Asynchronous Combinational Hardware Design Flow for XILINX FPGAs

Igor Lemberski (igor.lemberski@inbox.lv)

Riga Technical University Faculty of Computer Science and Information Technology: Rigas Tehniska universitate Datorzinatnes un informacijas tehnologijas fakultate https://orcid.org/0000-0001-9596-1506

Marina Uhanova

Riga Technical University Faculty of Computer Science and Information Technology: Rigas Tehniska universitate Datorzinatnes un informacijas tehnologijas fakultate

Viktors Gopejenko

ISMA University of Applied Sciences: Informacijas Sistemu Menedzmenta Augstskola

Artjoms Suponenkovs

Riga Technical University Faculty of Computer Science and Information Technology: Rigas Tehniska universitate Datorzinatnes un informacijas tehnologijas fakultate

Aleksandrs Berezhnojs

ISMA University of Applied Sciences: Informacijas Sistemu Menedzmenta Augstskola

Research Article

Keywords: asynchronous logic, design flow, dual-rail function, decomposition, look-up-table

Posted Date: May 2nd, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1352990/v1

Asynchronous Combinational Hardware Design Flow for XILINX FPGAs

Igor Lemberski [1] (corresponding author)
igor.lemberski@inbox.lv

Marina Uhanova [1]
marina.uhanova@rtu.lv

Viktors Gopejenko [2]
viktors.gopejenko@venta.lv

Artjoms Suponenkovs [1]
artjoms.suponenkovs@rtu.lv

Aleksandrs Berezhnojs [2]
alexander.v.berezhnoy@gmail.com

[1]*Riga Technical University*
Faculty of Computer Science and Information Technology
Setas. Iela 1, LV-1048, Riga, Latvia

[2]*ISMA University of Applied Sciences*
Lomonosova iela 1, LV-1019, Riga, Latvia

An asynchronous combinational hardware design flow for commercially available XILINX devices is discussed. It consists of logical and technological stages.
At the logical stage, the multi-level logic is optimized using resubstitution procedures. Also, a relaxation procedure is proposed to increase the speed at the reset stage.
The optimization produces non-indicative circuits. The conditions of the distributed indication are formulated and the procedure is proposed.
The technological stage is based on Webpack place and route software. To ensure stability inside a circuit, timing constraints (less strong than proposed in the literature) are formulated and can be easily satisfied. The simulation shows that resulting circuits are hazard-free ones.
A set of benchmarks is processed. Compared to the conventional dual-rail design, the logical optimization reduces the total number of look-up-tables (LUTs) and number of LUTs in a critical path. At the technological stage, the experiments show that before the relaxation, the propagation delay in the set phase is smaller (due to early output) than in the reset phase. It is confirmed that the relaxation increases the speed in the reset phase. Also, the (dynamic) power is higher for the benchmarks optimized for speed.

Keywords: asynchronous logic, design flow, dual-rail function, decomposition, look-up-table

1. Introduction

A field-programmable gate array (FPGA) is an attractive environment for the asynchronous implementation. XILINX FPGA is one of the most popular programmable logic devices. It comprises of logic function generators (called Look-Up-Tables-LUTs) grouped into Configurable Logic Blocks (CLBs), latches and wires between CLBs. Currently, it is supposed for the synchronous implementation. Both academic and commercial tools have been created for the design. The majority of publications on asynchronous design targeting reconfigurable logic are based on specific architectures, what narrows the range of potential users. From our point of view, more prospective approach is to develop asynchronous logic design methods and tools based on commercially available hardware and software. It may raise interest in asynchronous logic among the professionals with the expertise in synchronous hardware design.

The goal of the paper is to demonstrate that commercially available reconfigurable logic is suitable for asynchronous design.

The most of existing asynchronous design methods refer to so called Quasi-Delay-Insensitive (QDI) logic, where both gates and wire delays are taken into account. Following timing constraints should hold: state-holding (C-) element feedback delay is close to zero and wire forks are isochronic ones (means, that from a fork point to any end point a signal propagates at the same time). The above constraints ensure stability inside the circuit by the moment when the signals reach the outputs.

An approach to asynchronous logic synthesis targeting XILINX and ALTERA devices is proposed in[1]. It is based on the design using a commercial tool and further manual correction to satisfy isochronic constraints.

However, in practice it is rather difficult (or sometimes even not possible) to balance wire delays. In the paper, the timing constraints (less strong than isochronic ones) are formulated and can be easily satisfied.

The most popular method of QDI logic design is based on dual-rail Delay-Insensitive Minterm Synthesis[2] (DIMS). DIMS operates under so called strong constraints, what means that output signals change the state, when all input signals change the state. As a result, the outputs indicate a moment, when DIMS computations are completed. The other state-of-the-art methods are oriented on the implementation using direct logic[2], Null-Convention-Logic (NCL)[3], simple (NOR, NAND etc)[4,5] and complex (NAND-NAND) [6,7] gates.

NCL_D and NCL_X are approaches for multi-level synthesis[5]. Both of them are based on logic decomposition into nodes (gates or even more complex functions[7] of given number of inputs) and transformation into dual-rail modules. In NCL_D, a node is implemented as DIMS, where Sum-Of-Minterms (SOM) functions are represented as a Sum-of=ONset (SONM) and OFfset Minterms (SOFM) and each minterm is implemented as a state-holding (C-) element. It ensures operation under the strong constraints. In SOM, minterms are mutually orthogonal ones. As a result, each input state is covered by a single term (minterm). In the case of NCL_X, Sum-Of-Products (SOP) representation obtained after the minimization is allowed. As a result, nodes are simpler and faster, operate under so called weak constraints (part of output signals is in a new state, when part of input signals is in a new state, however all outputs are in the new state, when all inputs are in the new state) and modified weak constraints[6], known also as an early output (all outputs change a state, if not all inputs change the state). The distributed indication[8] or extra Completion Detection (CD) circuitry is needed to indicate the moment, when computations are completed.

Recently, much improvement was performed in NCL design. The NCL implementations of simple gates[9], more complex return-to-zero threshold logic[10] as well as combination of return-to-zero and return-to-one gates[11,12] were proposed. The area (number of transistors) and performance were optimization criteria.

Since LUT architecture implements SOM logic, the design methods are oriented on NCL_D logic (where each node is represented as conventional DIMS (see Ch.4.1) or more economical (w.r.t. LUT capacity) SOP-based DIMS (Ch.4.2) which was recently proposed[18] and incorporated into our design flow. The area is expressed as a number of

LUTs (not transistors) and the speed is estimated as the longest (critical) path from inputs to outputs.

Our design flow consists of logical and technological stages. The features of our logical design are as follows: 1) in single-rail, each node is a complex (SOP) function of given number of inputs; it is in contrast to existing approaches, where each node is a simple gate (NOR, NAND etc), 2) an initial (single-rail) optimal network is produced using ABC script[13], 3) transformed into dual-rail one and 4) further optimized using a resubstitution procedure. The resubstitution is formulated and solved as a covering task: the output of the node, whose inputs have been selected for the resubstitution, is split into the set of dichotomies. The set of inputs satisfying formulated target (optimization for the speed or area) are sought to cover the dichotomies. The resulting network may not be indicative. Instead of acknowledgement block, the distributed indication is produced.
The technological design is based on Webpack place and route software. The communication between stages is done via Webpack interface in Electronic Design Interchange Format (EDIF)[21]. To transform the circuit logical description into EDIF, a converter has been developed and included into the design flow. To ensure stability inside a circuit, timing constraints (less strong than conventional ones) are formulated and can be easily satisfied.

2. CLB and LUT Overview

In XILINX, a Look-Up-Table (LUT) is a main resource for logic function generation. A few LUTs are combined into a structure called a Configurable Logic Block (CLB).
CLB architecture varies across XILINX families. For example, 7-series FPGA CLB[14] contains two slices (fig.1) with four *6* input LUTs and 8 latches each.

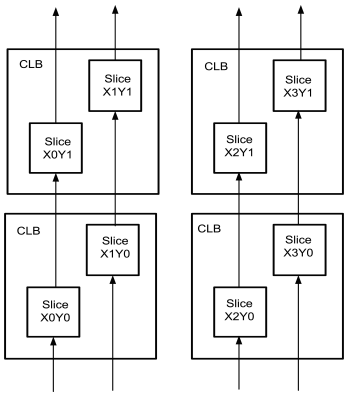


Fig. 1. CLB architecture

A typical *k*-input LUT (further, *k*-LUT) consists of a multiplexer with *k* select inputs and SRAM indexed by 2^k (all possible) input combinations (fig.2a,b). Any function value may be indexed by the select inputs values. The LUT structure is carefully designed w.r.t.

delays, namely 1) the propagation delay is the same and does not depend on the function; 2) inside LUT, both input signal and its inversion (internally produced by LUT) switch at the same moment. The mention features will be exploited in hazard-free logic design.

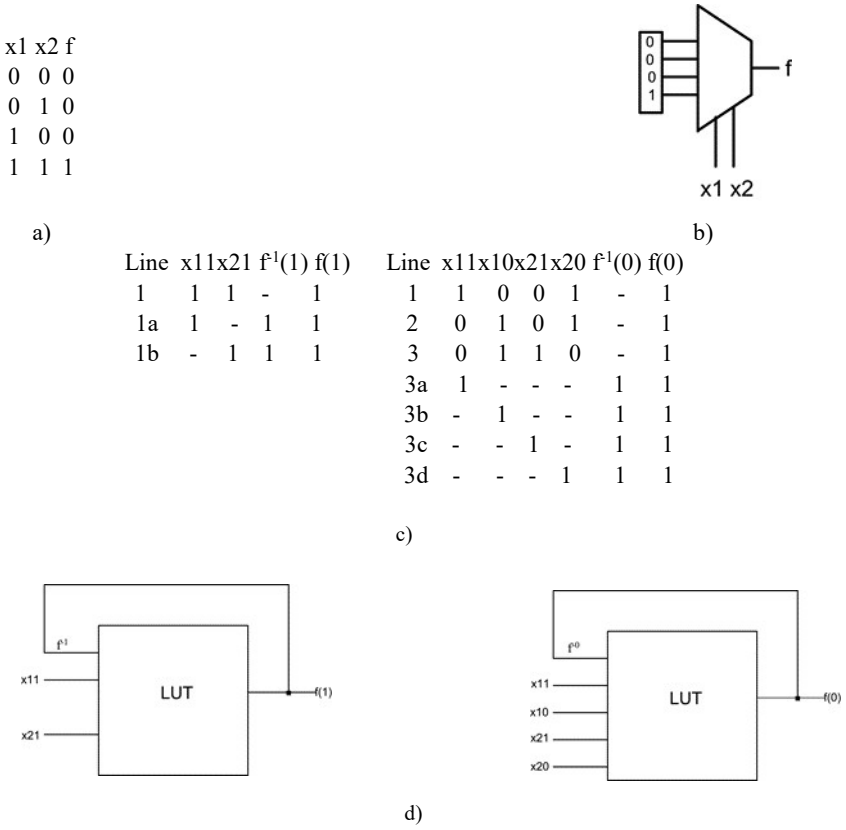


Fig.2. Function AND truth table(a), LUT structure(b), DIMS implementation(c), LUT structure(d)

3. Dual-rail logic

In dual-rail, Sum-of-ONset Products (SONP) and Sum-of-OFfset Products (SOFP) are implemented. Let *f(1), f(0)* be such functions of *n* variables: $x_1, x_2, \ldots, x_n$. It is supposed that each variable *x* may be in one of three states: states *1*, *0* (so called working states) or undefined (spacer state). To represent three-state input x_i, $i = 1, 2, \ldots, n$, two signals x_{i1} and x_{i0} are introduced, where $x_{i1} = 1$ and $x_{i0} = 0$, if x_i is in the state 1, $x_{i1}=0$ and $x_{i0}=1$, if x_i is in the state 0, $x_{i1}=x_{i0}=0$, if x_i is in the spacer state. The combination $x_{i1}=x_{i0}=1$ is not allowed. Similarly, functions *f(1)* and *f(0)* represent three-state function. For example, function AND SONP and SOFP representations are as follows: $f(1) = x_{11}x_{21}$, $f(0) = x_{10} \vee x_{20}$.

4. Mapping into LUT

4.1. *DIMS*

In[15], LUT-oriented DIMS implementation is proposed for the dual-rail SONM and SOFM function, where the function is implemented together with C-element as a whole (fig.2c). For function AND, SONM and SOFM representations are as follows: $f(1) = x_{11}x_{21}$, $f(0) = x_{10}x_{20} \vee x_{11}x_{20} \vee x_{10}x_{21}$. Additional lines (1a,1b and 3a-3b) and feedbacks $f^1(1), f^1(0)$ are required to implement C-elements. The feedbacks should be close to zero[1]. Since (in some cases) feedback goes through the global routing, its length and therefore, the delay value may be much greater than zero. Instead of feedback zero delay, the timing constraint which is less strong and easily implemented will be offered in Ch. 5. As already mentioned, DIMS ensures strong indication. No hazards are produced since switching from/to spacer/working state implies switching a single minterm. LUT structure implementing DIMS is given in fig.2d. It requires two LUTs of *8* inputs in total.

4.2. *SOP-based DIMS*

SOP-based DIMS is designed to reduce the capacity of LUT required for the implementation. In the contrast to conventional DIMS, more than one minterm may be switched on/off. However, such a behavior does not result in hazards and the functionality is preserved.

Given SOP of single–rail function: $f=x_1x_2 \vee x_2x_3$. Its dual-rail SONP is: $f(1)= x_{11}x_{21} \vee x_{21}x_{31}$ and SONM (suitable for LUT implementation) is represented as follows: $f(1)= x_{11}x_{21}x_{31} \vee x_{10}x_{21}x_{31} \vee x_{11}x_{21}x_{30}$.

One can see that SONP does not depend on signals x_{10}, x_{30}. Therefore, it would be attractive to remove these signals from the dual-rail SOM representation and therefore reduce LUT capacity required for the implementation.

Dual-rail SONM can be re-written as follows: $f(1)= x_{11}x_{21}x_{31} \vee x'_{11}x_{21}x_{31} \vee x_{11}x_{21}x'_{31}$, where x'_{11}, x'_{31} – inversion, internally produced by LUT (fig.3). Similar to conventional DIMS, state-holding logic should be implemented based on the feedback (for simplicity, not shown in fig.3). We call such representation as SOP-based DIMS.

There are three working states that switches the function output on:

1) $x_{11}=x_{21}=1, x_{31}=0, x'_{11}=0, x'_{31}=1$ (state 1)

2) $x_{11}=0, x_{21}=x_{31}=1, x'_{11}=1, x'_{31}=0$ (state 2)

3) $x_{11}=x_{21}=x_{31}=1, x'_{11}=x'_{31}=0$ (state 3)

and spacer state $(x_{11}=x_{21}=x_{31}=0, x'_{11}=x'_{31}=1)$.

Assume, *d* is a LUT propagation delay.

Suppose switching from spacer state to:

1) state 1. Switching signals x_{11}, x_{21} on results in switching minterm $x_{11}x_{21}x'_{31}$ and (after delay d) the output on (fig.3, line 3)). No switching the rest of minterms.

2) state 2. Switching signals x_{21}, x_{31} on results in switching minterm $x'_{11}x_{21}x_{31}$ and (after delay d) the output on (fig.3, line 2). No switching the rest of minterms.

2) state 3. The process is illustrated by timing diagrams (fig.4). Suppose, signals x_{11}, x_{21} switch their values on before it is done by the signal x_{31}. After delay *d*, line 3 switches on what results in switching on the output (early indication). Once signal x_{31} switches on, signal x'_{31} switches off at the same moment.

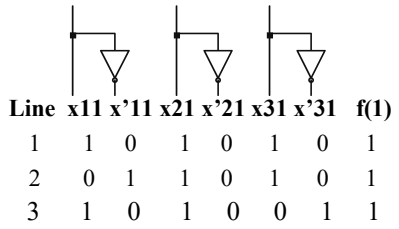
Line 3 switches off and line 1 switches on simultaneously. Therefore, no glitch occurs and the output keeps stability.

The same conclusion is done when signals x_{11}, x_{31} as well signals x_{21}, x_{31} change their values before it is done by signal x_{21} and signal x_{11} respectively.

Similarly, once a circuit switches from a working state to spacer one and signal x_{31} switches off before signals x_{11}, x_{21}, then signal x'_{31} switches on at the same moment. Line 2 switches off and line 3 switches on simultaneously. As a result, no glitch occurs and output is stable. The same conclusion can be done once signal x_{11} or signal x_{21} switches off first.

One can see that SOP-based implementation targeting LUT depends on three external signals (x_{11}, x_{21}, x_{31}) and requires LUT of *4*-inputs (remember, a single input is required to arrange the feedback).

Note, that the conventional SOM implementation is based on dual-rail function: $f(1)= x_{11}x_{21}x_{31} \vee x_{10}x_{21}x_{31} \vee x_{11}x_{21}x_{30}$ and requires *6*-input LUT.



Line	x11	x'11	x21	x'21	x31	x'31	f(1)
1	1	0	1	0	1	0	1
2	0	1	1	0	1	0	1
3	1	0	1	0	0	1	1

Fig.3. SOP-based DIMS implementation

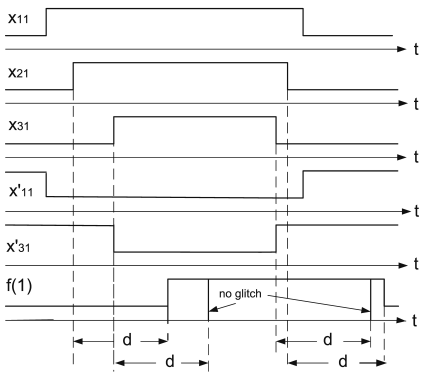


Fig.4. Timing diagrams

4.3. *From Two-Level to Multi-Level Representation*

Single-rail two-level multi-output logic can be represented by multi-output PLA (truth) table of dimension $p*(n+m)$, where p is number of minterms for which m functions are described, n-number of primary variables (fig.5a).

```
x0x1x2f0f1          .inputs x0 x1 x2
  000 00            .outputs f0 f1
  001 10            .names x0 x1 g1
  010 01             00 1
  011 00            .names x1 x2 g2
  100 10             00 1
  101 00            .names g1 g2 f0
  110 01             10 1
  111 00             01 1
                    .names x2 g2 f1
                     00 1
```

a) b)

```
.inputs x01 x00 x11 x10 x21 x20
.outputs f01 f00 f11 f10
.names x00 x10 g11                  .names x11 x21 g20
 11 1                                1- 1
.names x10 x20 g21                   -1 1
 11 1
.names x20 g20 f11                  .names x21 g21 f10
 11 1                                1- 1
.names g10 g11 g20 g21 f01           -1 1
 0110 1
 1001 1                             .names g10 g11 g20 g21 f00
.names x01 x11g10                    1010 1
 1- 1                                0101 1
 -1 1
```

c)

Fig.5. PLA table (a), single-rail SOP BLIF(b), dual-rail SOP BLIF (c)

Using existing tool (ABC), the initial (single-rail) multi-level network is produced in BLIF[22] format (fig.5b), where each node is a complex (AND-OR) function[23] of given number of variables and represented as SOP; it is in contrast to[5], where each node is a simple gate (NOR, NAND etc.). Each node function depends of given number of inputs at most and is represented as a single-output PLA table of three valued (0,1, do not care) input vectors, where the output for the listed input vectors accepts value 1. The set of primary inputs and outputs are given in the BLIF header, using keywords .inputs, .outputs. For each PLA table, inputs and single-output names are described using keyword .names.

In fig. 5a, initial PLA is given and SOP BLIF is produced (fig. 5b). Then each node function is transformed into dual-rail SOP (fig.5c). For mapping into LUTs, SOP-based DIMS is produced (fig.6). In fig.5,6, notations *x01, x11, x21, f01, f11, g11, g21* stand for

$x_0(1)$, $x_1(1)$, $x_2(1)$, $f_0(1)$, $f_1(1)$, $g_1(1)$, $g_2(1)$ respectively. Similarly, notations *x00, x10, x20, f00, f10, g10, g20* stand for $x_1(0)$, $x_2(0)$, $x_3(0)$, $f_0(0)$, $f_1(0)$, $g_1(0)$, $g_2(0)$ respectively.
Inverted signals produced by LUT are not shown.
For *k*-LUTs library, the number of each node inputs in dual-rail should not exceed *(k-1)* (one input is required to arrange the feedback). Therefore, the single-rail network should contain nodes of $\lfloor (k-1)/2 \rfloor$ inputs at most. As a result of the decomposition, LUT utilization ratio (number of LUTs inputs supplied by signals vs the total number ones) may be low since 1) some nodes depend on less than $\lfloor (k-1)/2 \rfloor$ variables; 2) in dual-rail, at least one LUT input will always be unused if *k* is an odd number. To utilization ratio is improved during the resubstitution.

```
.inputs x01 x00 x11 x10 x21 x20
.outputs f01 f00 f11 f10
.names x00 x10 g11
 11 1
.names x10 x20 g21
 11 1
.names x20 g20 f11
 11 1
.names g10 g11 g20 g21 f01
 0110 1
 1001 1
.names x01 x11 g10
 10
 01
 11
.names x11 x21 g20
 10 1
 01 1
 11 1
.names x21 g21 f10
 10 1
 01 1
 11 1
.names g10 g11 g20 g21 f00
 1010 1
 0101 1
```

Fig.6. SOP-based DIMS

5. Timing constraints

The stability is a crucial problem in asynchronous logic. Namely, once output signals respond to the new input state, one has to be sure that each signals inside a circuit reaches a new state and remains in this state. The known solution is based on two constraints: 1) a feedback delay which ensures state-holding (C-) element implementation should be close to zero; 2) all wire forks are isochronic ones[16]. We are aware, that in some cases, non-isochronic forks does not imply instability[17]. The methods of detecting such forks have been developed for circuits implemented using simple gates and cannot be applied directly to LUT- oriented logic. Furthermore, the experiments show that the number of non-isochronic forks are usually very small (around 10% on average). In our approach, we follow the constraint[16]. Forked signal timing constraints (less strong than isochronic one) are formulated.

5.1. *Feedback delay constraint*

Consider a fragment of a circuit which includes two LUTs (fig.7) (for simplicity, lower LUT feedback is not shown). Suppose that the signal propagates to the output through the global routing with the delay D. Similarly, the feedback signal propagates through the

global routing of delay *P*. To ensure circuit stability, the completion detection (CD) signal should switch on when both circuit output and LUT input (through the feedback) signals are reached. The indication is based on the output signal propagation time *D* and delay *t*, which is set to satisfy following inequality: $P < D+t$.

5.2. *Wire fork constraint*

Consider a forked signal that supplies two LUTs and propagates to the output through the upper path and does not propagate through the lower path (fig.7). To ensure circuit stability, the signal CD should go up once the forked signal 1) propagating through the upper path reaches the output and 2) propagating through the lower path reaches a LUT input. As a result, the indication should satisfy following inequality: $d_1+D+t > d_2$ *or* $d_2-d_1 < D+t$.

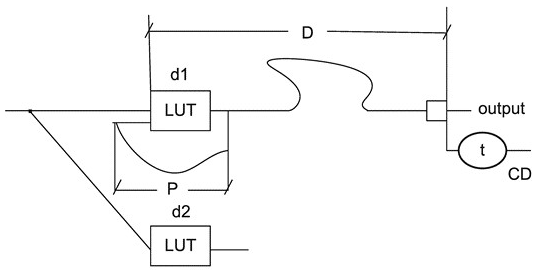


Fig.7. Timing constraints

Since the path to the output involves global routing, delay *D* is much greater than delay difference $d=d_2-d_1$ in forked wires. Therefore, it is expected that delay $t=0$. Also, delay *d* can be decreased by mapping into neighboring LUTs and CLBs. Note, that there is no constrain, which guides the neighboring mapping. Therefore, User Constraints File (UCF) constraints LOC (specifying slice) and BEL (specifying LUT in the given slice) are used.

The example of using LOC, BEL constraints is as follows:

```
INST "XLXI_1/LUT_f1"  BEL = D6LUT;
INST "XLXI_1/LUT_f1"  LOC = SLICE_X2Y59;
```

where function *f1* is mapped into D6LUT of slice X2Y59.

6. Design flow

The preliminary version of the design flow was proposed in[19]. The final version is described in this paper. The design flow is based on Webpack ISE. Originally, Webpack is oriented on synchronous design and includes logical and technological stages (fig. 8). An interface between these two stages is represented in EDIF[21]. In our design (fig.9),

optimal asynchronous synthesis (which replaces Webpack logical stage) is proposed. It produces an optimal multi-level network in BLIF using ABC tool, its transformation into the dual-rail network, further optimization (resubstitution), indication and relaxation procedures are applied (the details will be described further). Next, the network is compiled into EDIF (2EDIF) (the compiler has been developed and incorporated into the design flow) and passed to Webpack technological stage, where place and route (P&R) is done. It is supported by UCF, where design constrains (LOC, BEL) are described. At the technological stage the features of design are as follows:

- to ensure timing constraint, delays *D, P* and *d* (see Ch. 5) are measured and once for 1) each LUT *D*>*P*, and 2) each fork D>d then delay *t* is set to 0; otherwise, delay *t* is chosen to satisfy following inequalities: *t*>*P*-*D* (for each LUT) and *t* > *d*-*D* (for each fork).
- to reduce delay *d*, LUTs supplied by forked signals are allocated as closely as possible (preferably, in the same or neighbouring CLBs) what can be described in UCF; it results in reducing delay *t;*
- P&R tool is executed;
- the above steps iterate (branch NO in fig. 9) till the moment, when delay *t* becomes suitable (or *0*) or cannot be reduced more (branch YES).

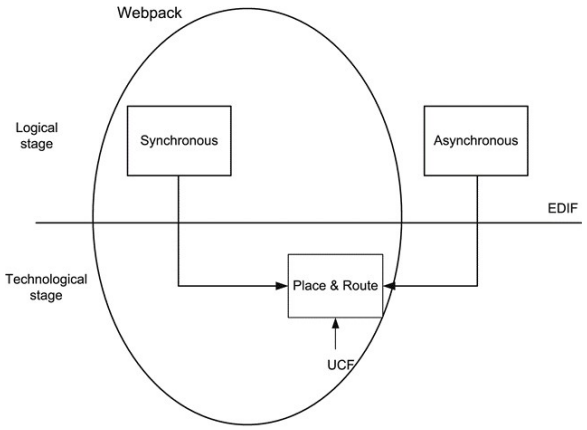


Fig. 8. Synchronous and asynchronous design using Webpack

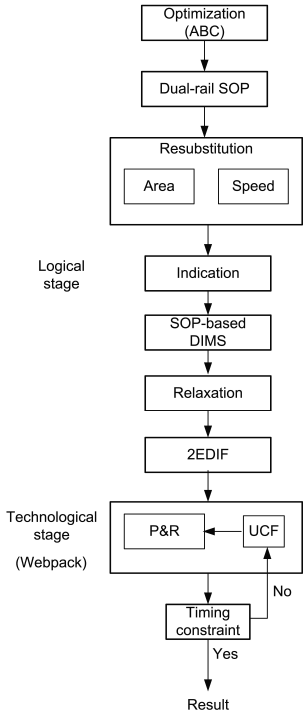


Fig.9. Design flow

7. Resubstitution

The optimization includes the resubstitution procedure which is based on the structure called an extended PLA table.

7.1. *Extended PLA table*

Consider dual-rail multi-output function *F,* $|F|=2m$, *m*-number of single-rail outputs, which depends on set *X* of inputs: $|X|=2n$, *n*-number of single-rail inputs. Suppose that its dual-rail SOP representation in BLIF contains the set of nodes G, where each node function depends on set X′ of inputs, $X' \subset X \cup G \cup F$.

Taking the advantage of PLA format representation into account, it is expediently to describe SOP BLIF representation by so called extended PLA table.

Initially, primary input vectors in dual-rail are listed. The procedure of the extended PLA table design is as follows:

1) Nodes are ranged in the topological order: primary inputs are assigned to level *0*; nodes depending on signals of level *(i-1)* or less are assigned to level *i*.

2) Node functions of level i are designed based on nodes outputs of levels *0,1,…, i-1* and added to the PLA table;

3) For each extended PLA table row, an embedded vector of *X'* variables is checked if it covers the node function g_i onset vector of *X'* variables. If yes, value *1* is put on the cross point of this row and column g_i. Otherwise, value *0* is put.
For BLIF (fig.5), extended PLA is given in table 1. Initially, 8 vectors of primary inputs (fig.5a) are transformed in dual-rail and listed (columns *x01, x11, ..., x20*). As an example, consider design of level *1* node function *g20*. It depends on inputs *X'={x11, x21} (1-, -1)* and cover embedded vector *X'* of rows *2,3,4,6,7*. Value *1* is put on the cross points of column *g20* and mentioned rows.
Similarly, the remaining columns are designed.

Table 1. Dual-rail extended PLA

Row No	x01	x00	x11	x10	x21	x20	g11	g10	g21	g20	f01	f00	f11	f10
1	0	1	0	1	0	1	1	0	1	0	0	1	0	1
2	0	1	0	1	1	0	1	0	0	1	1	0	0	1
3	0	1	1	0	0	1	0	1	0	1	0	1	1	0
4	0	1	1	0	1	0	0	1	0	1	0	1	0	1
5	1	0	0	1	0	1	0	1	1	0	1	0	0	1
6	1	0	0	1	1	0	0	1	0	1	0	1	0	1
7	1	0	1	0	0	1	0	1	0	1	0	1	1	0
8	1	0	1	0	1	0	0	1	0	1	0	1	0	1

7.2. *Optimization via Resubstitution*

The resubstitution is a procedure of replacing a set of variables, a node depends on, by the other set of variables while preserving node functionality. Such a replacement is accepted if a better network (for example, simpler in some sense as given one) is produced.

7.2.1. *Dichotomies and a covering table*

Suppose, one needs to resubstitute input g_i of node g_j while preserving functionality of node g_j. The latter means that for each pair of on-, offset minterms there exists at least one input distinguishing this pair.
Definition 8.2.1. A seed dichotomy (further called a dichotomy for brevity) is a 3-valued vector accepting value 1 in any single position, value 0 in any other single position and do not cares in the remaining positions.
Assume, a function onset has *anum* minterms, an offset contains *bnum* minterms. A set of *anum x bnum* dichotomies is created. Each dichotomy is associated with a *pair (i,j)* of *i-th* onset and *j-th* offset minterms, $1 \le i \le anum$, $anum+1 \le j \le anum+bnum$ and contain value *1* in the *i-th* position, value *0*- in *anum+j* position, do not care value-in the remaining positions. A set of dichotomies can be presented as a table of size *(anumxbnum)x x(anum+bnum)* where each column is associated with a dichotomy and each row - with an on- /offset vector. Function *f01* (table 1) dichotomies are given in table 2. A covering table (table 3) is a two-dimension matrix, where each column associates with a dichotomy and each row-with an input. Note, that in dual-rail logic the function is represented as the set of onset minterms, where each minterm depends on positive

variables. It implies a covering rule as follows: if dual-rail variable x, $x \in \{x_1, x_0\}$, distinguishes dichotomy *(i,j),* where i is the onset minterm, j-offset one, it should accept value *1* in the *i-th* onset and *0* in the *j-th* offset. The complement function is represented as a set of offset minterms. Therefore, to distinguish the above mentioned dichotomy *(i,j)*, variable x should accept value *1* in the *j-th* offset and *0* in the *i-th* onset. The covering table is given in table 3.
Since node g_{10} has fanout 1, it can be considered for removing. Since dichotomy 7 is covered by input x_{01}, it can resubstitute input g_{10} (crossed out).
Next, node g_{11} has also fanout 1 and therefore, is a candidate for removing (crossed out). The covering includes the set of five inputs: $x_{01}, x_{00}, x_{10}, x_{21}, x_{20}$. (in grey).

7.2.2. *Resubstitution for speed*

A path between inputs and the output, where a signal propagates the maximal number of nodes (LUTs) is called as critical one.
At the logical level, the speed is estimated the number of nodes in the critical path.
Suppose, node g_j is on a critical path. It is supplied with input(-s) of maximal level L. The resubstitution replaces input(-s) of level L for ones of maximal level L', $L' < L$, while preserving node functionality. Note, that the number of node inputs should not exceed *(k-1),* where k-number of LUT inputs (one input is used to arrange a feedback).

7.2.3. *Resubstitution for area*

At the logical level, the area is estimated as the number of nodes (LUTs) in the multi-level network.
Given node g_j supplied by node g_i output of fan-out n_0. Once node g_i is replaced, its fan-out n_0 decreases: $n_0 = n_0 - 1$ (fig.10).

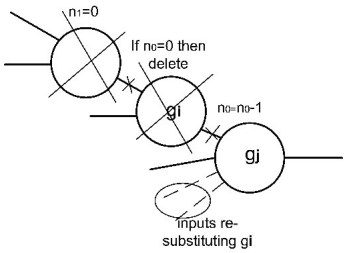


Fig.10. Resubstitution

The resubstitution is applied iteratively either to achieve $n_0 = 0$ or make sure, that fanout n_0 cannot be reduced further. If $n_0 = 0$, node g_i is removed, what results in decreasing the fan-out of the node, supplying node g_i: $n_1 = n_1 - 1$. This node is also removed, if $n_1 = 0$ and so on. Two strategies of selecting sets of inputs as the candidates for the cover are applied. If $n_0 \neq 1$, then firstly, such inputs are sought among the set of primary inputs and ones produced by the nodes, which have already been considered to be resubstituted,

however their fan-outs didn't reach value *0*. If the cover isn't successful, outputs of the remaining nodes are also considered as candidates for the cover. If priority is given to the number of levels, only those nodes which do not increase the critical path, should be involved in the covering procedure. Again, the limitation on the node inputs number *(k-1)* should hold.

Table 2. Function f01 dichotomies

Line No	Dichotomy No 1 2 3 4 5 6 7 8 9101112
1	0 - - - - - 0 - - - - -
2	1 1 1 1 1 1 - - - - - -
3	- 0 - - - - - 0 - - - -
4	- - 0 - - - - - 0 - - -
5	- - - - - - - 1 1 1 1 1 1
6	- - - 0 - - - - - 0 - -
7	- - - - 0 - - - - - 0 -
8	- - - - - 0 - - - - - 0

Table 3. Covering table

Inputs	Dichotomy No 1 2 3 4 5 6 7 8 9101112
x01	x x x
x00	x x x
x11	
x10	x x x x x x x x
x21	x x x
x20	x x x
g11	~~x x x x x~~
g10	~~x~~
g21	x x x x x
g20	x
f11	
f10	x x x x

8. Operation discipline and indication

8.1. *Operation in a set- up phase*

In a set-up phase, proposed SOP-based logic operates under early output discipline (fig.11). Therefore, it may be non-indicative. Consider SOP-based DIMS function (Ch.4.1) $f(1)= x_{11}x_{21}x_{31} \vee x'_{11}x_{21}x_{31} \vee x_{11}x_{21}x'_{31}$. If $x_{21}=x_{31}=1$, then output $f(1)=1$ independently on x_1 value. As a result, input x_1 is not acknowledged by the function. Similarly, if $x_{11}= x_{21}=1$, then output $f(1)=1$ independently on x_3 value. Note, that resetting all inputs is acknowledged by resetting LUT output (since each input is supported by a state-holding element). Therefore, during the reset phase logic operates as strongly indicative one and the indication should be ensured in the set phase only. As a result, operation disciplines in the set/reset phases are asymmetric. Furthermore, in the set phase the operation speed is higher (due to early output) than in the reset phase, where outputs

change the states once all inputs change their states. Once logic is incorporated into multi-level circuit, the distributed indication is required.

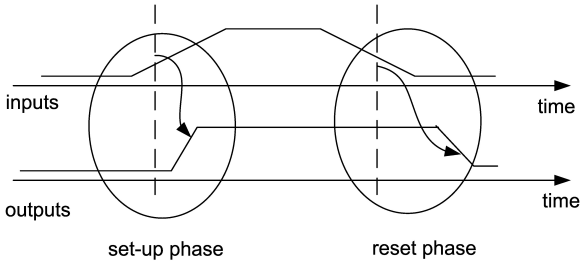


Fig. 11. SOP-based logic operation discipline

8.2. *Distributed Indication*

Definition 9.2.1. A multi-level circuit is an indicative one, if any node input (both primary and internal) change is acknowledged by the node output (either primary or internal) change. As a result, all primary inputs are acknowledged by the primary outputs. Given: 1) a node function $g_i(X_i)$, where X_i is a set of signals, generating input states; 2) $\{A,B,C,...\}$ – a set of input states; 3) transitions between them. Suppose, logic operates under four phase behaviour discipline, where state A is a spacer one (all input signals are in zero: *00..0*); (*B,C,...)* are working states (at least one input signal is in 1). As shown, indication in the set phases only should be checked and, if necessary, ensured. Consider a transition between states *A, B*.

Definition 9.2.2. The transition states are ones, which occur during transition between states *A* and *B*.

Denote the set of transition states as *(A, B).*

Definition 9.2.3. The transitions are called as ones with mutually exclusive transition states if $C_j \notin (A,B_i)$ for each transition between states *A,* B_i and working state C_j, which can appear after state *A:* $C_j \neq B_i$, $B_i, C_j \in \{B, C, ...\}$.

Note, that logic with mutually exclusive transition states is hazard-free and indicative one.

Theorem. The logic contains mutually exclusive transition states, iff there is no pair of working states (say, *B, C*), which are in relation: B ≥ C. The proof can be found in [23].

Example. Consider function g_i (x_1,x_2,x_3,x_4) (fig.12a). Set *(A,B)*, where *A=0000* (spacer state) and *B=1010* (1st row) is: *(A,B)={1000, 0010}*. One can find state *C, C=0010* (3rd row), where $B \geq C$, and therefore, state *C* can appear after state *A*, $C \in (A,B)$. As a result, input x_1 isn't indicated.

To keep information whether or not any input *x* is indicated by output *g*, $x \in X'$ vector *x*(g)* is created, where input *x* states which are not indicated are switched to *0* (for input x_1, see fig.12a).

Usually, more than one node (set G' of nodes) are supplied by input x (fig. 12b). Therefore, distributed indication is supposed. In this case, vector $x^*(g)$ is created for each node $g \in G'$. The distributed indication is checked based on the following equality:

$$\bigvee_{\forall g \in G'} \left(g \& x^*(g) \right) = x, \quad (1)$$

where G'- set of nodes supplied by x, $G' \subset G$.

The procedure of the indication is proposed in[24]. It is based on the following corollary.

Corollary 1. If a node is supplied by input x and its negation y or supplied by input x only which accepts value 1 in all states, then input x is indicative one.

The procedure includes following steps:

1) Input x indication is checked using equation (1);
2) If equation (1) does not hold, a node which is not supplied by input x and/or its negation y, is extended by adding input x and /or its negation (note that total number of each node inputs should not exceed $(k-1)$). It iterates until equation (1) holds. The node selection is performed in a greedy manner (the extension which maximizes the number of indicative states is accepted).
3) Otherwise, input x cannot be indicated.

8.3. *Operation in a reset phase and relaxation*

The performance at the reset phase can be increased using a procedure called relaxation. Given the extended PLA (fig.13a), which contains functions depending on input x_1: $z_1=z_1(x_1,x_2)$, $z_2=z_2(x_1,x_3)$, $z_3=z_3(x_1,x_2,x_3,x_4)$. In the reset phase, the functions ensure indication for input x_1, namely, resetting each input state, where $x_1=1$ (see highlights in column x_1), implies at least one function resetting its output. By solving a covering task, one can see that the number of such functions can be reduced, namely, functions z_1, z_2 ensure input x_1 indication (highlights in columns z_1, z_2). It implies some features on function z_3 implementation in LUT, namely, state-holding element of input x_1 is not required (fig.13b). Such a relaxation increases the speed in the reset phase since function z_3 operates under early output discipline (fig.11).

9. Resubstitution Procedures

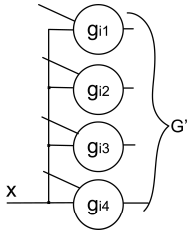
The preliminary versions of the resubstitution can be found in[24]. Compared to[24], the procedures presented below are more fast and produce better optimization result what confirmed experimentally (see Ch.10).

Procedure Resub_speed (fig.14) minimizes a critical path. It inputs an initial dual-rail BLIF and extended PLA. Within a critical path, node g_i is extracted and its inputs of the maximal level are resubstituted for inputs of less levels (lines 7,8). In the obtained circuit, the indication is checked. If successful, it replaces a current BLIF (lines 9,10) and the procedure searches for a new critical path. Otherwise, the next node in the critical path is checked. If all nodes have been checked but the critical path level hasn't been reduced, SOP-based DIMS is created and BLIF is returned.

Procedure Resub_area (fig.15) is similar to Resub_speed. Firstly, node g_i of the minimal fanout is extracted and inputs implied by g_i output, are resubstituted. If it results in a zero fanout, the node is removed. Secondly, the obtained circuit is checked for the indication. If successful, it replaces current BLIF and the procedure searches for a non-checked node of the minimal fan-out. Otherwise, the procedure continues searching for non-checked nodes. Once all nodes are checked, SOP-based DIMS is created and BLIF is returned.
Note, that any single-rail BLIF obtained as a result of ABC decomposition, can always be transformed into dual-rail indicating one by representing each node as DIMS. Therefore, indicating BLIF can always be obtained.

x1*	x1	x2	x3	x4	gi
0	1	0	1	0	1
1	1	1	0	1	1
0	0	0	1	0	1

a)



b)

Fig.12. Indication: example (a); distributed indication (b)

10. Experiment

The set of experiments has been done.

10.1. *Logical optimization*

The subset of biggest MCNC91 and IWLS'93 benchmarks has been processed. The implementation is performed using 7 series FPGA (Spartan-7). We applied logical optimization procedures for area and speed and compare result with conventional NCL_D networks with one exception: each node is a complex (AND-OR) function of given number of inputs and implemented as DIMS. It is in contrast to existing approach to NCL_D design, where each node is a simple gate. The summary of the experiment is given in table 4. In the *1st* column, names of benchmarks are listed. Since 7 series FPGA logic structure is based on *6*-LUTs, the single-rail decomposition into $\lfloor (6-1)/2 \rfloor = 2$ input nodes is done. Following ABC script is applied: *fx, st* and NCL-D network is produced. The number of levels in the critical path as well as number of nodes are listed in the next two columns. Then the number of levels and nodes in the optimized indicating networks targeting 1) area, 2) speed, 3) speed followed by area are reported in the next three pairs of columns. Compared to NCL_D the improvement is achieved in the first two cases, however it is not the case for speed+area optimization, where during the indication, the critical path length of several benchmarks (m4, pope) increases and even exceeds one in initial BLIF. The execution time for the benchmarks of less than *1000* nodes does not exceeds 1 hour. For benchmarks of more than *1000* nodes, the timeout (I hour) for each

optimization procedure (resubstitution for speed, area) is set up and the obtained result is reported.

x1x2x3x4	*z1z2z3*
1010	101
0011	000
1110	111
0111	000
1101	011
0101	000
1000	100

a)

x1x2z1^{-1}z1	*x1x3z2^{-1}z2*	*x1x2x3x4z3^{-1}z3*
10- 1	11- 1	1010 - 1
11- 1	10- 1	1110 - 1
1-1 1	1-1 1	1101 - 1
-11 1	-11 1	- 1-- 1 1
		-- 1- 1 1
		--- 1 1 1

b)

Fig.13. Fragment of extended PLA (a); functions implementation in LUTs after the relaxation(b)

10.2. *Propagation delay*

Next, we measured the propagation delays in the set and reset phases. It was done during the simulation at the technological stage. Namely, the sequence of input states was generated within time interval 10^6 ns and applied to benchmarks optimized for the speed and area (table 5,6). The optimization with no relaxation (table 5) and one followed by the relaxation (table 6) were considered. For speed-oriented optimization, the experiment confirmed smaller propagation delay for almost all the benchmarks (see "Ratio" columns in tables 5,6).

```
1 Resub_speed(blif, extendedPLA)
2 {current_blif=blif
3   for a critical path
4  {
5   for each non-checked node gi within the critical path
6   supplied with inputs of maximal level L
7    {resubstitute inputs for ones of maximal level L'<L
8      remove nodes with zero fanout, if any
9      if (indication(blif_after_resub) then
10    current_blif=blif_after_resub and break else continue
11   }
12   if critical path level isn't reduced then break
13  }
14    current_blif=SOP-based DIMS representation
15    return current_blif
16 }
```

Fig.14. Resubstitution for speed

Note, that if the relaxation is not applied, the propagation delay in the reset phase is mostly bigger, than in the set one (table 5), what expected theoretically (recall, in the set

phase, the circuit operates under early output discipline and under the strong indication in the reset phase). The relaxation speeds up the performance in the reset phase and for almost all benchmarks, the propagation delay in the reset phase becomes smaller, than in the set phase (table 6). Also, the (dynamic) power is higher for the benchmarks optimized for speed.

```
1 Resub_area(blif, extendedPLA)
2  {current_blif=blif;
3   for each non-checked node gi with minimal fan-out
4    {resubstitute inputs implied by gi output, if possible;
5      remove nodes with zero fan-out, if any;
6      if (indication(blif_after_resub) then current_blif=
7      =blif_after_resub and continue else continue;
9    }
10    current_blif=SOP-based DIMS representation
11    return current_blif;
12 }
```

Fig.15. Resubstitution for area

10.3. *Constraint calculation*

Within the simulation, the measurement was done to calculate timing constraint *t*. For this purpose, each LUT feedback as well as fork were detected and delays *P, D*, *d* (see Ch. 5) were measured. Surprisingly, for all benchmarks, $P<D$ and $D>d$. Therefore, $t=0$, One can conclude that P&R tool allocates LUTs supplied by forked signals rather closely. During the simulation, hazard-free implementation has been confirmed.

11. Conclusion and future work

The goal of the paper is to demonstrate that commercially available reconfigurable logic is suitable for asynchronous design.

A design flow oriented on asynchronous combinational logic design for XILINX devices is discussed. It consists of logical and technological stages. The logical design includes the optimal synthesis targeting area and speed. Since LUT architecture implements SOM logic, our design method is based on NCL_D. The area is expressed as a number of LUTs (not transistors) and speed is estimated as a number of LUTs in the longest (critical) path from inputs to outputs. The features of our logical design are as follows: 1) in single-rail, each node is a complex (AND-OR) function of given number of variables; it is in contrast to existing approaches, where each node is a simple gate (NOR, NAND etc.), 2) the initial (single-rail) network is produced using ABC script; 3) transformed into dual-rail one and 4) further optimized using resubstitution procedure. Then the indication and relaxation procedures are applied. The resubstitution is formulated and solved as a covering task: the output of the node, whose inputs have been selected for the resubstitution, is split into the set of dichotomies. The set of inputs satisfying formulated

target (optimization for the speed or area) are sought to cover the dichotomies. The resulting network may not be indicative. Instead of acknowledgement block, the distributed indication is produced. To speed up execution in the reset phase the procedure called relaxation is proposed.

Table 4.Area and speed for various optimization strategies

Bench-marks .	NCL-D: ABC .		Resub/area		Resub/speed		Resub/ speed+area	
	#level	#node	#level	#node	levels	nodes	levels	nodes
5xp1	9	210	7	64	5	118	5	88
9sym	16	404	14	245	13	304	13	268
apex3	16	2180	23	1230	12	1897	14	1144
b11	7	132	7	76	3	90	3	74
bw	7	284	10	129	4	158	5	145
clip	10	344	10	182	8	232	8	201
dk17	7	108	5	52	4	64	4	62
ex1010	15	2338	18	1370	13	2150	14	1403
luc	9	252	15	136	5	152	6	140
m2	11	426	14	202	7	308	11	259
m4	13	930	20	503	10	850	15	503
max46	19	280	13	162	12	216	12	176
max128	12	878	16	411	8	560	9	486
max512	15	776	15	443	11	610	14	445
pope	12	960	29	616	11	913	13	477
rd84	11	206	11	76	6	110	7	98
tms	11	338	15	210	7	220	7	170
z5xp1	10	288	10	111	6	156	6	121
z9sym	15	354	15	205	13	282	13	215

The technological design is based on Webpack ISE place and route software. The communication between stages is done in the format called EDIF. To transform the circuit logical description (in BLIF) into EDIF, a converter has been developed and included into the design flow. To ensure stability inside a circuit, timing constraints (less strong than isochronic ones) are formulated.

The experiments show that no special efforts are required to satisfy constraints. The placement is guided by UCF constraints BEL, LOC. A state-holding element is implemented based on a logic element feedback. Simulation shows, that resulting circuits are stable. The set of experiments has been done. Firstly, it was shown that compared to conventional NCL_D logic our resubstitions procedures for the area and speed reduce the number of nodes and the length of the critical path.

Secondly, propagation delays for logic networks optimized for the area and speed are measured before and after the relaxation. Before the relaxation, the propagation delay in the reset phase for almost all benchmarks is bigger, than in the set one what expected theoretically. The relaxation speeds up the performance in the reset phase and for almost

Table 5. Propagation delays, power (before relaxation)

Bench-marks	Set/ Reset	Propagation delay (ns)		Ratio (Speed/ Area)	Power (mW)
		Area	Speed		
5xp1	Set	15.042	14.927	0.992	71
	Reset	15.818	15.437	0.976	73
dc2	Set	16.726	16.141	0.965	53
	Reset	16.804	16.691	0.993	54
m2	Set	21.292	20.592	0.967	80
	Reset	23.101	21.251	0.920	86
tms	Set	24.278	17.644	0.727	72
	Reset	25.685	18.131	0.706	95
wim	Set	8.941	8.900	0.996	80
	Reset	8.997	8.898	0.989	81
max128	Set	26.641	23.365	0.877	89
	Reset	29.883	26.308	0.880	98
exp	Set	21.387	19.529	0.913	81
	Reset	22.518	22.512	0.999	84
max46	Set	18.884	18.516	0.981	23
	Reset	20.219	19.116	0.945	24
root	Set	19.248	17.942	0.932	38
	Reset	21.334	18.147	0.851	42
sqr6	Set	16.806	17.068	1.016	73
	Reset	17.053	17.098	1.003	73
sex	Set	12.861	12.201	0.949	108
	Reset	12.913	12.335	0.955	113
dc1	Set	10.365	11.060	1.067	71
	Reset	10.720	11.275	1.052	68
dekoder	Set	9.176	9.088	0.990	79
	Reset	9.306	9.227	0.992	79
newtag	Set	10.764	10.125	0.941	23
	Reset	10.763	10.124	0.941	23
risk	Set	15.007	13.013	0.867	188
	Reset	15.501	13.750	0.887	213

Table 6.Propagation delays, power (after relaxation)

Bench-marks	Set/ Reset	Propagation delay (ns)		Ratio (Speed/ Area)	Power (mW)
		Area	Speed		
5xp1	Set	15.777	15.853	1.005	71
	Reset	15.038	15.691	1.043	71
dc2	Set	16.586	15.736	0.949	53
	Reset	16.468	15.342	0.932	56
m2	Set	21.040	18.778	0.892	84
	Reset	20.766	18.058	0.870	95
tms	Set	23.759	19.054	0.802	79
	Reset	21.409	17.577	0.821	93
wim	Set	8.912	8.471	0.951	81
	Reset	8.812	7.902	0.897	87
max128	Set	25.249	21.613	0.856	102
	Reset	22.889	18.936	0.827	116
exp	Set	19.599	18.379	0.938	90
	Reset	18.735	18.060	0.964	95
max46	Set	17.868	17.756	0.994	23
	Reset	18.060	16.714	0.925	24
root	Set	20.001	17.782	0.889	39
	Reset	19.722	17.291	0.877	43
sqr6	Set	17.293	16.706	0.966	72
	Reset	17.036	16.398	0.963	75
sex	Set	12.280	12.529	1.020	115
	Reset	11.716	12.325	1.052	111
dc1	Set	10.107	10.357	1.025	73
	Reset	10.194	10.171	0.998	73
dekoder	Set	9.434	9.279	0.984	78
	Reset	9.329	9.089	0.974	79
newtag	Set	10.764	10.125	0.941	23
	Reset	10.763	10.034	0.932	24
risk	Set	13.518	13.803	1.021	208
	Reset	13.819	13.973	1.011	205

all benchmarks, the propagation delay in the reset phase becomes even smaller, than in the set phase.

Once optimization for speed is produced, for almost all benchmarks propagation delay is smaller than in the case of the optimization for area. Also, the (dynamic) power is higher for the benchmarks optimized for speed.
During the simulation, hazard-free implementation has been confirmed.
The project was started in 2017, when Webpack ISE was an actual platform for XILINX design. Further, it is planned to adapt presented design flow for a new Webpack Edition (Vivado Webpack Edition[20]). Also, the flow will be extended for sequential hardware design.

References

1. Q.T.Ho, J.-B. Rigaud, L. Fesquet, M. Renaudin, R. Rolland, Implementing Asynchronous Circuits on LUT Based FPGAs, Proceedings of the 12th International Conference on Field-Programmable Logic and Applications (FPL2002), Montpellier, France, 2002, pp. 36 – 46
2. C.D. Nielsen, Evaluation of Function Block Designs, Technical Report 1994-135, Department of Computer Science, Technical University of Denmark, Denmark, 43 pp. 1994.
3. S.C. Smith, J. Di, Designing Asynchronous Circuits using NULL Convention Logic (NCL), Morgan & Claypool, 2009, 96 p.
4. J.Cortadella, A.Kondratyev, L.Lavagno, C.Sotiriou, Coping with the Variability of Combinational Logic Delays, IEEE Int. Conf. On Computer Design (ICCD), October 2004, pp.505-508
5. A. Kondratyev, K. Lwin, Design of Asynchronous Circuits using Synchronous CAD Tools, IEEE Design & Test, Vol. 19, Issue 4, July 2002, pp. 107-117
6. I. Lemberski, P. Fišer, R. Suleimanov, Asynchronous Sum-of-Product Logic Minimization and Orthogonalization, International Journal of Circuit Theory and Applications, John Wiley & Sons Ltd, vol. 42, issue 6, June 2014, pp. 562-571
7. I. Lemberski, P. Fišer, Area and Speed Oriented Implementations of Asynchronous Logic Operating Under Strong Constraints, in Proc. of 13th Euromicro Conference on Digital Systems Design (DSD), Lille (France), September 1-3, 2010, pp. 155-162.
8. W.Toms. Synthesis of Quasi-Delay-Insensitive Datapath Circuits, Ph.D. thesis, 2006
9. R. B. Reese, R. A. Taylor, Uncle (Unified NCL Environment) [Online], Mississippi State University, 2011, available: http://my.ece.msstate.edu/faculty/reese/uncle/UNCLE.pdf
10. R. B. Reese, S. C. Smith, M. A. Thornton, Uncle – An RTL Approach to Asynchronous Design, 18th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'12), 2012, pp.
11. MT Moreira, NLV Calazans, AN Silva, MGA Martins, AI Reis, RP Ribas, Semi-custom NCL Design with Commercial EDA Frameworks: Is it Possible?, International Symposium on Asynchronous Circuits and Systems (ASYNC'14), Potsdam, 2014, pp. 53-60.
12. MLL Sartori, RN Wuerdig, MT Moreira, NLV Calazans, Pulsar: Constraining QDI Circuits Cycle Time Using Traditional EDA Tools, 25th IEEE International Symposium on Asynchronous Circuits and Systems, 2019, pp.114-123
13. A. Mishchenko, S. Chatterjee, R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis", In Proc. of the 43rd Annual Design Automation Conference (DAC'06), CA, USA, July, 2006, pp. 532-536
14. 7 Series FPGAs Configurable Logic Block, User Guide, UG474 (v1.8) September 27, 2016
15. I. Lemberski, LUT-oriented dual-rail quasi-delay insensitive logic synthesis, Electronics Letters, vol. 50, Issue 7 , March 2014, pp. 503 – 505

16. E.J. Sparsø. S. Furber, Principles of Asynchronous Circuit Design, Kluwer Academic Publishers, 2001,337 p
17. N. Sretasereekul, T. Nanya, Eliminating isochronic-fork constraints in quasi-delay-insensitive circuits, Proceedings of the 2001 Asia and South Pacific Design Automation Conference, Japan, 2001, pp. 437– 442
18. I. Lemberski, A. Suponenkovs, Asynchronous Logic Design Targeting LUTs, The 7th Mediterranean Conference on Embedded Computing, June, 2018, Budva, Montenegro (MECO2018), pp. 420-425
19. I. Lemberski, V. Gopejenko, Semi-Automatic Asynchronous Logic Synthesis in XILINX: Design Flow and Case Study, IFAC-PapersOnLine, vol.52, issue 27,2019, pp. 50-55
20. https://www.xilinx.com/products/design-tools/vivado/vivado-webpack.html
21. S. Rubin, Computer Aids for VLSI Design, Reading, Massachusetts: Addison-Wesley,1987, http://www.rulabinsky.com/cavd/text/chapd.html
22. E. Sentovich at al, SIS: a System for Sequential Circuit Synthesis, Electronic research Laboratory Memorandum, No UCB/ERL M92/41, 1992, 45 p
23. I. Lemberski, P. Fišer, Multi-Level Implementation of Asynchronous Logic Using Two-Level Nodes, 4TH IFAC Workshop on Discrete-Event System Design (DesDes'09), 6-8 October, 2009, Spain, pp. 90-96
24. I. Lemberski, M. Uhanova, A. Suponenkovs, Distributed Indication in LUT-Based Asynchronous Logic, IFAC-PapersOnLine, vol.52, issue 27,2019, pp.257-264