Multi-Level Implementation of Asynchronous Logic Using Two-Level Nodes

Igor Lemberski*, Petr Fišer**

** Baltic International Academy, Riga, Latvia (e-mail: Igor.Lemberski@bsa.edu.lv)*
*** Czech Technical University in Prague, FIT, Dept. of Digital Design, Prague, Czech Republic (e-mail: fiserp@fit.cvut.cz)*

Abstract. A novel synthesis method of a dual-rail asynchronous multi-level logic is proposed. The logic is implemented as a monotonous multi-level network of minimized AND-OR nodes together with the completion detection logic. Each node is a hazard-free structure. It is achieved based on the product term minimization constraint that the authors have formulated and proved in their previous paper. The MCNC and ISCAS benchmark sets were processed and the area overhead with respect to the synchronous implementation was evaluated. Then the implementation complexity of the proposed method and a state-of-the-art method based on the duplication of every gate was compared. A considerable improvement was obtained.

Keywords: asynchronous logic, decomposition, multi-level implementation, Boolean network, node

1. Introduction

The asynchronous logic is classified depending on the mode of interaction with the environment. In the *input-output mode*, the environment is allowed to change the input state once the new output state is produced. There is no assumption about the internal signals and the environment is allowed to change the input state before the circuit is stabilized in response to the previous input state. In the *fundamental mode*, the logic operates based on the following discipline: the environment changes the input state once the output state has changed in response to the current input state and each gate inside the circuit is stable. Both design methodologies assume either bounded (a maximal value is known) or unbounded (a maximal value is unknown) gate and wire delays.

In case of the fundamental mode (accepted in this paper) with the *bounded delays*, the moment when the environment may change the input state is estimated based on the worst case propagation delay [Unger, 1969]. Within this model, only one input signal can be changed at a time. In [Nowick, 1993], the generalized fundamental mode was proposed where multiple input changes are allowed during a narrow time interval. For such a mode, the method of hazard-free two-level implementation was proposed [Nowick, 1995]. The multi-level (hazard not increasing) transformation is applied to optimize the implementation [Unger, 1969 and Kung, 1992]. The methods of hazard-free technology mapping were proposed in [Beerel, 1996 and Siegel, 1993].

In case of the *unbounded delays*, the circuit should be capable to recognize the moment when input and output states have changed. For this purpose, both inputs and outputs are implemented using a dual-rail encoding. To change an input state the environment should reset it first (change to so called space state). The output state resets too, as a result. After that the environment sets a new input state. It implies a new output state. The multi-level implementations of the dual–rail asynchronous logic were proposed in [Cortadella, 2004 and Ligthart, 2000]. These methods are based on the initial circuit decomposition into simple (OR, AND, NOR, NAND, etc.) two-input gates. Further, each gate is mapped into *DIMS* [Sparsø, 1992] or into a so called *threshold gate* [Ligthart, 2000]. As a result, the circuit total complexity is very high. In [Cortadella, 2004], each simple gate is doubled to ensure *monotonicity* and as a result a *hazard-free implementation*. In [Lemberski, 2009], a two-level (NOR-NOR, NAND-NAND) dual-rail asynchronous logic suitable for mapping onto the conventional two-level structure was offered. Using this result, we propose a method that is based on the initial logic function decomposition into a single-rail Boolean network, where each node is represented as a two-level logic and the network is further transformation into a dual-rail one to ensure *monotonicity* and *hazard-free* implementation. Although our approach slightly increases the complexity of the functional logic, the completion detection logic complexity reduces significantly since the number of nodes that should be supplied with the completion detection is less than in [Cortadella, 2004]. As a result, considerable improvement in the sense of the total complexity is obtained.

2. Preliminaries

2.1. *Input/Output Dual-Rail Encoding*

Let $F = \{f_1, f_2, \ldots, f_q\}$ be an asynchronous multi-output function of n inputs X: $X = \{x_1, x_2, \ldots , x_n\}$ and q outputs. Let $Y = \{y_1, y_2, \ldots, y_m\}$, $f_1, f_2, \ldots, f_q \in Y$, $m \geq q$, be a set of single output Boolean nodes obtained as a result of a decomposition. Each node function y_c depends on given r or less number on inputs: $y_c = \{z_{c1}, z_{c2} , \ldots , z_{cr}\}$, $| y_c | \leq r$, z_{c1},

$z_{c2}, \dots, z_{cr} \in \{X \cup Y\}$ and can be implemented as a two-level (AND-OR) complex gate (Fig. 1). We call it as a single-rail multi-level representation

Generally, an asynchronous logic should be capable: 1) to recognize the moment when a new input state (generated by the environment) appears on the inputs and the moment when the circuit generates a new output state in the response to the input one; 2) to notify the environment on new input and output states. After receiving the notification, the environment can generate the next input state. To solve this problem, inputs/outputs are implemented using a dual-rail encoding.

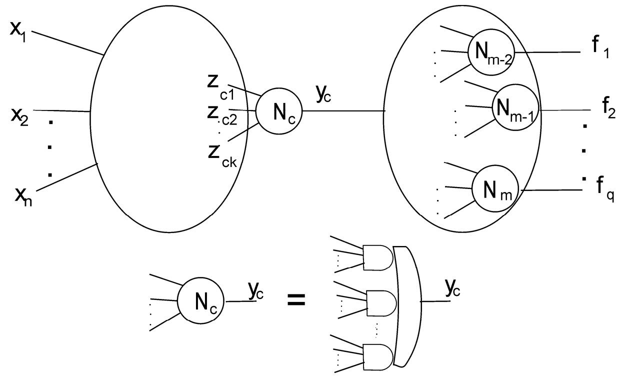

Fig. 1 Single–rail multi-level Boolean network

In dual-rail logic, it is supposed that each primary input from the set X and output node from the set Y may be in one of these three states: states 1, 0 (so called working states) or undefined (space state). To implement a three-state input x_i, $i = 1, 2, \dots, n$, two signals $x_i^{(1)}$ and $x_i^{(0)}$ are introduced, where $x_i^{(1)} = 1$ and $x_i^{(0)} = 0$, if x_i is in state 1, $x_i^{(1)} = 0$ and $x_i^{(0)} = 1$ if x_i is in state 0, $x_i^{(1)} = x_i^{(0)} = 0$ if x_i is in the space state. The combination $x_i^{(1)} = x_i^{(0)} = 1$ is not allowed. Similarly, to implement a three-state node function, the function y_c, $c = 1, 2, \dots, t$ should be represented in both positive $y_c^{(1)}$ and negative $y_c^{(0)}$ forms. If $y_c^{(1)} = 1$, $y_c^{(0)} = 0$, then the function y_c is in state 1, if $y_c^{(1)} = 0$, $y_c^{(0)} = 1$, then the function y_c is in state 0, if $y_c(1) = y_c(0) = 0$, then function y_c is in the space state. The combination $y_c(1) = y_c(0) = 1$ is not allowed. To change the input state, the environment should reset it first to the space state and after that set it to a proper working state. In the reset phase, the output state changes from the working state to the space one and in the set phase the new output state is recognized.

As a result of the decomposition, each function y_c is represented as a pair: $y_c = (y_c^{(1)}, y_c^{(0)})$, where $y_c^{(1)}, y_c^{(0)}$ describe ON-, OFF- sets ($y_c^{(0)}$ can be generated as a complement of the ON-set). After that each node (both ON- and OFF- sets) can be minimized to reduce the implementation cost. In [Lemberski, 2009], we formulated a minimization constraint that the two–level logic should satisfy, to ensure a hazard-free implementation. Namely, each function y_c should be represented as a pair of minimized Sum-of-Products (SOP) forms: $y_c = (Y_c^{(1)}, Y_c^{(0)})$, where $Y_c^{(1)}, Y_c^{(0)}$ are ON- , OFF- sets of product terms, $t_i \cap t_j = \emptyset$, for $\forall (t_i, t_j)$: $t_i, t_j \in Y_c^{(1)}$, and $t_i, t_j \in Y_c^{(0)}$. A sum of the orthogonal products is called a Disjoint-Sum-Of-Products (DSOP). In [Cortadella, 2004], conditions are formulated under which a Boolean network can be implemented as hazard-free logic. The conditions are based on each node *monotonicity and hazard-free implementation.*

3. Structure of Multi-Level Implementation Using Complex Nodes

3.1. *Monotonicity and Hazard-Free*

Our structure is based on the concept of the *monotonicity* of the nodes introduced in [Cortadella, 2004] and the condition of each two-level (AND-OR) node *hazard-free* implementation proposed in [Lemberski, 2009].

Monotonicity. A node N_c generating the function y_c is *positive* if for each input z_c in its local fan-in it holds the following: if the input z_c is positive (negative) then the function y_c is positive (negative). A node N_c generating function y_c is *negative* if for each input z_c in its local fan-in it holds the following: if the input z_c is positive (negative) then the function y_c is negative (positive). The node N_c is monotonic if it is either *positive* or *negative.*

The node monotonicity is easily achieved by the dual-rail encoding.

Hazard-free implementation. The Boolean network is hazard-free if each node is hazard-free.

The hazard-free implementation of the two-level positive dual-rail structures based on the formulated minimization condition: product terms implementing two-level AND-OR logic should be mutually orthogonal [Lemberski, 2009].

Note that in [Cortadella, 2004], the Boolean network with simple nodes (AND, OR, NAND gates, etc.) was considered. For such a network, the node monotonicity is the one requirement that guarantees its (and as a result, of the whole network) hazard-free implementation. However, it is not the case for the network with complex AND-OR nodes, where the additional condition (to ensure the node hazard-free implementation) should be formulated.

3.2. *Basic Structure for Node Implementation*

The implementation was proposed in [Lemberski, 2009] for multi-output logic (in our case, it should be reduced to a single-output one). It consists of two blocks (Fig. 2): a two-level AND-OR and the completion detection logic. Each AND gate implements a product term obtained after the minimization (remember, only the minimization that produces mutually orthogonal terms is allowed). Each product term is described by the set S(t_k), |S(t_k)| ≤ n, where S(t_k) is a set of term t_k literals (input signals), $k = 1, 2 ,…, p$. Since logic with the unbounded delays is supposed, one needs a signal to indicate the moment when both inputs and outputs are in the proper (working or space) state. For this purpose, the completion detection logic is introduced. Once all inputs and outputs are in the working state (means: either $x_i^{(1)}$ or $x_i^{(0)}$ and $f_c^{(1)}$ or $f_c^{(0)}$ are in the state 1, $i = 1, 2, … , n$, $c = 1, 2, …, q$) then the signal D is going up too. To change the input state, both inputs and outputs should go to the space state ($x_i^{(1)} = x_i^{(0)} = f_c^{(1)} = f_c^{(0)} = 0$). It results in the signal D going down. Once $D = 0$, then the new input can be set up.

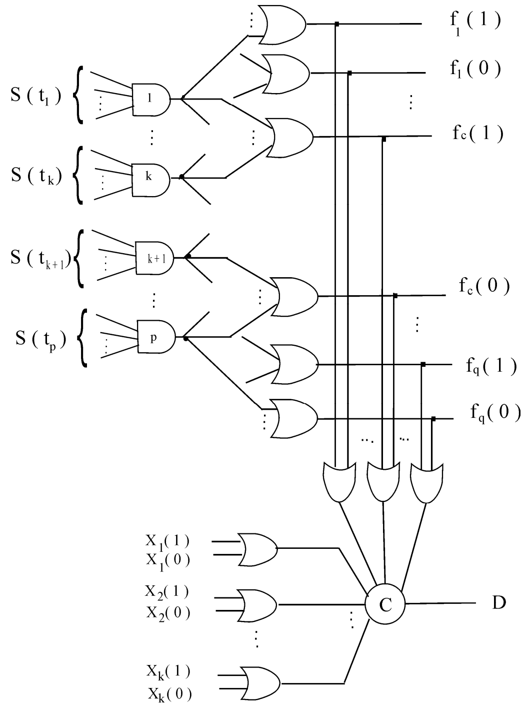

Fig. 2. Dual-rail two-level node

3.3. Multi–Level Network

Given an arbitrary multi-level Boolean network (Fig. 1). The network is transformed into the dual-rail one based on the rules described in Section 2. Then, each pair of nodes representing a function in both its positive and negative form is mapped into the structure depicted in Fig. 2. The multi-level structure consists of two blocks (Fig. 3): the functional one implemented as a multi-level logic with two-level AND-OR single-output nodes with a fan-in limited to $2k$ (remember, once given a single–rail node, then in dual-rail each input is represented as two signals) and the completion detection logic that is obtained by merging the completion detection logic of all two-level nodes. The completion detection should indicate the proper state (working or space) of not only the network primary inputs and outputs but node outputs as well. The logic is based on $(n+m)$ C-elements together with $(n+m)$ two-input OR gates, where n is number of primary inputs and m the number of nodes (including the ones generating q primary outputs). The completion detection signal D (Fig. 3) is going up, when both primary inputs and node outputs are all in a working state and going down when the signals mentioned are all in a space state.

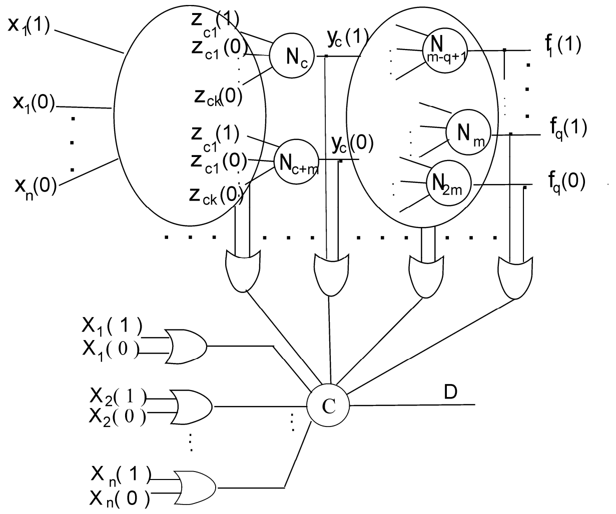

Fig. 3. Dual-rail multi-level network

4. Synthesis Procedure

The process of the synthesis of the multi-level dual-rail logic with AND-OR nodes is based on the tools ABC [Berkeley], Espresso [Brayton, 1984] and DSOP [Bernasconi, 2008]. First, an ABC script is applied to the initial circuit representation to obtain a multi-level single-rail Boolean network with the fan-in of each node limited to k. For this, we have decided to employ a LUT mapping synthesis process, since each LUT is actually represented as a single-output AND-OR node with a limited number of inputs (in the ABC output format).

We have used a sequence of ABC commands recommended for the LUT synthesis in the ABC reference guide. This command sequence was repeated 4-times, to obtain better results.

```
strash
balance
fpga –K k
```

Fig. 4. The LUT decomposition script. Substitute k for the maximum node fan-in

Then, the network is transformed into the dual–rail representation, by computing a complement of each node (using the “sharp” operator [Brayton, 1984]). As a result, the number of nodes is doubled, while now the node functions may depend on $2k$ inputs or less (since a positive and negative signal is represented as a separate rail). Next, the minimization is performed (using Espresso) for the OFF–set nodes to obtain the minimized function: $y_c = (Y_c^{(1)}, Y_c^{(0)})$, $y_c^{(1)} \subseteq Y_c^{(1)}$, $y_c^{(0)} \subseteq Y_c^{(0)}$. Finally, we run DSOP [Bernasconi, 2008] for all the nodes, to obtain mutually orthogonal terms.

5. Experimental Results

5.1. *Experimental background*

We have processed the MCNC [Yang, 1991] and ISCAS [Brglez, 1985, 1989] sets of benchmarks, 228 circuits altogether. We evaluate the complexity (expressed as the gate equivalents (GEs) number [De Micheli, 1994]) of the proposed asynchronous implementation of these circuits.

For the structure proposed, we estimate the complexity of the functional network and the completion detection logic separately. Then, the total complexity is calculated. To avoid additional inverters and therefore decrease the implementation complexity, we use negative (NAND-NAND) gates instead of AND-OR ones in the functional block and NOR gates instead of OR ones in the completion detection logic. As a result, the signal D = 1 (D = 0), when all inputs and outputs are in the space (working) state. Duplicated terms are implemented only once. We suppose a technology independent synthesis (fan-ins of negative gates and C-element are not limited). The gate complexity is estimated as follows: an *n*-input NAND or NOR gate requires 0.5*n* GEs [Sparsø, 2001]. To implement an (*n*+*m*)-input C-element, (*n*+*m*+1) GEs are required. To implement *n*+*m* two-input NOR gates, 0.5(*n*+*m*) GEs are required. Totally, (1.5*n*+1.5*m*+1) GEs are required to implement the completion detection logic for an *n*-input multi-level logic with *m* nodes.

Note, that the complexity of the sequential logic memory (flip-flops, latches) is not included in the results.

5.2. *Selection of k*

The first issue addressed in the experiments is a proper selection of *k* (maximum node fan-in). For large *k*'s, there often arise problems with computing complements of the nodes, for an exponential complexity of the operation. More importantly, nodes with a high fan-in are difficult to be implemented in technology. On the other hand, small *k*'s induce more nodes, which makes the completion detection logic more complex.

A similar problem has been encountered in the design of the FPGA fabrics, when deciding for the optimum look-up tables (LUT) size [Gao, 2005]. It has been found that implementing the design using 4- or 5- input LUTs brings most benefits. We have also reproduced this observation by performing numerous experiments. An example is shown in Fig. 5, for the *9sym* MCNC benchmark circuit. We have synthesized this circuit using the LUT-decomposition script (see Fig. 4), for *k* varying from 2 to 20. The total complexity of the asynchronous logic (i.e., the functional logic with the completion detection) was measured. A deep global minimum can be observed for *k = 4*. Very similar results are obtained from a vast majority of other benchmark circuits, for both decomposition scripts. For this reason, all the following experiments will be performed for *k* = 4.

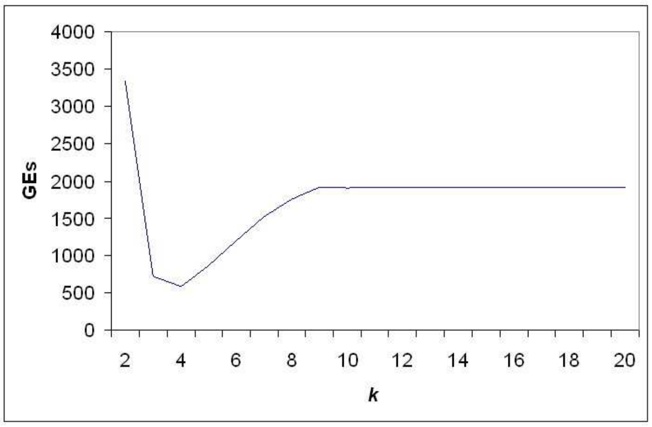

Fig. 5. Influence of *k* on the size of the resulting logic

5.3. *Standard Benchmarks Results*

Results obtained for selected MCNC [Yang, 1991] and ISCAS [Brglez, 1985, 1989] benchmark circuits are presented in the summary Table 1. We have evaluated the area overhead of our proposed asynchronous logic design method w.r.t. a conventional synchronous design. Then, we have compared our method with a state-of-the-art asynchronous logic design method proposed in [Cortadella, 2004]. In all the cases, 4-input AND-OR nodes are considered.

In Table 1, first, the benchmark name and numbers of its primary inputs and outputs (*n*, *q*) are given. Synthesis results obtained by decomposing the original circuit into a network of 4-input AND-OR nodes are shown in the following triplet of columns "*Synchronous*". The first column indicates the number of network levels (critical path), the number of decomposed circuit nodes follows, the last column shows the complexity of the circuit's synchronous implementation, in terms of GEs.

The complexity of the proposed asynchronous multi-level implementation of the circuits is shown next. Complexities of the functional logic ("*Funct. GEs*") and the completion detection logic ("*CD GEs*") are shown first, then the values are summed together to obtain the final asynchronous logic complexity ("*Total GEs*"). The area increase of the asynchronous logic w.r.t. the synchronous implementation is shown in the next column ("*Over.*").

Complexities of the asynchronous multi-level implementation proposed in [Cortadella, 2004] are shown in the next triplet of columns. Again, the functional, completion detection and total complexities are given. The area reduction obtained by our method, w.r.t. [Cortadella, 2004], is shown in the last table column (“*Impr*”).

5.4. Summary of the Experiments

We have processed 228 benchmark circuits altogether. The area overhead of the asynchronous implementation, compared to the synchronous implementation is increased by 64% in the average. When compared to the state-of-the-art approach, we have obtained an average improvement of 17%. However, for some circuits, the improvement reaches up to 40%.

6. Conclusion

A novel synthesis method of a dual-rail asynchronous multi-level logic is proposed. The logic is implemented as a monotonous multi-level network of minimized AND-OR nodes together with the completion detection logic. Each node is a hazard-free structure. It is achieved based on the product term minimization constraint (product terms must be mutually orthogonal) that the authors have formulated and proved in [Lemberski, 2009]. The MCNC and ISCAS benchmarks were processed and the complexity of the synchronous and asynchronous implementations was compared. For the asynchronous logic, the area overhead is 64% in the average. In comparison with the state-of-the-art approach, we reached a 17% area improvement in the average.

Acknowledgment

For the second author, this research has been supported by MSMT under research program MSM6840770014 and GA102/09/1668.

References

Beerel, P., Yun, K.Y., and Chou, W.C. (1996). Opimizing Average-Case Delay in Technology Mapping of Burst-Mode Circuits, IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, pp. 244-259.

Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification'. http://www.eecs.berkeley.edu/~alanmi/abc/.

Bernasconi, A., Ciriani, V., Luccio, F., and Pagli, L. (2008). A New Heuristic for DSOP Minimization, Proc. 8th Int. Workshop on Boolean Problems (IWSBP'08), Freiberg, Germany, 18.-19.9.2008, pp. 169-174.

Brayton, R.K., et al. (1984). *Logic minimization algorithms for VLSI synthesis*, Boston, MA, Kluwer Academic Publishers, 192 pp.

Brglez, F. and Fujiwara, H. (1985). A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan, Proc. of ISCAS 1985, pp. 663-698.

Brglez, F., Bryan, D., and Kozminski, K. (1989). Combinational Profiles of Sequential Benchmark Circuits, Proc. of ISCAS, pp. 1929-1934.

Cortadella, J., Kondratyev, A., Lavagno, L., and Sotiriou, C. (2004). Coping with the Variability of Combinational Logic Delays, IEEE Int. Conf. On Computer Design, pp. 505-508.

De Micheli, G. (1994). *Synthesis and Optimization of Digital Circuits*. McGraw-Hill.

Gao, H., Yang, Y., Ma, X., and Dong, G. (2005). Analysis of the effect of LUT size on FPGA area and delay using theoretical derivations, Proc. of the Sixth International Symposium on Quality of Electronic Design”, 21.-23. 3, pp. 370-374.

Kung, D. (1992). Hazard-Non-Increasing Gate–Level Optimization Algorithm, IEEE Int. Conf. On Computer–Aided Design, pp. 631-634.

Lemberski, I. and Fišer, P. (2009). Asynchronous Two-Level Logic of Reduced Cost, IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, April 15-17, 2009, Liberec, Czech Republic, pp. 68-73.

Ligthart, M., Fant, K., Smith, R., Taubin, A., and Kondratyev, A. (2000). Asynchronous Design Using Commercial HDL Synthesis Tools, 6-th Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, pp. 114-125.

Nowick, S.M. (1993). Automatic Synthesis of Burst-Mode Asynchronous Controllers, Ph.D. thesis, Stanfort University, March 193.

Nowick, S.M. and Dill, D.L. (1995). Exact Two-Level Minimization of Hazard-Free Logic with Multiple-Input Changes, *IEEE CAD*, vol. 14, August 1995, pp. 986-997.

Siegel, P., Micheli, G.D., and Dill, D. (1993). Automatic Technology Mapping for Generalized Fundamental Mode Asynchronous Designs, IEEE Design Automation Conference, pp. 61-67.

Sparsø, E.J., Staunstrup, J., and M. Dantzer-Sørensen (1992) Design of delay insensitive circuits using multi-ring structures, In Prc. of the Conference on European Design Automation, pp. 15-20.

Sparsø, E.J. and Furber, S. (2001). *Principles of Asynchronous Circuit Design*, Kluwer Academic Publishers, 337 p.

Unger, S.H. (1969). *Asynchronous Sequential Switching Circuits*, John Wiley & Sons, Inc.

Yang, S. (1991). Logic Synthesis and Optimization Benchmarks User Guide, Technical Report 1991-IWLS-UG-Saeyang, MCNC, Research Triangle Park, NC.

Table 1. Comparison results

Benchmark circuit			***Synchronous***			***Proposed asynchronous***				*Cortadella, 2004*			
Name	***n***	***q***	***Lev.***	***Nodes***	***GEs***	***Funct. GEs***	***CD GEs***	***Total GEs***	***Over.***	***Funct. GEs***	***CD GEs***	***Total GEs***	***Impr.***
al2	16	47	2	62	140.5	297	118	415	66%	281	170.5	451.5	8%
alcom	15	38	2	50	87.5	179	98.5	277.5	68%	175	107.5	282.5	2%
alu1	12	8	1	8	27.5	62.5	31	93.5	71%	55	58	113	17%
b2	16	17	7	643	2038.5	4348	989.5	5337.5	62%	4077	2536	6613	19%
b9	41	21	3	49	130	273	136	409	68%	260	226	486	16%
bc0	26	11	7	510	1683.5	3534.5	805	4339.5	61%	3367	2105.5	5472.5	21%
c1355	41	32	4	74	751	1595	173.5	1768.5	58%	1502	674.5	2176.5	19%
c2670	233	140	7	300	936	1831	659.5	2490.5	62%	1872	1346.5	3218.5	23%
c7552	207	108	8	600	3110.5	6329.5	1142.5	7472	58%	6221	3443.5	9664.5	23%
c8	28	18	3	69	211	419.5	145	564.5	63%	422	313	735	23%
c880	60	26	8	122	562.5	1186.5	274	1460.5	61%	1125	700	1825	20%
cc	21	20	2	25	51	105.5	61	166.5	69%	102	89.5	191.5	13%
chkn	29	7	8	158	497	1068	281.5	1349.5	63%	994	640	1634	17%
cht	47	36	3	46	165.5	358	140.5	498.5	67%	331	262	593	16%
cordic	23	2	8	614	1801	3926.5	956.5	4883	63%	3602	2254	5856	17%
count	35	16	4	46	169	388	122.5	510.5	67%	338	263.5	601.5	15%
cps	24	109	5	701	1676.5	3628	1076.5	4704.5	64%	3353	1984	5337	12%
cu	14	11	3	20	53.5	110.5	52	162.5	67%	107	80.5	187.5	13%
dalu	75	16	11	466	1971	4226.5	812.5	5039	61%	3942	2437	6379	21%
dc1	4	7	1	7	48	73	17.5	90.5	47%	96	55	151	40%
dc2	8	7	4	36	130.5	275	65.5	340.5	62%	261	173.5	434.5	22%
duke2	22	29	5	238	478	983	391	1374	65%	956	524.5	1480.5	7%
e64	65	65	4	332	559.5	1119	595	1714	67%	1119	595	1714	0%
ex4	128	28	6	206	673.5	1501	481	1982	66%	1347	1060	2407	18%
ex5p	8	63	6	1132	2487	5219.5	1711	6930.5	64%	4974	2804.5	7778.5	11%
ex7	16	5	4	38	147.5	327	82	409	64%	295	197.5	492.5	17%
example2	85	66	4	132	395.5	893.5	326.5	1220	68%	791	625	1416	14%
f51m	8	8	4	80	233	486	131.5	617.5	62%	466	292	758	19%
frg1	28	3	6	209	485.5	1020.5	356.5	1377	65%	971	560.5	1531.5	10%
frg2	143	139	6	524	1339	3084	986.5	4070.5	67%	2678	1876	4554	11%
i10	257	224	12	871	3034.5	6335.5	1676.5	8012	62%	6069	3842.5	9911.5	19%
i2	201	1	5	75	171.5	356.5	415	771.5	78%	343	455.5	798.5	3%
i3	132	6	3	46	214	492	268	760	72%	428	460	888	14%
ibm	48	17	5	87	275	649.5	203.5	853	68%	550	434.5	984.5	13%
misex2	25	18	3	45	86.5	215	106	321	73%	173	139	312	-3%
s1196	32	32	7	220	672.5	1467.5	376	1843.5	64%	1345	868	2213	17%
s15850.1	611	684	13	1293	3997.5	8498	2609.5	11107.5	64%	7995	5500	13495	18%
s35932	1763	2048	4	3200	10658	22676	7013.5	29689.5	64%	21316	14150.5	35466.5	16%
s382	24	27	3	55	165	359	110.5	469.5	65%	330	229	559	16%
s38417	1664	1742	9	3610	12308	26197	7144	33341	63%	24616	17528.5	42144.5	21%
s9234.1	247	250	8	681	2202.5	4664	1262.5	5926.5	63%	4405	2933.5	7338.5	19%
s953	45	52	4	197	447.5	950.5	329.5	1280	65%	895	590.5	1485.5	14%
too_large	38	3	7	2707	6017	12586.5	4118.5	16705	64%	12034	6151	18185	8%
ts10	22	16	3	48	256	528	106	634	60%	512	346	858	26%
ttt2	24	21	5	152	404	929.5	265	1194.5	66%	808	550	1358	12%
unreg	36	16	2	48	136	305	127	432	69%	272	247	519	17%
vda	17	39	5	413	835	1637.5	646	2283.5	63%	1670	887.5	2557.5	11%
x2dn	82	56	4	95	218	438	236.5	674.5	68%	436	371.5	807.5	16%
x3	135	99	5	333	1073.5	2299	703	3002	64%	2147	1550.5	3697.5	19%
x4	94	71	5	210	542	1232.5	448	1680.5	68%	1084	830.5	1914.5	12%
x6dn	39	5	6	172	552.5	1204.5	317.5	1522	64%	1105	731.5	1836.5	17%
xparc	41	73	12	1530	4804.5	9986	2351.5	12337.5	61%	9609	5777.5	15386.5	20%
Total									64%				17%