Power, Delay and Area Efficient Self-Timed Multiplexer and Demultiplexer Designs

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

Abstract—**Efficient gate level design methods for robust self-timed realization of arbitrary size multiplexer and demultiplexer function blocks, using elements of a commercial standard cell library are discussed in this paper. While the optimal self-timed multiplexer implementations correspond to strong-indication, the optimal self-timed demultiplexer implementations pertain to weak-indication phenomenon. The design methods presented are scalable and enable achieving simultaneous optimization in power, delay and area parameters.**

I. INTRODUCTION

Self-timed (ST) logic design, in general, guarantees that the required functionality is satisfied irrespective of delays in the circuit components or signal wires. Hence, they are inherently elastic, comprising the ability to deal with device irregularities, which are becoming prominent in ultra deep submicron technologies. The latest SIA's ITRS update on design [1] projects parametric variation of device delay to increase from a current figure of 10% to 25% by 2020. Design blocks reuse (as a percentage of all logic) is anticipated to increase from a current figure of 38% to 55% by 2020. Interest in ST designs is on the rise, as they feature an innate tolerance to variations in supply voltage, temperature and fabrication process parameters. In addition, they promise greater modularity, less EMI, no clock distribution and clock skew problems and eliminate unnecessary power consumption.

Multiplexers (MUXes) and demultiplexers (DEMUXes) are common building blocks of data paths and are used extensively in numerous applications including processor busses, network switches and digital signal processing stages incorporating resource sharing. In this paper, we consider efficient asynchronous realizations of MUX and DEMUX logic as function blocks, characterized by four-phase handshaking protocol and dual-rail input encoding, the robust self-timed approach having its roots in Muller's pioneering work of the 1950's and 60's [2]. Circuits designed following the four-phase protocol DR encoding approach are generally quasi-delay-insensitive (QDI), since the class of DI circuits is rather small [3]. QDI is as robust as the DI class to variable operating conditions and transistor variations [4]. A circuit is QDI if and only if the production rule set describing it is stable and non-interfering [5]. It is also an attractive design style mainly for the simple timing closure and analysis it permits. QDI circuit design assumes that both operators and wires can take an arbitrary time (finite and positive time) to switch, except for certain wires that form isochronic forks [6] (weakest compromise to delay insensitivity). The isochronic fork assumption has been defined by Martin in [6] 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"*. The 4-phase signaling protocol is also known as the return-to-zero protocol, wherein input data alternates between valid data and empty data (also called spacer). The dual-rail (DR) input encoding protocol is a delay-insensitive (DI) protocol, with the DR code being the widely used member of the family of DI codes [7].

A function block is the asynchronous equivalent of a synchronous combinational logic circuit [8]. But apart from satisfying the requisite functionality, it is required to possess the additional attribute of being transparent to handshaking as implemented by its surrounding latches. A function block can be classified as either strongly indicating or weakly indicating depending on how it behaves with respect to the handshaking transparency. In case of the former, all the inputs need to become valid/empty before valid/empty outputs can be produced, while in case of the latter, valid/empty outputs can be produced as soon as a subset of the inputs have become valid/empty. However, unless all its inputs have not become valid/empty, all its outputs should not become valid/empty. The above conditions formulated by Seitz [9] ensure that the generation of all the primary outputs in a function block wholly indicates the arrival of all the input data and also the completion of computation within the block.

II. PREVIOUS WORK AND PROBLEM STATEMENT

A number of self-timed logic design techniques exist either for generic or specific function block design adhering to the property of indication [9] [10] [11] [12] [13] [14], employing 4-phase handshaking and DR input encoding. However, many of these suffer from limitations with increase in primary circuit inputs. Methods of [9], [10] can conform to both strong-indication and weak-indication timing models, but they

A major part of this research is funded by EPSRC, UK through the SEDATE project grant EP/D052238/1.

require at the minimum the generation of all minterms, which is O[2^n] for 'n' inputs, resulting in a huge input space consideration. In case of [8], decomposition of multiple inputs C-element is often necessitated. This can give rise to unacknowledged transitions on gate outputs (usually called gate-orphans) within the circuit making it Q^nDI for naïve decomposition, leading to violation of speed-independence conditions. Despite a cautious speed-independent (SI) logic decomposition, the area overhead severely exacerbates. In [9], a self-timed function block design was proposed, but decomposition procedures for the monotonic implementation of the combinatorial DRN were not put forth, which restricts the scalability of this approach. Also, the DRN can evaluate to the correct empty state in case of a spacer, signaling the completion of computation within the function module even with some internal nodes not reset and thereby gate-orphans get created. A method to effectively realize function blocks using conventional logic gates has been proposed in [12], but it overlooks the partial gate-orphans that are generated within the circuit, which might be difficult to ascertain. Also, adequate care may be required to ensure that they do not become critical by extensive timing analysis. [13] presents an efficient method, but it necessitates building a library using custom-defined standard cells (27 proprietary macros are used) for technology mapping. A recent work [14] dealing with the synthesis of QDI circuits (using 2-input C-elements and 2-input OR gates) corresponding to DR input encoding (also, any generic *m*-of-*n* code), encompasses a decomposition technique incorporating elements of both conventional rectangle covering based multilevel logic synthesis and speed-independent decomposition. Though it is a versatile method, it also suffers from the problem of input space explosion as the entire input space is to be covered (i.e., all the canonical product terms of a function need to be considered).

It is a proven fact that if individual function blocks satisfy strong/weak-indication constraints, then they can be combined to form larger function blocks, which also pertain to a similar timing regime [9]. This property can be utilized to aid the construction of iterative logic circuits, such as MUXes, DEMUXes, adders and magnitude comparators. Hence, the main issue addressed in this paper is to efficiently realize self-timed MUX and DEMUX functionality of any specification in a robust asynchronous style by adhering to the property of indicatability, within the ambit of 4-phase handshaking protocol and DR encoding, whilst satisfying the monotonic cover constraint [15]. The elements of a commercial standard cell library are being used for physical realization.

III. Terminologies and Definitions

A. *Support set and Dependency set of a Boolean cube*

The support set S(C) entails the enumeration of all the literals that are a function of the cube, while a cube's dependency set D(C) entails enumeration of all its support set literals in their actual form for its evaluation to a logic '1'.

For a cube C specified by *ab'c'd*, its S(C) and D(C) are:

$$S(C) = \{a,b,c,d\} \quad (1)$$

$$D(C) = \{a,b',c',d\} \quad (2)$$

B. *Cubes Support Intersection set (CSI), Cubes Dependency Intersection set (CDI) and Polarity Eliminated CDI set*

The intersection of the support set of two cubes (dependency set of two cubes) is characterized by the literals that are common to the support set (dependency set) of both the cubes. This is referred to as CSI (CDI). The polarity eliminated CDI (CDI_{PE}) set consists of the variables of CDI set represented in their uncomplemented form. For e.g. with $D(C_1)$ and $D(C_2)$ specified by $\{a',b,c,d\}$ and $\{a',b',c,f\}$ respectively, the corresponding CSI, CDI and CDI_{PE} sets are,

$$CSI\ [S(C_1), S(C_2)] = \{a,b,c\} \quad (3)$$

$$CDI\ [D(C_1), D(C_2)] = \{a',c\} \quad (4)$$

$$CDI_{PE}\ [D(C_1), D(C_2)] = \{a,c\} \quad (5)$$

C. *Covering cube, Covered cube [16] and Cover extent*

We say a cube C_1 as fully covering another cube C_2, if $D(C_2)$ is a subset of $D(C_1)$. Cover extent (CE) is a measure, which basically quantifies the degree of sharing (common variables) between the two Boolean cubes C_1 and C_2.

$$CDI\ [D(C_1), D(C_2)] = D(C_2) \text{ and } CE = |D(C_2)| \quad (6)$$

D. *Sum-of-Products and Disjoint Sum-of-Products [17]*

A Boolean formula is said to be in sum-of-products (SOP) form if it consists of a disjunction of standard product terms, each of which is a conjunction of literals.

A Boolean equation is said to be in mutually orthogonal or disjoint SOP (MOSOP or DSOP) form if and only if it consists of an array of conjunctions which are mutually orthogonal, i.e. the cubes do not overlap or they are disjoint. Every Boolean cube is mutually orthogonal to every other Boolean cube in a DSOP. When two Boolean cubes C_1 and C_2 are mutually orthogonal, the following inequalities are valid.

$$|CSI\ [S(C_1), S(C_2)]| \geq 1 \quad (7)$$

$$|CDI\ [D(C_1), D(C_2)]| \geq 0 \quad (8)$$

E. *Mutual Orthogonality set and Degree of Mutual Orthogonality*

Mutual orthogonality set, MO characterizes or isolates the input variables that are responsible for making two Boolean cubes (say C_1 and C_2) mutually orthogonal. It is given by the set-theoretic difference of CDI and CSI, of cubes C_1 and C_2.

$$MO\ [C_1, C_2] = CSI\ [S(C_1), S(C_2)] \setminus CDI_{PE}\ [D(C_1), D(C_2)] \quad (9)$$

The degree of mutual orthogonality (DMO) between two primary input cubes C_1 and C_2, DMO, is an integer measure of the number of primary inputs in which C_1 and C_2 exhibit orthogonality. A generalization of the DMO between C_1 and C_2 is then given by,

$$DMO = |MO\ [C_1, C_2]| \quad (10)$$

F. *Speed-Independent Shared Cube*

If and only if, for two mutually orthogonal cubes C_1 and C_2, (11) and (12) are satisfied, then a common cube can be extracted from them, which we shall refer to as the speed-independent shared cube, SISC. Hence, between C_1 and C_2, DMO is unity. Subsequently, both C_1 and C_2 can be represented in terms of a conjunction involving the SISC. Assuming (11) and (12) are satisfied by C_1 and C_2, let us label the SISC extracted from them as C_3. Hence, $D(C_3)$ is a subset of $D(C_1)$ and $D(C_2)$. Also $S(C_3)$ is a subset of $S(C_1)$ and $S(C_2)$.

$$\text{CSI}\,[S(C_1), S(C_2)] = S(C_1) = S(C_2) \quad (11)$$

$$|\text{CDI}\,[D(C_1), D(C_2)]| = |D(C_1)|\text{-}1 = |D(C_2)|\text{-}1 \quad (12)$$

The elements of $D(C_3)$ are found out using (13).

$$D(C_3) = \text{CDI}\,[D(C_1), D(C_2)] \quad (13)$$

$$\text{CE} = |D(C_1)|\text{-}1 = |D(C_2)|\text{-}1 = |S(C_1)|\text{-}1 = |S(C_2)|\text{-}1 \quad (14)$$

The terminologies (some are proposed) mentioned above describe speed-independent logic decomposition rules anew based on set theory which form the basis of robust ST designs.

IV. Strong-Indication Multiplexer Designs

The regularity implicit in MUX functionality can be best exploited to facilitate their efficient ST implementations. To clarify this, the basic equations governing the true and false outputs of a 2:1 MUX are first given.

$$y1 = a1s0 + b1s1 \quad (15)$$

$$y0 = a0s0 + b0s1 \quad (16)$$

Equations (15) and (16) are minimum MOSOP forms, despite being the minimum SOP expressions for a 2:1 MUX. It is easy to comprehend that the general expressions for true and false outputs of an arbitrary 2^n:1 MUX with n select inputs correspond to a minimum MOSOP form. Hence, the problem now relatively narrows down to effective speed-independent logic decomposition. Figures 1, 2 and 3 portray a 2:1 MUX implementation based on the methods of [9], [10] and [14] respectively. The C-element is indicated by the marking of letter 'C' within an AND gate, in the diagrams that follow.

Two design techniques have been proposed at the gate level: a **s**trong-**i**ndication **d**esign using **C**-elements and **O**R gates (SIDCO) and a **s**trongly **i**ndicating design utilizing **C**-elements, **A**ND gates and **O**R gates (SIDCAO). The structural block diagram representation of an arbitrary MUX design (SIDCO) is shown in figure 4.

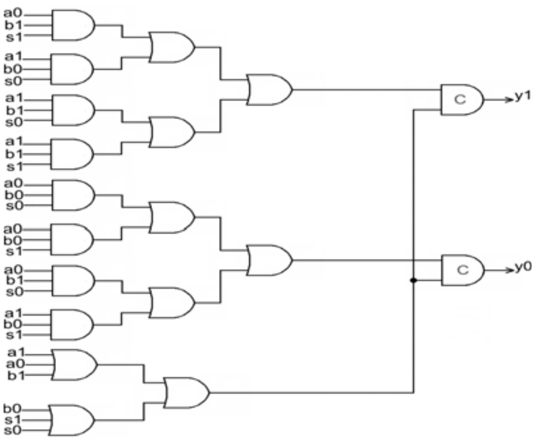

Figure 1. Seitz's 2-to-1 MUX realization

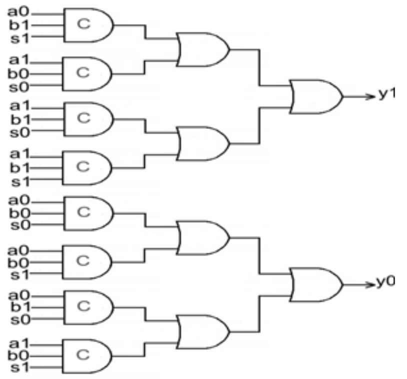

Figure 2. DIMS 2-to-1 MUX implementation

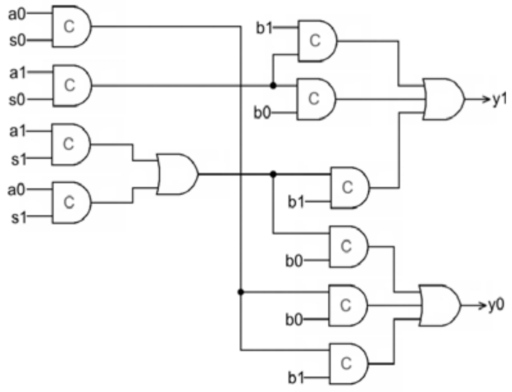

Figure 3. Toms 2-to-1 MUX synthesis

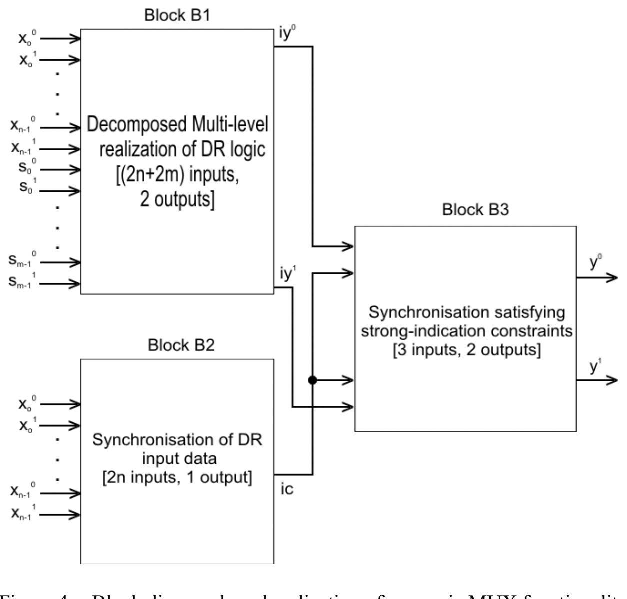
Block B1
Decomposed Multi-level realization of DR logic [(2n+2m) inputs, 2 outputs]
Block B2
Synchronisation of DR input data [2n inputs, 1 output]
Block B3
Synchronisation satisfying strong-indication constraints [3 inputs, 2 outputs]
iy⁰
iy¹
ic
y⁰
y¹

Figure 4. Block diagram based realization of a generic MUX functionality

In figure 4, block B1 contains the SI decomposed multi-level logic realization of a MUX functionality (with m select inputs and n data inputs; where $n = 2^m$), implemented in a ST fashion, which strictly satisfies the monotonic cover constraint (MCC). Block B2 guarantees the arrival of all the DR data inputs for both valid data and spacer values. Block B3 is mainly meant to ensure that the strong-indication criterion is satisfied by synchronizing arrival of all the data inputs with the outputs of function block B1. iy^0 and iy^1 are logically equivalent to y^0 and y^1. Figure 5 shows the proposed realization of a 2:1 MUX logic based on this design style.

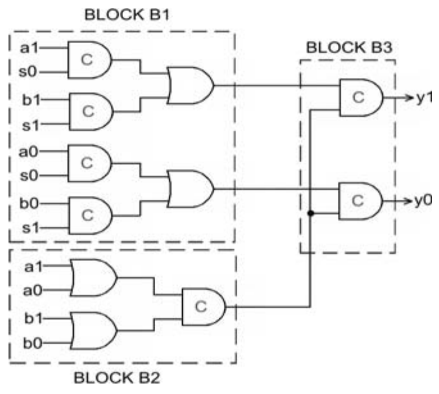

Figure 5. Proposed 2:1 MUX logic (SIDCO)

An alternative design is possible with AND gates replacing the C-elements in the first logic level of block B1. As a result, block B2 would now have ($2n$ + $2m$) inputs, to satisfy the property of indication. This leads to a slightly different synthesis solution, as can be seen in figure 6. Since MUX logic has only a single output, weak-indication is not possible.

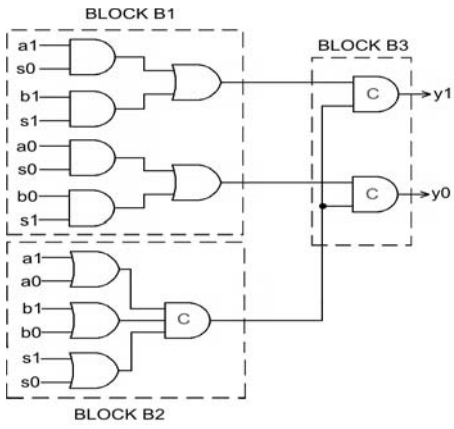

Figure 6. Alternative realization of 2:1 MUX logic (SIDCAO)

V. Weak-Indication Demultiplexer Designs

The ST realization of data distributor functionality is also based on a general design methodology, represented by the block diagram illustration in figure 7. However, block B1 realization involves some complexity, in that a translation of the minimum SOP forms of the true and false DEMUX outputs into their respective minimum MOSOP forms is first necessary, followed by an effective SI decomposition. Thus a **w**eak-**i**ndication **d**esign based on **C**-elements and **O**R gates (WIDCO) is possible. Also, another logic realization based on **C**-elements, **A**ND gates and **O**R gates is also feasible (WIDCAO), with a slight modification to the overall structure shown in figure 7. In this case, block B2 would now consist of ($2m$ + 2) inputs. Also iy_0^1 is additionally fed to block B3 and synchronized with the signal sc of block B2 to produce y_0^1, though both these are logically equivalent signals. The two different implementations of a 1:4 DEMUX are portrayed by figures 8 and 9 respectively. DEMUX realizations based on other methods have been omitted here for reasons of brevity.

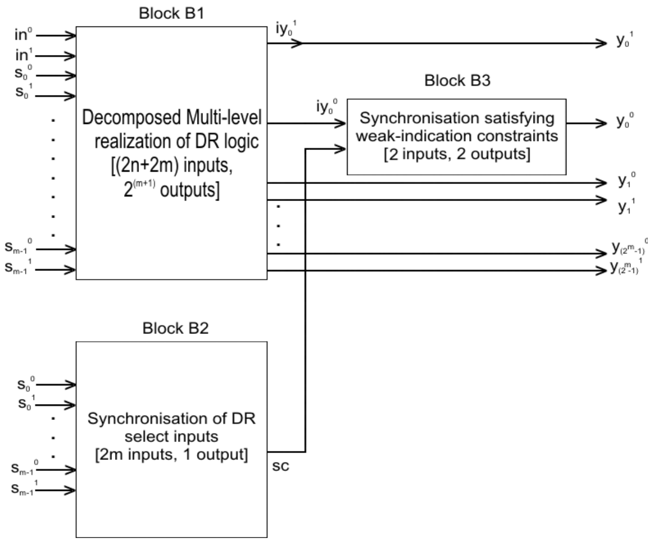
Block B1
Block B2
Block B3
in⁰
in¹
s₀⁰
s₀¹
sₘ₋₁⁰
sₘ₋₁¹
iy₀¹
iy₀⁰
Decomposed Multi-level realization of DR logic [(2n+2m) inputs, 2^(m+1) outputs]
Synchronisation satisfying weak-indication constraints [2 inputs, 2 outputs]
Synchronisation of DR select inputs [2m inputs, 1 output]
sc
y₀¹
y₀⁰
y₁⁰
y₁¹
y(2^m-1)⁰
y(2^m-1)¹

Figure 7. Block diagram based generic DEMUX functionality realization

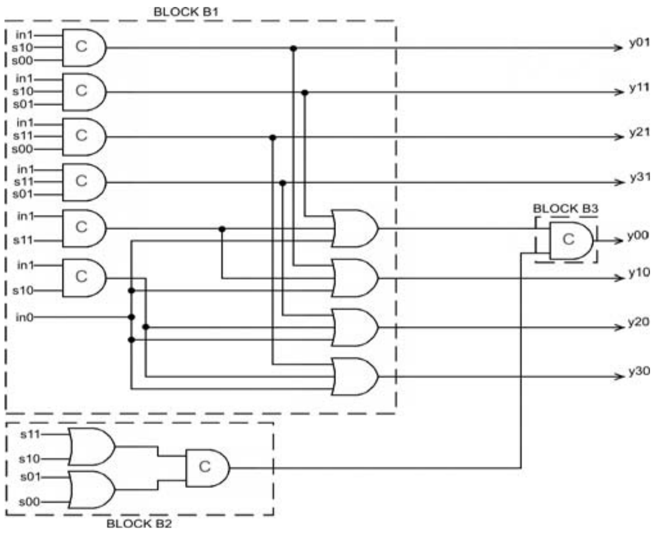

Figure 8. Proposed 1:4 DEMUX logic (WIDCO)

For direct MUX and DEMUX realizations, extraction of SISC constitutes an essential step. In case of MUX logic, they are primarily a unique conjunction of the select inputs. In case of higher order MUXes, the granularity of the SISC is set at a maximum. A parent SISC could then give rise to two off-springs (child nodes), and these child nodes can act as parent SISCs, provided each has two off-springs. This hierarchy is extendable for function realization of higher dimensions.

Throughout this work, the synthesis (mainly logic decomposition) of both MUX and DEMUX functionality is primarily technology-dependent with focus on delay optimization, on the foundation of a base function set comprising the following cells: AND2, AND3, AND4, OR2, OR3 and Muller C-element functionalities (CE2, CE3 and CE4) described using complex gates (AO222, AO2222 and AO12), of the high-speed 130nm Faraday CMOS standard cell library. Strongly indicating DEMUX designs are also possible based a modification of the above block diagram; nevertheless they would only be at the expense of increase in area, delay and power metrics and so they have not been considered.

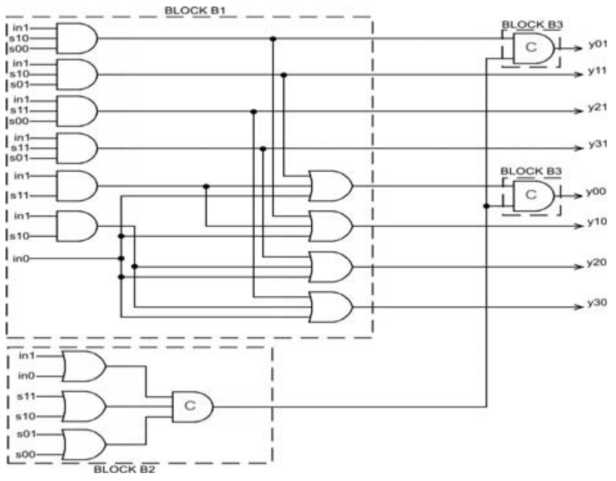

Figure 9. Alternative implementation of 1:4 DEMUX logic (WIDCAO)

VI. Simulation Mechanism, Results and Conclusions

The simulation set-up for MUX and DEMUX logic are depicted by figures 10 and 11 respectively. The primary inputs for both the MUX and DEMUX logic are assumed to arrive from the environment. The input acknowledge signal from the environment (ideally from the succeeding stage logic) is either embedded into the data path logic pertaining to the function block, where possible, or synchronized with each of the outputs of the function block using separate latches.

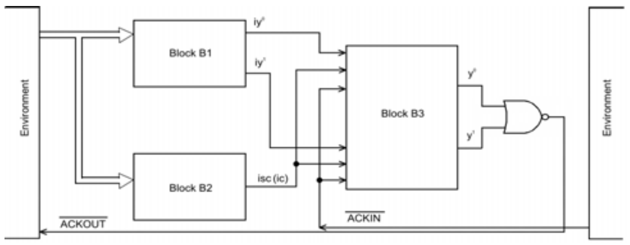

Figure 10. Simulation set-up for MUX functionality

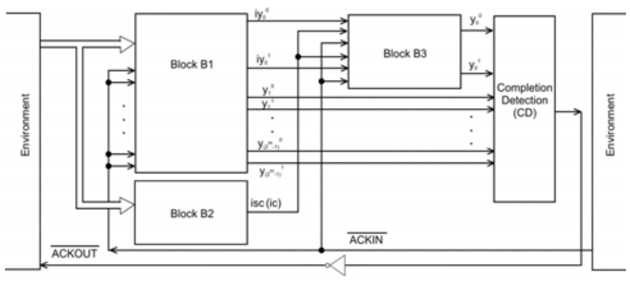

Figure 11. Simulation set-up for DEMUX functionality

The input sequences used for simulation represent testing of the MUX and DEMUX DR outputs for all unique input combinations. The input patterns are fed to the MUX and DEMUX circuits every 4ns. The maximum data path delay is the actual propagation delays encountered while traversing the longest path from a primary input signal of the current stage function block to the inverted acknowledge signal generated out of it, after crossing the output latches. This signal is in turn meant to be fed back as the acknowledge input for the previous stage. Though forward latency specifies the actual worst case combinational delay encountered within a function block, the maximum data path delay gives the summation of forward latency and the delay associated with the CD circuit.

TABLE I. Simulation results for different MUX designs

MUX size	Logic realization method	Total power (μW)	Path delay (ns)	Cells area (μm^2)
2:1	Seitz [9]	23.14	0.75	155
	DIMS [10]	17.88	0.88	242
	Toms [14]	21.39	1.00	167
	SIDCO	27.58	0.81	130
	SIDCAO	28.98	0.94	118
4:1	Seitz_tree	104.65	1.33	393
	DIMS_tree	51.82	1.65	670
	Toms	56.14	1.75	577
	SIDCO	44.24	1.05	317
	SIDCAO	54.76	1.21	224
8:1	Seitz_tree	231.58	2.15	836
	DIMS_tree	121.88	2.41	1526
	Toms_tree	130.90	2.48	1265
	SIDCO	78.82	1.40	748
	SIDCAO	87.54	1.40	429
	SIDCO_SIDCO_tree	150.08	1.48	654
	SIDCAO_SIDCO_tree	95.12	1.38	431
	SIDCAO_SIDCAO_tree	112.45	1.51	461
16:1	Seitz_tree	349.15	2.60	1818
	DIMS_tree	345.13	2.82	3334
	Toms_tree	312.34	3.28	2837
	SIDCO	138.99	1.74	1315
	SIDCAO	160.85	1.63	1014
	SIDCO_SIDCO_tree	367.38	1.59	1429
	SIDCAO_SIDCO_tree	201.08	1.60	953
	SIDCAO_SIDCAO_tree	232.92	1.70	929
32:1	Seitz_tree	718.62	3.07	3674
	DIMS_tree	716.99	3.58	6854
	Toms_tree	649.60	4.00	5653
	SIDCO	257.82	2.08	2464
	SIDCAO	273.06	1.87	2160
	SIDCO_SIDCO_tree	765.41	1.92	2838
	SIDCAO_SIDCO_tree	400.06	1.83	1863
	SIDCAO_SIDCAO_tree	486.85	2.17	1865

A 2-input NOR gate at the output of block B3 performs the function of completion detection (CD) for MUX logic as shown in figure 10, while for the DEMUX logic a conventional CD circuitry (composed of OR gates and a C-element tree) is required. The simulations have all been performed using Cadence and Synopsys tools on a Linux platform, targeting the high-speed 130nm Faraday (UMC) CMOS process for a typical PVT corner. The recommended supply voltage of 1.2V was used, at an ambient temperature of

25°C. The MUX and DEMUX designs of different approaches exhibit fanout-of-2 output drive strength, while the inputs possess the driving strength of the minimum sized inverter in the cell library. Appropriate minimum sized buffer cells were used for the logic realizations so as to eliminate timing violations. Power, delay and area metrics for the MUX functionality are given in Table I and those for the DEMUX functionality are mentioned in Table II. Total power dissipation is the summation of dynamic (switching + internal) and leakage power components. Path delay refers to the maximum delay encountered in the data path, as explained before, and cell area indicates the combined area of data path logic, output registers and CD circuitry.

TABLE II. SIMULATION RESULTS FOR DIFFERENT DEMUX DESIGNS

DEMUX size	Logic realization method	Total power (μW)	Path delay (ns)	Cells area (μm^2)
1:2	Seitz [9]	37.11	0.90	144
	DIMS [10]	30.86	0.97	135
	Toms [14]	30.99	0.98	135
	WIDCO	32.35	1.28	123
	WIDCAO	40.41	1.16	137
1:4	Seitz_tree	74.50	1.35	298
	DIMS_tree	69.30	1.52	385
	Toms	75.27	1.63	327
	WIDCO	77.73	1.62	329
	WIDCAO	79.21	1.50	278
1:8	Seitz_tree	160.18	1.87	713
	DIMS_tree	156.43	2.28	1033
	Toms_tree	161.94	2.34	688
	WIDCO	166.43	2.16	810
	WIDCAO	162.77	2.02	574
1:16	Seitz_tree	416.99	2.62	1304
	DIMS_tree	418.46	2.90	1813
	Toms_tree	435.21	2.92	1523
	WIDCO	360.10	2.76	1460
	WIDCAO	362.14	2.47	1228
	WIDCO_WIDCO_tree	610.12	2.74	1443
	WIDCAO_WIDCO_tree	386.92	2.52	1384
	WIDCAO_WIDCAO_tree	408.85	2.56	1145
1:32	Seitz_tree	850.62	3.03	3052
	DIMS_tree	979.64	3.73	4514
	Toms_tree	902.41	3.65	3076
	WIDCO	704.49	3.24	2649
	WIDCAO	700.59	2.92	2414
	WIDCO_WIDCO_tree	1381.89	3.20	3431
	WIDCAO_WIDCO_tree	864.39	2.83	3369
	WIDCAO_WIDCAO_tree	822.40	3.00	2394

For the MUX and DEMUX logic, tree structures are essential for the other methods [9] [10] [14] to facilitate delay-optimized implementations for MUXes (DEMUXes) with 4 (8) inputs and more, as direct realizations may not be feasible or would incur heavy area and considerable delay and power penalty. This is because of the exponential increase in input space by O(2^n), but both direct and tree type structures are practically feasible based on the proposed approach. A 4:1 MUX is formed with two 2:1 MUXes in the first level and a 2:1 MUX in the second level; an 8:1 MUX with two 4:1 MUXes in the first level and a 2:1 MUX in the second level; a 16:1 MUX with four 4:1 MUXes in the first level and a 4:1 MUX in the second level and a 32:1 MUX with four 8:1 MUXes in the first level and a 4:1 MUX in the second level, to achieve delay optimized implementations. It is the converse for DEMUX logic realization based on tree structures. In case of MUX logic, SIDCO_SIDCO tree refers to the combination where a MUX based on SIDCO was used for the first and second levels of the tree. SIDCAO_SIDCO tree refers to the combination where a MUX based on SIDCAO was used for the first level of the tree alone. With a SIDCAO utilized for both the first and second levels, a SIDCAO_SIDCAO tree structure results. Based on similar lines, WIDCO_WIDCO, WIDCAO_WIDCO and WIDCAO_WIDCAO tree structures can be obtained for DEMUX logic. The above mentioned logic tree cascades for MUX and DEMUX designs constitute block B1 of figures 4 and 7 respectively. Nevertheless, block B2 would require modification for SIDCAO_SIDCAO and WIDCAO_WIDCAO tree structures. The internal signals of Seitz, SIDCAO_SIDCAO and WIDCAO_WIDCAO tree structures are carefully indicated to preserve gate-orphan freedom. The proposed MUX and DEMUX design methods enable direct (cumbersome with other approaches) and tree type realizations, which are better than those of other methods in terms of power, delay and area; especially with higher orders, evident from the values listed in Tables I and II.

REFERENCES

[1] SIA's ITRS report 2007 edition, Available: http://www.itrs.net

[2] D.E. Muller, "Asynchronous logics and application to information processing," in *Switching Theory in Space Technology*, Stanford University Press, pp. 289-297, 1963.

[3] A.J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," *Proc. 6th MIT Conf. on Adv. Res. in VLSI*, pp. 263-278, 1980.

[4] T.M. Mak, "Is CMOS more reliable with scaling?," *Proc. IEEE Intl. On-Line Testing Workshop*, July 2002.

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

[6] A.J. Martin, "Compiling communicating processes into delay-insensitive VLSI circuits," *Dist. Comp.,* vol. 1(4), pp. 226-234, 1986.

[7] T. Verhoeff, "Delay-insensitive codes: an overview," *Distributed Computing*, vol. 3, no. 1, pp. 1-8, 1988.

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

[9] C.L. Seitz, "Chapter 7 – System Timing", in *Introduction to VLSI Systems*, C.A. Mead and L.A. Conway (Eds.), Addison-Wesley, 1980.

[10] J. Sparso and J. Staunstrup, "Delay-insensitive multi-ring structures," *Integration, the VLSI journal*, vol. 15, no. 1, pp. 313-340, Oct. 1993.

[11] I. David et al., "An efficient implementation of Boolean functions as self-timed circuits," *IEEE Trans. on Comp.,* vol. 41(1), pp. 2-11, 1992.

[12] X. Li and J.W. Sanders, "Efficient function block implementation of self-timed circuits," *Proc. 47th MWSCAS*, vol. 2, pp. 269-272, 2004.

[13] K.M. Fant and S.A. Brandt, "NULL convention logic: a complete and consistent logic for asynchronous digital circuit synthesis," *Proc. Intl. Conf. on Appl. Spec. Sys., Arch. and Processors*, pp. 261-273, 1996.

[14] W.B. Toms, "Synthesis of Quasi-Delay-Insensitive Datapath Circuits," *PhD thesis*, University of Manchester, 2006.

[15] A. Kondratyev et al., "Hazard-free implementation of speed-independent circuits," *IEEE Trans. on CAD,* vol. 17(9), 749-771, 1998.

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

[17] T. Sasao (Ed.), *Logic Synthesis and Optimization*, Kluwer, MA, 1993.