Cost-aware synthesis of asynchronous circuits based on partial acknowledgement

Yu Zhou, Danil Sokolov and Alex Yakovlev
School of Electrical, Electronic and Computer Engineering, University of Newcastle upon Tyne
e-mail: {yu.zhou, danil.sokolov, alex.yakovlev}@ncl.ac.uk

ABSTRACT

Designing asynchronous circuits by reusing existing synchronous tools has become a promising solution to the problem of poor CAD support in asynchronous world. A straightforward way is to structurally map the gates in a synchronous netlist to their functionally equivalent modules which use delay-insensitive codes. Different trade-offs exist in previous methods between the overheads of the implementations and their robustness. The aim of this paper is to optimise the area of asynchronous circuits using partial acknowledgement concept. We employ this concept in two design flows, which are implemented in a software tool to evaluate the efficiency of the method. The benchmark results show the average reduction in area by 28% and in the number of inter-functional module wires that require timing verification by 67%, compared to NCL-X.

1. INTRODUCTION

In the past few years, designers were trying to build a bridge between synchronous and asynchronous circuits. Rather than synthesising the circuits from "truly" asynchronous specifications such as communicating processes [9] or signal transition graph [2, 14], asynchronous circuits are built at a low cost by reusing synchronous CAD tools in their design flows. Asynchrony, in return, makes the circuits more robust in coping with the variations in the fabrication process, which is an increasingly important concern to digital circuit designers.

The well-known approaches to asynchronous circuit synthesis are NCL-D [8] and NCL-X [6]. Both of them map a gate in the synchronous circuit to a dual-rail encoded module with an equivalent function. NCL-D replaces a gate with one of the NCL (Null Convention Logic) operators based on threshold logic [11]. A further optimisation of such a NCL network is introduced in [5]. Besides the threshold logic, other techniques exist to implement the functional modules used in a NCL-D circuit. They include DIMS (Delay Insensitive Minterm Synthesis) [20], DL (Direct Logic) [21] and RDL (Reduced Direct Logic) [13]. NCL-D approach is very robust because all the inter-module wires are allowed to have arbitrary delays without damaging the circuit's correct behaviour. Critical forks exist only within a functional module whose timing closure can be ensured through careful routing of the library elements. Whilst robust, it suffers from large area and slow speed. NCL-X, on the other hand, uses simpler and faster functional modules for the functional part of a circuit. However, it needs extra *completion detection* (CD) circuitry to tell when the computations are done.

This paper proposes two design flows to synthesise asynchronous circuits from synchronous ones. We formulate the first design flow (DF1) as a unate covering problem and its solution decides which type of functional module replaces a gate in the synchronous netlist, NCL-D or NCL-X. More "fine-grained" functional modules are added to the library of the second design flow (DF2) and we formulate it as a binate covering problem. We define the partial acknowledgement to guide the synthesis procedure. Intuitively, the covering problems determine which type of functional module is chosen to replace a gate in the original circuit so that all the circuit variables are "covered" by the meaning of partial acknowledgement. Solutions to these covering problems are then found to minimise the cost functions that are defined in terms of a circuit's area in this paper. DF1 reduces the circuit's area compared with NCL-D but increases the number of inter-module wires to be verified for timing closure. The DF2 further reduces the overhead of the first one with the penalty of a 50% increase in verification demand. Compared with NCL-X solutions, DF2 has reduced both the circuit area (by 28% on average) and the verification demand (by 67% on average).

Our work is also motivated by the optimisation of an asynchronous circuit's performance. We believe a circuit can be synthesised to keep its robustness level to some extent without hurdling the data-dependent flows. For this purpose, we will apply the idea of partial acknowledgement during the static timing analysis of a circuit. The functional modules synthesised in DF2 are the candidates for this future work.

ICCAD'06, November 5-9, 2006, San Jose, CA

2. BACKGROUND

2.1 Functional modules using dual-rail code

For each bit of a signal, the dual-rail encoding under the return-to-zero protocol (*a.k.a.* 4-phased protocol) represents null by $rail^0 = rail^1 = 0$ and valid code word 1(0) by $rail^0 = 0$, $rail^1 = 1$ ($rail^0 = 1$, $rail^1 = 0$). In a circuit using this convention, data and null wavefronts alternate (the circuit signals

switch from *nulls* to valid code words and from valid code words to nulls, respectively).

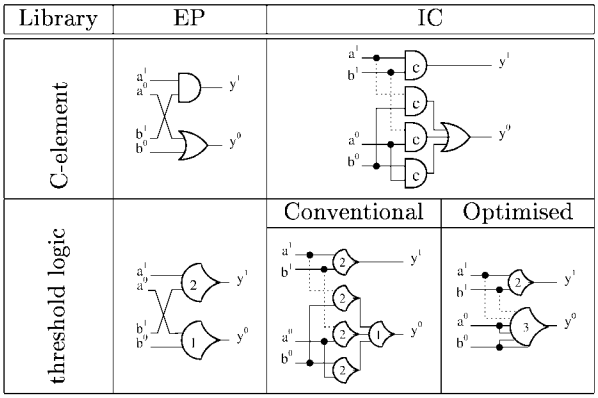


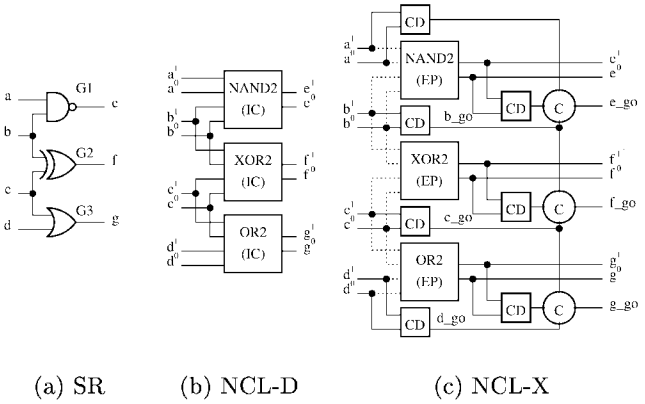
Table 1: Different implementations of an AND gate

Gates in a single-rail netlist are converted to their corresponding functional modules, the dual-rail encoded entities with the equivalent functions. We categorise the functional modules into two classes by their impact on the data flowing through them. Table 1 illustrates the classification of the functional modules implementing an AND gate. The first class can produce a valid (null) output when only partial inputs are valid (null) and hence called *early propagative* (EP). For example, a code word of 0 (y^0=1, y^1 =0) is produced at the output of an EP AND functional module in Table 1 when only one input turns into 0 (a^0=1, a^1 =0) in a data wavefront. The second class must wait for all the inputs becoming valid (null) before producing a valid (null) output. It is called *input complete* (IC) according to the definition of completeness of inputs [4]. Table 1 demonstrates the implementation of an IC AND gate based on Muller-C elements (DIMS) and on threshold logic (the 2NCL operator). Optimisation using sum-partitioning and product-partitioning is available in the threshold logic implementation [18, 11, 17].

2.2 Asynchronous circuits implementations and their timing assumptions

Asynchronous circuits rely on the dependencies between signals, rather than the global clock, to tell when the computation is done. The monotonic transitions from nulls (valid code words) to valid code words (nulls) provide one scheme where the dependencies are sought in the two phases separately. Furthermore, signalling schemes are proposed for a circuit's behaviour: the "weak conditions" by Seitz [16] and the "completeness of inputs" [4] by NCL.

NCL-D [8, 20] builds an asynchronous circuit by replacing the gates in a synchronous one with their corresponding IC functional modules. Figure 1(b) illustrates how this method implements the circuit in Figure 1(a). Satisfying the weak conditions and the "completeness of inputs" requirement, it is very robust except for some "small" timing assumptions required for some forks in an IC module, formally known as the "isochronic fork assumptions" [1]. We show these forks with dotted wires in Table 1 when inputs a and b change from *nulls* to "1"s in a valid data wavefront. On the other hand, NCL-X replaces the gates with their EP functional modules for computation and demands



(a) SR (b) NCL-D (c) NCL-X

Figure 1: Timing assumptions in different implementation methods

extra completion detection (CD) circuitry for the "completeness of inputs". Figure 1(c) illustrates the implementation of NCL-X, where the CDs are simply the OR gates whose outputs are synchronised by the C-element trees. In NCL-X, the timing assumption imposed on the inter functional module wires (the dotted ones in Figure 1(c)) is to bound their delays by the stabilisation time of the CD circuitry.

3. PARTIAL ACKNOWLEDGEMENT

Suppose v_i is a dual-rail encoded variable in circuit C fanning into F, a functional module whose dual-rail output is v_j. Let C work under the return-to-zero protocol. We have $v_i\uparrow$ denoting the rising phase transition of v_i from null to a valid code word and $v_i\downarrow$ the falling phase transition of v_i from the valid code word to null. A valid code word represents either 0 or 1.

We define that the rising phase transition of v_i is partially acknowledged by that of v_j, iff:

$$(\forall v_i\uparrow\colon \exists v_j\uparrow\colon v_i\uparrow \rightarrow v_j\uparrow) \tag{3.1}$$

Similarly, we define that the falling phase transition of v_i is partially acknowledged by that of v_j, iff:

$$(\forall v_i\downarrow\colon \exists v_j\downarrow\colon v_i\downarrow \rightarrow v_j\downarrow) \tag{3.2}$$

The following proposition can be proved by contradiction. If the rising phase transition of v_i is partially acknowledged by that of v_j, transition of v_j from null to valid code word indicates the transition of v_i from *null* to valid code word. The proposition can be derived for the falling phase transitions of v_i and v_j, too. The acknowledgement is partial because we don't know if the same relations as those defined in (3.1) and (3.2) exist between v_i and other functional modules' outputs.

v_i is partially acknowledged if both its rising and falling phase transitions are partially acknowledged. In particular, v_i is partially acknowledged by v_j, iff

$$\begin{array}{l}(\forall v_i\uparrow\colon \exists v_j\uparrow\colon v_i\uparrow \rightarrow v_j\uparrow)\wedge\\ (\forall v_i\downarrow\colon \exists v_j\downarrow\colon v_i\downarrow \rightarrow v_j\downarrow)\end{array} \tag{3.3}$$

According to our definitions, the output of an IC functional module partially acknowledges all its inputs. On the contrary, the output of an EP functional module partially acknowledges none of its inputs. In NCL-D, an input (internal) variable is partially acknowledged by all the outputs of the functional modules it fans into. In NCL-X, however, a variable is only partially acknowledged by the outputs of

the completion detection circuitry.

4. DESIGNS FLOWS BASED ON PARTIAL ACKNOWLEDGEMENT

Our two design flows have the same objective, which is described below.

Design objective: Given the synchronous single-rail netlist, implement its equivalent circuit where every gate in the original netlist is replaced by the appropriate type of dual-rail encoded functional module. Find a solution with the *minimum area* under the requirement that each input and internal variable in the final implementation is at least partially acknowledged.

The first design flow (DF1) contains only IC and EP functional modules and is formulated as a unate covering problem. The second design flow (DF2) is supplemented with functional modules partially acknowledging particular phase transitions of certain inputs and is defined as a binate covering problem. Unate (and binate) covering problems (also referred as *mincost sat* problems [12]) are discussed in detail in [3]. A unate covering problem has its well known application in 2-level circuit minimisation and binate covering problem in technology mapping [15]. We choose the constraint equation forms of the covering problems for our presentation, though the matrix forms work as well.

4.1 Design flow 1

Revisit the example of Figure 1(a). This time we replace gate *G1* and *G3* with their IC implementations whereas *G2* its EP one. Figure 2 illustrates the decision. According to our definitions, *a* and *d* are *acknowledged* while *b* and *c* are *partially acknowledged* by *G1* and *G3*, respectively. It satisfies the requirement of the design objective with 60 transistors by RDL. As a comparison, NCL-X uses 110 transistors plus extra interconnections in the CD circuitry. In terms of robustness, the amount of inter-module wires required for timing verification is greatly reduced (4 vs 12).

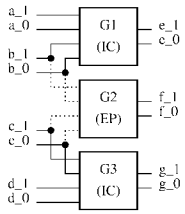


Figure 2: Intuitive implementation of the exemplar circuit

4.1.1 Formulation of DF1 as a unate covering problem

Let v_i be an input or internal variable and G_i be a gate belonging to circuit C. A variable in DF1 can only be partially acknowledged by the IC functional modules it fans into. Let boolean variable G_i^{ic} denote the implementation of G_i as an IC functional module. A clause is written for each variable of the circuit indicating which IC functional modules can partially acknowledge it. For example, the clause for variable v_i can be written as $c_{v_i} = \sum_{G_j \in direct-fan-out(v_i)} G_j^{ic}$. Suppose w_i is the cost of the implementation of G_i as an IC functional module, the design objective is formulated as a unate covering problem:

Find the assignment to the boolean variable G_i^{ic} of every gate in the circuit that satisfies the constraint equation:

$$\Pi_{v_j \in \{inputs\ and\ internal\ variables\ of\ C\}} c_{v_j} = 1, \quad (4.1)$$

while minimising the cost function $\sum_i (w_i \bullet G_i^{ic})$. Satisfaction of 4.1 ensures each input and internal variable in the circuit is partially acknowledged. After finding one satisfying assignment, implement a gate G_i by its IC functional module if $G_i^{ic} = 1$ or otherwise by its EP functional module.

Example. In the circuit shown in Figure 1(a), the clauses of inputs *a, b, c* and *d* are G_1^{ic}, $G_1^{ic} + G_2^{ic}$, $G_2^{ic} + G_3^{ic}$ and G_3^{ic}, respectively. Therefore, the constraint equation of the circuit can be expressed as $G_1^{ic}(G_1^{ic}+G_2^{ic})(G_2^{ic}+G_3^{ic})G_3^{ic} = 1$. The solution is obvious: $G_1^{ic} = G_3^{ic} = 1$. As a result, $G_1$and G_3 are implemented as IC functional modules while G_2 as EP.

4.1.2 Timing assumptions

In the final implementation, a fork is non-critical if all the gates it fans into are implemented as their IC functional modules. For the fork wires fanning into EP functional modules, we need to ensure their timing closure to avoid a potential hazard.

4.2 Design flow 2

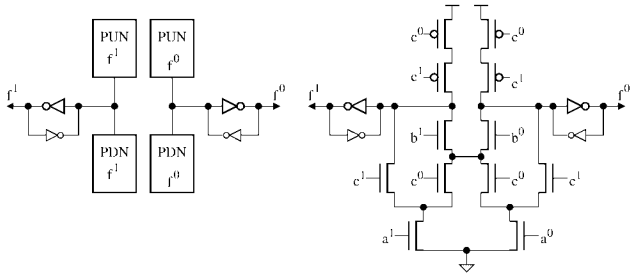
DF1 binds all the inputs of an IC module for their partial acknowledgement because a gate is casted either as an IC module or EP one. By enlarging the spectrum of the functional modules in DF2, we expect more optimisation space for both the area and speed. We first introduce how to synthesise a functional module that partially acknowledges the rising or(and) falling phase transitions of certain inputs. Then we formulate the design procedure satisfying the design objective as a binate covering problem.

In literature, Martin's adder [10] is a well-known example to distribute the evaluation of the inputs' *nulls* and valid words among those of the circuit's outputs. In [13], Nielsen proposed RDL for similar distributions. In this paper, we resort to Boole's expansion theorem in its dual-rail form.

4.2.1 Functional modules synthesis

The naming convention for a functional module used in DF2 is *a module's function_(input name followed by its phase transition to be partially acknowledged)**. A phase transition could be one of the following: rising phase transition (↑), falling phase transition (↓), or both rising and falling phase transition (⋆). For example, **AND3_a↑** denotes the 3-input-AND module that partially acknowledges the rising phase transition of input *a*. Another example, **MAJ3_a⋆b⋆**, represents a 3-input-majority-voting module where both the rising and falling phase transitions of *a* and *b* are partially acknowledged by the output variable. We know that the EP and IC functional modules in DF1 can be viewed as special instances of this enlarged library.

The implementations of the functional modules used in this paper is by, but not restricted in general to, pseudo-static logics. Figure 3(a) illustrates its prototype comprising the *pull-up-network* (PUN), *pull-down-network* (PDN), and the output inverter pairs. The weak feedback inverter is used to fight against the charge leakage problem, an alternative solution to which is the cross-coupled p-typed transistors used in the DCVSL style [7]. The rest of this section is



(a) Functional module (b) Multiplexer

Figure 3: Pseudo-static logic implementations

devoted to the design procedures of a functional module in 4 steps, through the example of **MUX2:1_a↑c⋆**.

Step 1: Derive the dual-rail functions, i.e., f^0 and f^1, of the functional module from its single-rail function f. f^1 is converted from f by replacing the uncomplemented variables in f with the corresponding *rail-1s* whereas the complemented ones with *rail-0s*. f^0 is simply the dual of f^1. E.g., the boolean function of module **MUX2:1_a↑c⋆** is $f = ca + c'b$, whose dual-rail boolean functions can be derived as $f^1 = c^1a^1 + c^0b^1$ and $f^0 = c^1a^0 + c^0b^0$.

Step 2: Decompose f^1 and f^0 w.r.t. the *dual-rail minterms* of the inputs whose rising phase transitions are chosen to be partially acknowledged, according to Boole's Expansion Theorem. If there are no such inputs, f^1 and f^0 remain as they are in step 1. *Dual-rail minterms* are the dual-rail equivalence of the minterms expressed in single-rail form. For instance, the *dual-rail minterms* in the example are a^0c^0, a^0c^1, a^1c^0 and a^1c^1. The decompositions are shown in 4.1.

$$f^1 = \sum_{i=0}^{2^n-1} m_i \cdot f^1_{m_i=1}, \quad f^0 = \sum_{i=0}^{2^n-1} m_i \cdot f^0_{m_i=1} \tag{4.1}$$

In 4.1, n is the number of the inputs whose rising phase transitions are chosen to be partially acknowledged. m_0,..., m_{2^n-1} are the $2^n - 1$ *dual-rail minterms*. $f^1_{m_i=1}$ ($f^0_{m_i=1}$) is the positive cofactor of f^1 (f^0) with respect to m_i.

The decomposition process of the example is listed in 4.2.

$$\begin{aligned} f^1 &= a^0c^0 \cdot f^1_{a^0=c^0=1} + a^0c^1 \cdot f^1_{a^0=c^1=1} \\ &+ a^1c^0 \cdot f^1_{a^1=c^0=1} + a^1c^1 \cdot f^1_{a^1=c^1=1} \\ &= a^0c^0 \cdot (b^1) + a^0c^1 \cdot (0) \\ &\quad + a^1c^0 \cdot (b^1) + a^1c^1 \cdot (1) \\ f^0 &= a^0c^0 \cdot f^0_{a^0=c^0=1} + a^0c^1 \cdot f^0_{a^0=c^1=1} \\ &+ a^1c^0 \cdot f^0_{a^1=c^0=1} + a^1c^1 \cdot f^0_{a^1=c^1=1} \\ &= a^0c^0 \cdot (b^0) + a^0c^1 \cdot (1) \\ &\quad + a^1c^0 \cdot (b^0) + a^1c^1 \cdot (0) \end{aligned} \tag{4.2}$$

Step 3: Build the PDN of the functional module by connecting the n-typed transistors due to 4.1. The connection rules follow those for the NMOS network and transistors are shared to save the area. The rising phase transitions of the chosen inputs are partially acknowledged through this connection as the series connection of a *dual-rail minterm* exists in any path from ground to the module's output rails, and, only one of the minterms will be asserted during a valid code word wavefront.

Step 4: Design the PUN of a functional module. If the falling phase transition of an input is to be partially acknowledged, cascade two p-typed transistors controlled by the input's dual rails in the paths from VDD to both f^1 and f^0 [13]. If no inputs of a functional module are partially acknowledged for their falling phase transitions, we make its PUN complementary to the PDN and thus remove the feedback inverters. Figure 3(b) shows the final implementation of the module **MUX2:1_a↑c⋆**.

4.2.2 *Determination of the design library*

Our synthesis method allows separation of the partial acknowledgements of the rising and falling phase transition for a circuit variable. However, the size of the library supporting this function and the complexity of the relevant covering problem are not practical. In fact, whether we choose to partially acknowledge the falling phase transition of a variable or not - this will not influence the area optimisation because its cost is a constant (4 p-typed transistors per circuit variable). Therefore, in this paper we require that a functional module will partially acknowledge both the rising and falling phase transitions of a variable, if it does acknowledge any phase transition of that variable. There is another consideration behind this requirement to exclude any possible "on" path between the power rails.

We list the costs (transistor counts) of some functional modules in Table 2. The EP column estimates the costs EP functional modules that are based on complementary CMOS. Decomposition is applied to the functional modules with large fan-ins.

4.2.3 *Formulation of DF2 as a binate covering problem*

An input or internal variable v_i can be partially acknowledged by any functional module it fans into containing a $v_i\star$ in the suffix. Table 3 lists the possible functional modules in Figure 1(a) to partially acknowledge the circuits input variables, where a Boolean variable m_i denotes the selection of the corresponding module in the final implementation.

The clause c_{v_i} for a circuit variable v_i to be partially acknowledged is the sum of all the Boolean variables covering it. For example, $c_a = m_1 + m_3$ and $c_b = m_2 + m_3 + m_4 + m_6$. The constraint equation to ensure that all the inputs and internal variables are partially acknowledged becomes the multiplication of the clauses corresponding to these circuit variables. For our example we have:

$(m_1 + m_3)(m_2 + m_3 + m_4 + m_6)(m_5 + m_6 + m_7 + m_9)(m_8 + m_9) = 1$.

However there are extra constraints in DF2. It is clear that a gate cannot be casted as different types of functional modules in one implementation. Therefore, no boolean variables can be simultaneously assigned to 1 whose corresponding functional modules are mapped from the same gate. For example, m_1, m_2 and m_3, the boolean variables corresponding to the functional modules mapped from G_1, are mutually exclusive. Applying the DeMorgan theorem to the boolean expression $\overline{(m_1m_2) + (m_2m_3) + (m_1m_3)}$, we have the sum-of-product equivalence $(\overline{m_1} + \overline{m_2})(\overline{m_2} + \overline{m_3})(\overline{m_1} + \overline{m_3})$. The additional clauses for the constraint equation of DF2 are derived for each gate in the circuit in a similar manner. The additional clauses generated for our example are:

$(\overline{m_1} + \overline{m_2})(\overline{m_2} + \overline{m_3})(\overline{m_1} + \overline{m_3})(\overline{m_4} + \overline{m_5})(\overline{m_5} + \overline{m_6})(\overline{m_4} + \overline{m_6})(\overline{m_7} + \overline{m_8})(\overline{m_8} + \overline{m_9})(\overline{m_7} + \overline{m_9})$.

The overall constraint equation of the example is:

$(m_1 + m_3)(m_2 + m_3 + m_4 + m_6)(m_5 + m_6 + m_7 + m_9)(m_8 +$

functional module	EP	_a⋆(_b⋆)	_c⋆	_a⋆b⋆	_a⋆c⋆(_b⋆c⋆)	_a⋆b⋆c⋆
		PUN, PDN, total	PUN, PDN, total	PUN, PDN, total	PUN, PDN, total	PUN, PDN, total
AND(OR)2	12	3, 4, 15	-, -, -	6, 6, 20	-, -, -	-, -, -
XOR2	20	4, 6, 18	-, -, -	8, 6, 22	-, -, -	-, -, -
AND(OR)3	16	3, 6, 17	3, 6, 17	6, 8, 22	6, 8, 22	9, 10, 27
AO21(OA21)	16	4, 8, 20	3, 6, 17	8, 8, 24	7, 8, 23	11, 10, 29
MAJ3	28	4, 8, 20	4, 8, 20	8, 8, 24	8, 8, 24	12, 12, 32
XOR3	40	4, 10, 22	4, 10, 22	8, 10, 26	8, 10, 26	12, 10, 30
MUX2:1	24	4, 10, 22	4, 6, 18	8, 10, 26	8, 8, 24	12, 14, 34

Table 2: Transistor counts of the example library used in DF2

$m_9)(\overline{m_1}+\overline{m_2})(\overline{m_2}+\overline{m_3})(\overline{m_1}+\overline{m_3})(\overline{m_4}+\overline{m_5})(\overline{m_5}+\overline{m_6})(\overline{m_4}+\overline{m_6})(\overline{m_7}+\overline{m_8})(\overline{m_8}+\overline{m_9})(\overline{m_7}+\overline{m_9}) = 1$.

The solution to DF2 is to find the assignment to the Boolean variables m_i that satisfies the constraint equation while minimising the cost function of $\sum(m_i \bullet cost(m_i))$. It is a binate covering problem in that the Boolean variables appear in both the complemented and uncomplemented literals. The solution with the minimal cost to our example is: $m_1 = m_6 = m_8 = 1$.

For every Boolean variable assigned the value 1, its functional module is used to implement the corresponding gate. If none of the Boolean variables of a gate's implementations is assigned the value 1, it is implemented as its EP functional module. *G1, G2* and *G3* in our example are implemented as *NAND2_a⋆*, *XOR2_b⋆c⋆* and *OR2_d⋆*, respectively. This implementation uses 52 transistors.

	type of functional module	inputs	cost	covers
m_1	NAND2_$a\star$	a, b	15	a
m_2	NAND2_$a\star$	a, b	15	b
m_3	NAND2_$a \star b\star$	a, b	20	a, b
m_4	XOR2_$a\star$	b, c	18	b
m_5	XOR2_$a\star$	b, c	18	c
m_6	XOR2_$a \star b\star$	b, c	22	b, c
m_7	OR2_$a\star$	c, d	15	c
m_8	OR2_$a\star$	c, d	15	d
m_9	OR2_$a \star b\star$	c, d	20	c, d

Table 3: Covering table for circuit in Figure 1(a)

4.2.4 *Timing assumptions*

Both critical and non-critical forks exist in the final implementation. It is very easy to locate the non-critical fork wires as they are the wires fanning in to the functional modules that do not *partially acknowledge* them, i.e., to the modules without the ⋆ notation in their corresponding input ports.

5. EXPERIMENTAL RESULTS

Indie (http://async.org.uk/screen/indie) is the automatic tool implemented to test the results of the two design flows. We choose a wide range of benchmarks including several combinational circuits in ISCAS85 and different S-boxes circuits for the advanced encryption standard (AES). The results are presented in two main categories: area (the transistor counts), see Table 4, and verification demand (the number of inter-module wires to be examined for the timing closure), see Table 5.

DF1 is tested using three different techniques to implement an IC functional module : DIMS, RDL and NCL. The "orig." column shows the circuits' area of NCL-D implementation using one of the three techniques. The EP functional modules used in DIMS and RDL are the complementary static CMOS logic. The transistor count of NCL functional modules is according to the pseudo-static implementations of the 2NCL operators [19]. The "opt." column shows the circuits' area and the reduction by DF1. The average reductions of the circuit when using DIMS, RDL and NCL are 25%, 15% and 15%, respectively. The table also shows the verification demand in DF1 by the number of inter-module wires that require timing assumptions, which is null in NCL-D.

Design flow 2 further reduces the circuit's areas of DF1 but increases the verification demand by an average of 50%. Compared with NCL-X circuits, DF2 reduces the area by an average of 28% and the verification demand by an average of 67%. The CD circuitry used in our estimation of NCL-X is optimised for its area. The verification demand for NCL-X could be less if the CD circuitry is placed locally in the EP functional modules when we don't apply this optimisation.

Circuit	NCL-X	DF1	DF2
clipper	39	10 (26%)	17 (44%)
comparator_4bit	67	15 (22%)	21 (21%)
74181	114	20 (18%)	37 (32%)
74182	30	10 (33%)	12 (40%)
c432	293	27 (10%)	60 (20%)
c499	435	98 (23%)	116 (27%)
c1908	593	117 (20%)	134 (23%)
sbox_no_pipe	455	81 (18%)	119 (26%)
sbox_kasumi	633	135 (21%)	134 (21%)
sbox_unbalanced	1333	383 (29%)	673 (50%)
Average		**22%**	**33%**

Table 5: Verification demand

6. CONCLUSION AND FUTURE WORK

In this paper we propose two design flows that explore the possibility to design asynchronous circuits where the reliability of the circuit is introduced by the definition of partial acknowledgement. The designs are directed in a cost-aware manner to optimise the area of final implementations. The improvements over previous methods are proved through a set of benchmarks. In DF2, we also introduced the general method to synthesise a functional module that can partially acknowledge arbitrary combinations of input transi-

circuit	DF1						DF2	
	DIMS		RDL		NCL		NCL-X	
	original	optimised	original	optimised	original	optimised	original	optitised
clipper	760	700 (92%)	290	277 (95%)	440	416 (95%)	290	208 (72%)
comparator_4bit	1330	954 (72%)	590	474 (80%)	770	602 (78%)	780	434 (56%)
74181	2432	1986 (82%)	1059	985 (93%)	1336	1200 (90%)	1216	813 (67%)
74182	570	420 (74%)	221	185 (84%)	330	262 (79%)	286	166 (58%)
c432	6726	6166 (92%)	2798	2624 (94%)	3894	3706 (95%)	2664	2293 (86%)
c499	8094	6462 (80%)	4171	3806 (91%)	4930	4508 (91%)	5392	3493 (65%)
c1908	11828	9204 (78%)	5737	4997 (87%)	7046	6078 (95%)	6682	4186 (63%)
sbox_no_pipe	9804	7936 (81%)	4663	4211 (90%)	5498	5060 (92%)	5760	3607 (63%)
sbox_kasumi	11020	7744 (70%)	5800	4792 (83%)	6380	5372 (84%)	6850	4603 (67%)
sbox_unbalanced	35720	24476 (69%)	15071	11858 (79%)	20592	16328 (79%)	10586	9209 (87%)
Average		75%		85%		85%		72%

Table 4: Area (transistor count)

tions. It builds the framework of our future research to combine timing analysis with optimisations that will maximise the circuit's performance in addition to the quantifications of isochronic forks.

7. REFERENCES

[1] K. v. Berkel. Beware the isochronic fork. *Integration, the VLSI journal*, 13(2):103–128, June 1992.

[2] T.-A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, MIT Laboratory for Computer Science, June 1987.

[3] G. D.Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996.

[4] K. M. Fant and S. A. Brandt. NULL conventional logic: A complete and consistent logic for asynchronous digital circuit synthesis. In *International Conference on Application-specific Systems, Architectures, and Processors*, pages 261–273, 1996.

[5] C. Jeong and S. M. Nowick. Optimal technology mapping and cell merger for asynchronous threshold networks. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 128–137. IEEE Computer Society Press, Mar. 2006.

[6] A. Kondratyev and K. Lwin. Design of asynchronous circuits using synchronous CAD tools. *IEEE Design & Test of Computers*, 19(4):107–117, 2002.

[7] L.Heller, W.Griffin, J.Davis, and N.Thoma. Cascode voltage switch logic: a differential CMOS logic family. In *International Solid State Circuits Conference*, pages 16–17, 1984.

[8] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev. Asynchronous design using commercial HDL synthesis tools. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 114–125. IEEE Computer Society Press, Apr. 2000.

[9] A. J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C. A. R. Hoare, editor, *Developments in Concurrency and Communication*, UT Year of Programming Series, pages 1–64. Addison-Wesley, 1990.

[10] A. J. Martin. Asynchronous datapaths and the design of an asynchronous adder. *Formal Methods in System Design*, 1(1):119–137, July 1992.

[11] K. M.Fant. *Logically Determined Design - Clockless System Design with NULL Convention Logic*. John Wiley & Sons, 2005.

[12] G. Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[13] C. D. Nielsen. Evaluation of function blocks for asynchronous design. In *Proc. European Design Automation Conference (EURO-DAC)*, pages 454–459. IEEE Computer Society Press, Sept. 1994.

[14] L. Y. Rosenblum and A. V. Yakovlev. Signal graphs: from self-timed to timed ones. In *Proceedings of International Workshop on Timed Petri Nets*, pages 199–207, Torino, Italy, July 1985. IEEE Computer Society Press.

[15] R. Rudell. *Logic Synthesis for VLSI Design*. PhD thesis, U.C. Berkeley, 1989.

[16] C. L. Seitz. System timing. In C. A. Mead and L. A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[17] S. C. Smith, R. F. DeMara, J. S. Yuan, D. Ferguson, and D.Lamb. Optimization of null convention self-timed circuits. *Journal of Systems Architecture*, 37(3):135–165, Aug. 2004.

[18] S. C. Smith, R. F. DeMara, J. S. Yuan, M. Hagedorn, and D. Ferguson. Delay-insensitive gate-level pipelining. *Integration, the VLSI journal*, 30(2):103–131, 2001.

[19] G. E. Sobelman and K. Fant. CMOS circuit design of threshold gates with hysteresis. In *Proc. International Symposium on Circuits and Systems*, pages 61–64, June 1998.

[20] J. Sparsø and J. Staunstrup. Delay-insensitive multi-ring structures. *Integration, the VLSI journal*, 15(3):313–340, Oct. 1993.

[21] T. E. Williams. *Self-Timed Rings and their Application to Division*. PhD thesis, Stanford University, June 1991.