistor cost metric used in [20] have been employed here. In case of the former, a *n*-input C-element (CE_n) is equated to (*n*-1) 2-input C-elements ((*n*-1)CE_2), a CE_2 is considered as equal to four 2-input gates ($4G_2$) and an *n*-input gate (G_n) is replaced by (*n*-1)G_2. In the latter case, where a strongly indicating network is synthesized using only CE_2 and 2-input OR gate (OR_2) components, the cost of a CE_2 is equated to 10 and that of an OR_2 as 6.

While we have adopted the former approach as it is for estimating the gate count; to compute the transistor cost, we have additionally considered the cost of a 3-input C-element as equal to 12, in line with the library specification. But, we have decomposed all *n*-input OR gates into (*n*-1) OR_2 gates, for ease of calculation. This results in a difference between actual and estimated transistor cost, as highlighted in Table I, for a DR weak-indication 4-to-2 encoder. However, the difference is not reflected in the gate count parameter as it is normalized.

TABLE I
TWO-INPUT GATE COUNT AND TRANSISTOR COST FOR WI 4:2 ENCODER

Logic function	2-input gate count	Transistor cost
4-to-2 Encoder	79 (actual)	212 (actual)
	79 (estimated)	220 (estimated)

We define the generalized optimization rules that are valid for disjunctive operations: a DNF sub-expression can be isolated from a logic network without creating an orphan if and only if:

- it is associated with both (only) a variable and its complement separately (or)
- at least one of its fork outputs is fed to an OR-gate input.

VI. Results and Discussion

Analysis has been carried out by considering some IWLS benchmarks [22] and few widely used digital logic circuits and these are listed in Table II. Extracting SOP sub-expressions would enable further reduction in transistor cost for logic functions. On the whole, for the circuit functionalities listed in Table II, the proposed method enables compact realizations (known thus far) for combinational benchmarks. However, this excludes SOP sub-expression extraction and variable substitution.

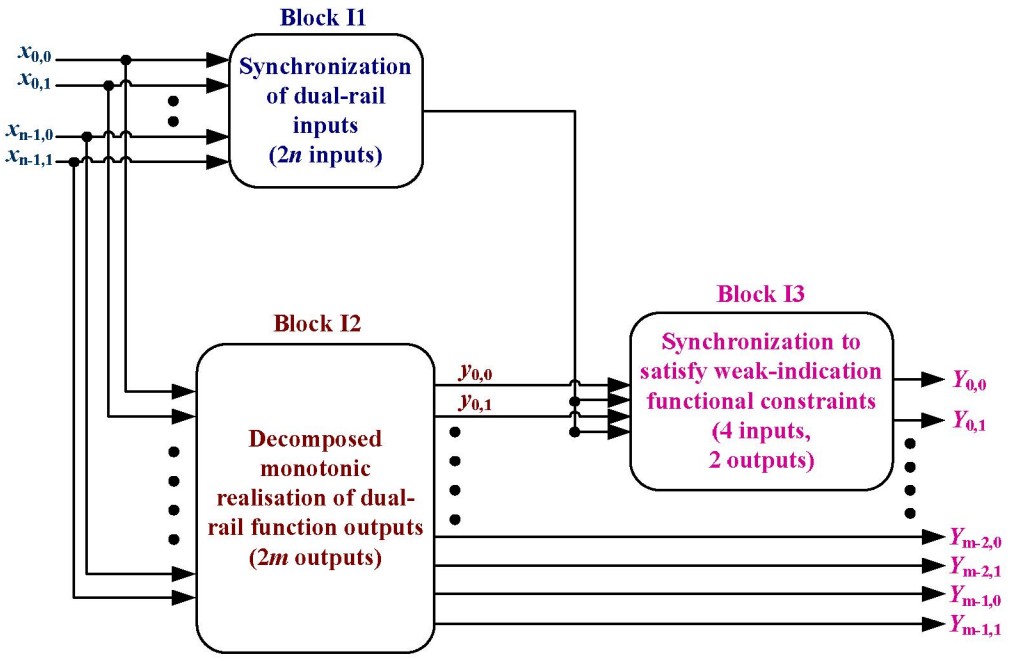

Figure 1. Block diagram of proposed architecture for a weakly indicating (WI) function block design

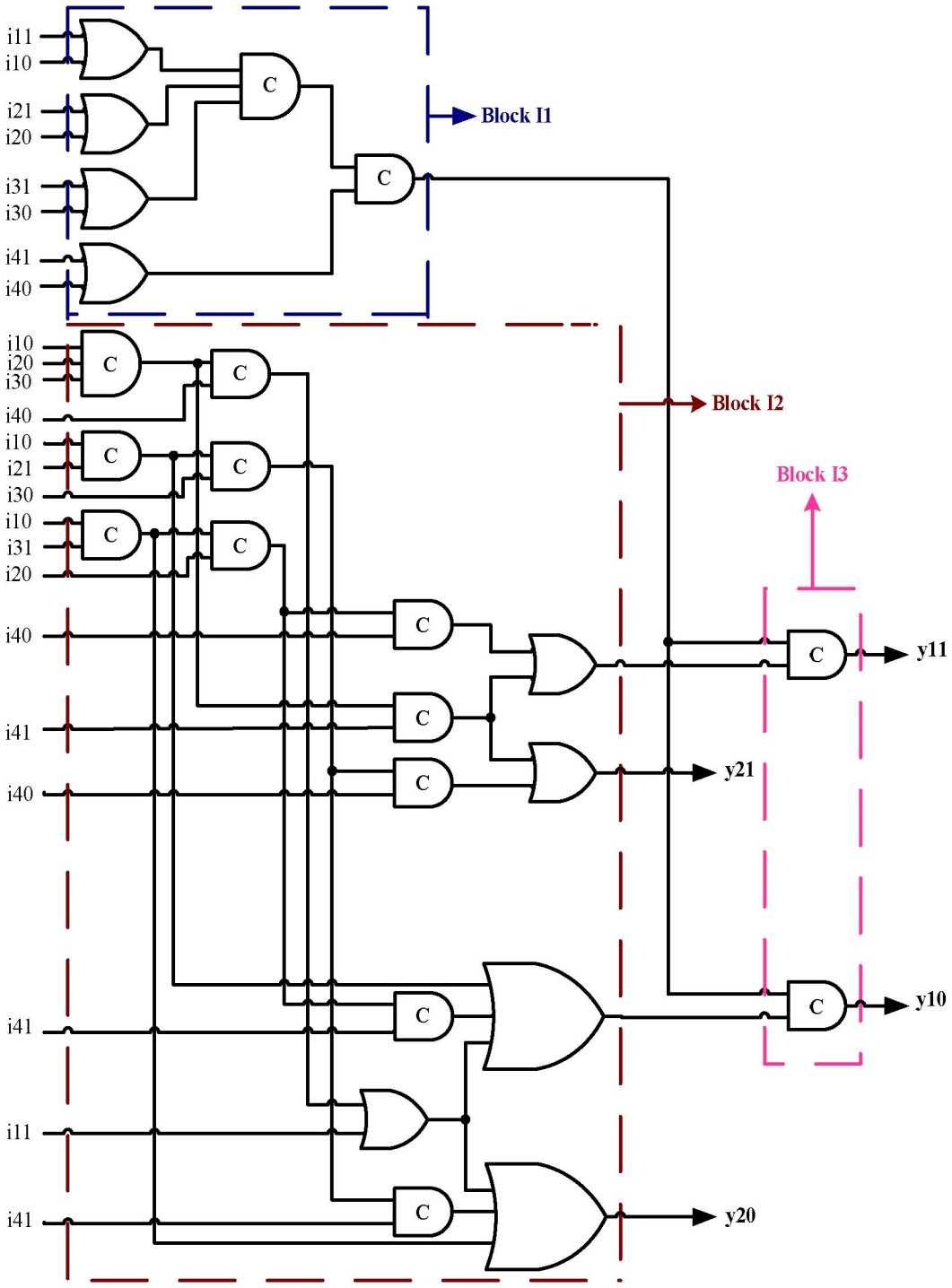

Figure 2. Optimal implementation of a weak-indication 4-to-2 Encoder

For higher order functions, a selective consideration of the input space is necessitated in order to facilitate speed-independent decompositions. This adds some more complexity into the decomposition process, which is quite complex as it decomposes several gates in the entire implementation space simultaneously. Hence speed-independent decompositions of larger functional specifications need to be analyzed by selectively considering a portion of the entire input space, employing valid multiple-input C-element decomposition rules.

Additionally, replacement of certain Muller elements by logical conjunction operators is presently being considered. This is of interest and importance as it will tend to further reduce the gap (in terms of area overhead) between synchronous and asynchronous logic realizations.

VII. Conclusion and Further Work

This paper presents a novel technique for gate level WI function block design of an arbitrary combinatorial logic, within the ambit of QDI with DR encoding and 4-phase

handshaking. To our knowledge, there has not been any method formulated to address the specific problem dealt with in this paper. Improving the efficacy of the synthesis technique and trying to address larger specifications forms a part of our present work. Power/performance evaluation of the designs pertaining to the proposed heuristic would be our next step.

TABLE II
TWO-INPUT GATE EQUIVALENT COUNT AND TRANSISTOR COST FOR WI REALIZATIONS OF COMBINATIONAL FUNCTIONS

Combinatorial logic and its specification	2-input gate count	Transistor cost
newtpla1 (10 inputs, 2 outputs)	167	480
con1 (7 inputs, 2 outputs)	183	514
newcwp (4 inputs, 5 outputs)	143	444
wim (4 inputs, 7 outputs)	130	422
newbyte (5 inputs, 8 outputs)	181	566
squar5 (5 inputs, 8 outputs)	500	1356
dc1 (4 inputs, 7 outputs)	193	614
c17 (5 inputs, 2 outputs)	115	300
newapla2 (6 inputs, 7 outputs)	337	914
4-bit priority encoder (4 inputs, 4 outputs)	55	168
8-bit priority encoder (8 inputs, 8 outputs)	128	422
3-to-8 decoder with active high Enable (4 inputs, 8 outputs)	104	350
4-to-16 decoder with active high Enable (5 inputs, 16 outputs)	213	758
5-to-32 decoder with active high Enable (6 inputs, 32 outputs)	442	1670

ACKNOWLEDGMENT

The authors acknowledge the support of EPSRC, UK for this research through the SEDATE project grant EP/D052238/1.

REFERENCES

[1] SIA's ITRS Report 2005, http://www.itrs.net

[2] A.J. Martin, S.M. Burns, T.K. Lee, D. Borkovic and P.J. Hazewindus, "The first asynchronous microprocessor: the test results," *Computer Architecture News*, vol. 17, no. 4, pp. 95-110, June 1989.

[3] K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken and F. Schalij, "A fully asynchronous low power error corrector for the DCC player," *Proc. IEEE ISSCC*, pp. 88-89, 1994.

[4] J. Sparso and S. Furber (Eds.), *Principles of Asynchronous Circuit Design – A Systems Perspective*, Kluwer Academic Publishers, 2001.

[5] A.J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," *Proc. 6th MIT Conference*, pp. 263-278, MIT Press, 1980.

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

[7] T.M. Mak, "Is CMOS more reliable with scaling," presented at *IEEE International On-Line Testing Workshop*, July 2002.

[8] R. Manohar and A.J. Martin, "Quasi-delay-insensitive circuits are Turing-complete," *Caltech CS Technical Report*, CS-TR-95-11, 1995.

[9] K. van Berkel, "Beware the isochronic fork," *Integration, the VLSI journal*, vol. 13, no. 2, pp. 103-128, June 1992.

[10] C.L. Seitz, *In Introduction to VLSI Systems*, *Chapter 7 – System Timing*, C.A. Mead and L.A. Conway (Eds.), Addison-Wesley, 1990.

[11] A.J. Martin, "Asynchronous datapaths and the design of an asynchronous adder," *Formal Methods in System Design*, vol. 1, no. 1, pp. 119-137, July 1992.

[12] J. Sparso and J. Staunstrup, "Delay-insensitive multi-ring structures," *Integration, the VLSI journal*, vol. 15, no. 3, pp. 313-340, Oct. 1993.

[13] F.-C. Cheng, S. Unger and M. Theobald, "Self-timed carry-lookahead adders," *IEEE Trans. on Computers*, vol. 49, no. 7, pp. 659-672, July 2000.

[14] X. Li and J.W. Sanders, "Efficient function block implementation of self-timed circuits," *Proc. 47th IEEE Intl. Midwest Symposium on Circuits and Systems*, vol. 2, pp. 269-272, 2004.

[15] C. Brej, "Early Output Logic and Anti-Tokens," *PhD thesis*, University of Manchester, 2006.

[16] A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno and M.P. Koutny, "On the models for asynchronous circuit behaviour with OR causality," *Formal Methods in System Design*, vol. 9, no. 3, pp. 189-234, November 1996.

[17] B. Teel and D. Wilde, "A logic minimizer for VLSI PLA design," *Proc. 19th IEEE Design Automation Conference*, pp. 156-162, 1982.

[18] R.K. Brayton, A.L. Sangiovanni-Vincentelli, C.T. McMullen and G.D. Hachtel, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

[19] I. David, R. Ginosar and M. Yoeli, "An efficient implementation of Boolean functions as self-timed circuits," *IEEE Trans. on Computers*, vol. 41, no. 1, pp. 2-11, January 1992.

[20] W.B. Toms and D.A. Edwards, "Efficient synthesis of speed independent combinational logic circuits," *Proc. 10th Asia and South-Pacific Design Automation Conference*, pp. 1022-1026, 2005.

[21] SEDATE project, *Case for Support: Proposed Research*, 2006.

[22] K. McElvain, "IWLS '93 Benchmark Set: Version 4.0," *distributed as part of the MCNC International Workshop on Logic Synthesis*, May 1993.

[23] A. Kondratyev, M. Kishinevsky and A. Yakovlev, "Hazard-free implementation of speed-independent circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 17, no. 9, pp. 749-771, September '98.

[24] A.D. Zakrevski, *Logical Synthesis of Cascade Circuits* (translated from Russian), Energiya, Moscow, 1974.