# An Efficient Flip-Flops Matching Engine

Solaiman Rahim, Jerome Rampon, Bruno Rouzeyre, Lionel Torres

# An Efficient Flip-Flops Matching Engine


| Solaiman Rahim | Jerome Rampon | Bruno Rouzeyre | Lionel Torres |
|---|---|---|---|
| **rahim@synplicity.com** | **jerome@synplicity.com** | **rouzeyre@lirmm.fr** | **torres@lirmm.fr** |
| LIRMM – Synplicity | Synplicity | LIRMM | LIRMM |
| Montpellier, FR | Montpellier, FR | Montpellier, FR | Montpellier, FR |



## Abstract

*Generic algorithms for sequential equivalence checking are computationally expensive because they are based on state space traversal. This is the main reason why commercial tools often use combinational equivalence checking techniques to verify sequential designs. This approach consists in identifying potential equivalent flip-flops or nets in the two designs under verification. This is called the matching step. Due to sequential optimizations performed during synthesis, which can remove, merge, replicate or retime flip-flops, this matching step can be very complex and incomplete. Moreover if the matching is incomplete, even if a fast and efficient SAT solver is used during the combinational equivalence-checking step, this kind of approach may fail. In this paper, we present a complete matching engine, which is able to handle optimized circuit and don't care conditions. The efficiency of the proposed engine is confirmed by experimental results on retimed and optimized circuits.*


## 1. Introduction

In order to avoid the CPU time explosion due to state space traversal, a common and practical verification approach to check the functional equivalence of two sequential designs is to map the sequential equivalence-checking problem into a combinational equivalence one. This is a two steps approach. The first step consists in finding a matching between the flip-flops (FFs for short) in the reference and in the implementation design. The second step is to verify the equivalence of the corresponding combinational blocks resulting of the matching using combinational formal techniques [1][2][3].

This verification method is efficient and especially applicable when only combinational synthesis techniques are used for optimization. Indeed if sequential optimizations (e.g. retiming, replication, merge, redundancy removal…) are performed during synthesis, the number and the input function of FFs may change. Thus the matching step can be very complex and may lead to an incomplete FFs matching as described in [4]. The used of a fast and efficient SAT solver for the combinational verification step could be inefficient for this type of problem. Experiments with industrial tools based on this method show that if two combinational blocks are found to be inequivalent, this is often due to an incomplete matching rather than a real bug in the circuit (False negative problem). It is thus very important to have an efficient matching technique.

The matching techniques in industrial CAD tools can be divided in two classes:

- the first one is non-functional based methods. They consist in using name comparison, simulation or structural similarities. Since, synthesis transformations may change FFs names or modify the designs structure, this method may leave a significant number of FFs unmatched or produce a wrong matching. Thus it cannot be used as an automated matching or handle sequential optimizations.
- the second class of matching techniques are functional ones. They consist in comparing Boolean functions at FFs inputs. Various techniques have been proposed in [5][6][7] based on induction, ATPG and partitioning techniques. They are all exact methods and are able, theoretically, to verify sequential optimized circuits. But in practice, due to function representation issues, those methods cannot match all FFs in the design under verification and should be completed with non-functional methods.

Moreover the proposed algorithms cannot handle all classes of designs. For example, the method [5] cannot deal with circuits with don't cares (often referred as don't care conditions) or circuits with stuck at FFs as defined in [8]. Method in [6] considers don't cares but doesn't propose a solution for retimed circuits or circuits with stuck at FFs [8]. The ATPG approach presented in [7] is a complete one but may run out of time for large designs. A method, which combines both non-functional and functional based methods, has been proposed in [9]. Its limit is reached with designs in which there is not a one

to one correspondence between the FFs in the reference and implementation (which may not be the case for sequential optimized circuits). The method presented in [15] can also be used to match FFs using the primary output. The idea is to find a permutation on the variable of the support (which could be FFs variables) of the function of the matched primary output, which make those functions equivalents.

In this paper, we propose a combined matching engine. It produces a fast, complete and automated FFs matching in most of the cases. It is able to prove complex sequential optimizations (as merge after retiming or stuck at FFs as defined in [8]) and avoid false negative problems. Our experimental results show that the procedure is able to prove 21/23 optimized industrial circuits where industrial equivalence checking tool could only proved 14 of them. The main reasons of the efficiency of our approach are:
- the use of an incremental approach which is able to handle merge, replication, don't care conditions, retiming, complex redundancy removal or any combination of them.
- Experiments shows that the rate of flips-flops matched with functional part is in average very important.

This paper is organized as follows. In section 2, we introduce some notions and definitions. The section 3 discussed about the overall procedure of our matching engine. Section 4 described the core engine of our functional matching algorithm and show its coupling with our redundancy removal published in [8]. In section 5, we present a basic algorithm to get an initial state to perform inverse matching (e.g. match all FFs equivalent modulo the inversion). Section 6 explains the non-functional part of our engine. Section 7 details our technique for retimed designs. The efficiency of our engine is presented in section 8 by applying it on retimed and optimized both industrial and ISCAS89 benchmarks.

## 2. Notations

The input of the matching engine is the product machine of the two designs under verification.

***Definition 1:***
The input space is noted X, the state space S, the initial state So $\in$ S, the next state function: $\Delta$: S $\times$ X $\rightarrow$ S, and the output function: $\Lambda$: S $\times$ X $\rightarrow$ B, where B = {0,1} denotes the set of Boolean values.

***Definition 2:***
The number of FFs of the product machine is noted N.

***Definition 3:***
$F_i$ and $V_i$ denote respectively the input function and the output variable of the flip-flop $R_i$.

***Definition 4:***
The set of don't care variables is noted D = $\{Dc_1,Dc_2,\ldots Dc_j\}$. The don't care ($Dc_i$) variables are used to specify all unspecified output values of a function.

***Definition 5:***
The inclusion operator $\subseteq$ between two functions f ($X_i$,$V_i$) and g($X_i$,$V_i$,$Dc_i$) is defined as follow:

$$f \subseteq g \text{ iff } \forall(X_1,..X_m)\ \forall(V_1,..V_N)\ \exists\ (Dc_1,..Dc_j) \,/\, f = g.$$

***Definition 6:***
Let R be the set of all FFs of the product machine $\{R_1,R_2,\ldots R_N\}$. The goal of the matching is to find in R the sets of all the functionally equivalent $R_i$ (functional matching) and the sets of all potential equivalent $R_i$ (non functional matching). Thus, the matching result can be viewed as a partitioning of the set of FFs into equivalent classes, each equivalent class representing a subset of (functionally or structurally) matched FFs. Essentially, the matching process is a merging process, i.e. initially the set R is partitioned into N classes $\{C_1, C_2,\ldots C_N\}$ where $C_i = \{R_i\}$. When two FFs match, their classes are merged. We note MS the matching status. MS denotes the state of the matching during our procedure. Initially no FFs have been matched and the matching status is:

$$MS = \bigcup_{i=0}^{N} C_i$$

***Definition 7:***
If a flip-flop $R_i$ match with a flip-flop $R_j$, $C_i$ and $C_j$ are merged into a single class $C_i$ such as $C_i \cup C_j = C_i$ where $C_i = \{i,j\}$ and $V_i = V_j$.

***Definition 8:***
The equivalence relation Meq (Match equivalent) is introduced such as $R_i$ Meq $R_j \Leftrightarrow F_i \subseteq F_j$. Meq is used to partitioned R during functional matching. The number of class created by Meq is noted Nceq.

***Definition 9:***
The equivalence relation Mpeq (Match potential-equivalent) is introduced such as $R_i$ Mpeq $R_j \Leftrightarrow$ $F_i$ is potentially equivalent to $F_j$. Mpeq is used to partitioned R during non functional matching.

The number of class created by Mpeq is noted Ncpeq.

***Definition 10:***
The number of FFs unmatched is noted Nunm. All classes with $|C_i| = 1$ correspond to an unmatched flip-flop. The number of FFs redundant is Nred.

***Definition 11:***
The transitive fanout (TFO) of a signal is the set of FFs and primary output structurally depending of this signal.
The transitive fanin (TFI) of a signal is the set of FFs and primary input present in the support of the signal.

***Definition 12:***
A synchronization sequence of a circuit is a sequence of primary input, which initializes all the FFs of the circuit. Finding an initial state of a circuit consists in finding a synchronization sequence.

# 3. The overall matching procedure

The overall matching procedure is presented in figure 1. The Step 0 of the procedure starts to find a state So. The state So is needed to be able to perform inverse matching during the functional matching (e.g. match all equivalent FFs modulo the inversion). The general method to compute an initial state as presented in [13] is very time consuming for large circuit. A basic heuristic has been developed to compute quickly an initial state. It is presented in section 5. The initial state could be given by the user when the heuristic failed.

The Step 1 of the procedure consists in running our functional matching. To be able to find functional equivalent FFs, we have used and extended the technique presented in [5]. It has been extended to handle don't care conditions and improved by decreasing its CPU time consumption. The matching status is then the union of the new class created by the functional matching using the Meq relation (which represents the matched FFs) and the class with a single flip-flop (which represents the unmatched one) and:

$$MS = \left( \bigcup_{i}^{Nceq} C_i \right) \cup \left( \bigcup_{j}^{Nunm} C_j \right)$$

Then the redundancy removal in [8] is applied. If redundant FFs have been found, functional matching is run again until no new improvement has been reached. The redundant FFs are removed from the set of unmatched one. The matching status becomes:

$$MS = \left( \bigcup_{i}^{Nceq} C_i \right) \cup \left( \bigcup_{j}^{Nunm - Nred} C_j \right)$$

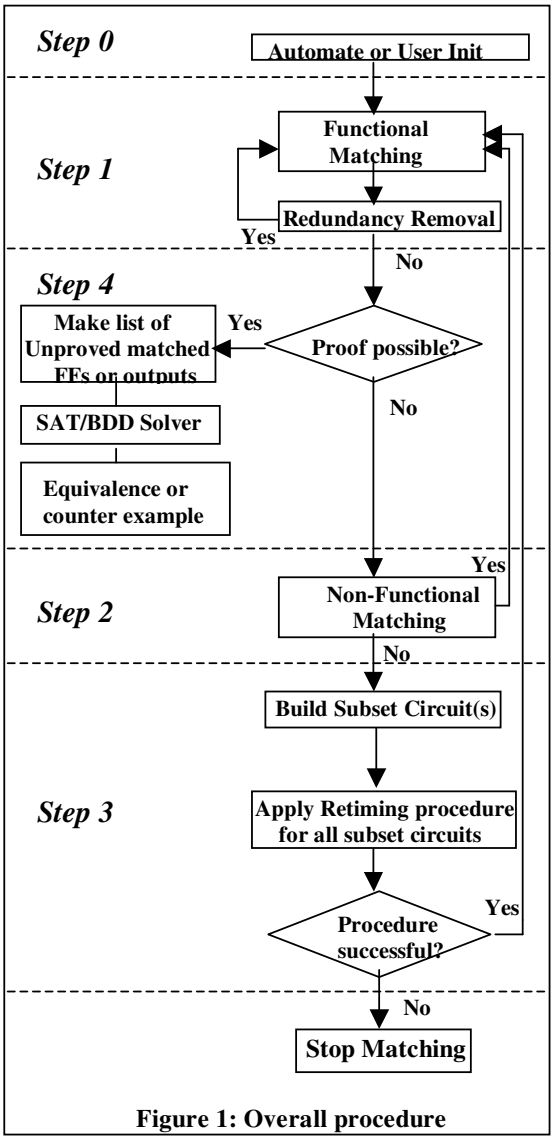



**Figure 1: Overall procedure**

The Step 2 of the procedure is the non-functional matching using the Mpeq relation. If new classes are created (i.e new matching has been found), all the procedure is launched again until no new improvements has been detected. Note that all FFs in the classes created should be proved in a separate process. The matching status is then the union of matched (functionally and structurally) FFs with the unmatched one. MS becomes:

$$MS = \left(\bigcup_{Nceq} C_i\right) \cup \left(\bigcup_{Ncpeq} C_j\right) \cup \left(\bigcup_{Nunm - Nred} C_k\right)$$

Step 3 is used to prove retimed part of the circuits. The retimed subset part of the circuit is extracted. Those parts come from the unmatched FFs. Then methods as [5][10][11][12] are applied depending on the structure of the subset circuits (pipelined or not). As the output of the subset circuits could be FFs, matching those circuits may lead to new matching. In this case, the overall procedure is launched again. The retiming method is presented in section 7

The overall procedure stops when the proof is possible (Step 4). The proof is possible when all non-redundant FFs have been matched and all outputs of the subsets circuits built by the retiming procedure could be proved. Then all primary outputs and structurally matched FFs are proved in a separate process. The combinational techniques used for this proof is more powerful than the one used in the functional matching.

# 4. Functional matching

The goal of the functional matching is to find all functional equivalent FFs of the product machine. Then, match them with the equivalence relation Meq. In this section, we present our partitioning technique in order to find all functional equivalent, included or inverted FFs of a design. It is an extension of the techniques presented in [5]. It has been extended to handle don't care conditions and improved by decreasing its CPU time consumption.

First, the technique partitions all $F_i$ in two partitions regarding the initial state of $R_i$. Then, the partitions are split iteratively as follows. The output variables of each $R_i$ are respectively set to V and $\neg V$ (regarding the initial state) and their input functions $F_i$ are computed with those variables. Then all equal, included or inverted function $F_i$ are put in a new partition represented by a new output variable $V_k$ ($\neg V_k$ for inverted functions) and the input functions $F_i$ are re-computed. Each function, which could not be computed due to function representation issues, is put in a different single partition. The process continues until it reaches a fixpoint i.e. when no new partitions are created. The theory of this method has been presented in [5] and is the same introducing the inclusion checking. At the fixpoint, all $F_i$ in the same partition are equivalent, included or inverted. Thus the corresponding FFs $R_i$ can be easily matched with the equivalence relation Meq. The matching status is then:

$$MS = \left(\bigcup_{Nceq} C_i\right) \cup \left(\bigcup_{Nunm} C_j\right)$$

The unmatched FFs may come from redundant FFs as explained in [8]. They can also come from FFs, whose input function could not be computed during the process.

The example on figure 2 illustrates the method. At the beginning no FFs have been matched, MS is represented in figure 3. The set $F = \{F_1,F_2,F_3,F_4,F_5,F_6,F_7\}$. Considering the initial state (**init**), $F_1=F_6=F_7=0$ and $F_2=F_3=F_4=F_5=1$. The set F is split in two partitions ($P_0$, $\neg V$) and ($P_1$, V) (where $P_0=\{F_1, F_6, F_7\}$ and $P_1=\{F_2, F_3, F_4, F_5\}$). After iteration1 (**iter1**) of the process $F_1 = \neg F_2$, $F_3 = F_4$, $F_4 \subseteq F_5$ and $F_6 = F_7$. The set F is refined in four partitions ($P_0$, $\neg V_1$), ($P_1$,$V_1$), ($P_2$,$V_2$), ($P_3$,$V_3$) (where $P_0=\{F_1\}$; $P_1=\{F_2\}$, $P_2 =\{F_3,F_4,F_5\}$; $P_3=\{F_6,F_7\}$. After iteration 2 (**iter2**), the relation between the functions remains the same i.e. $F_1 = \neg F_2$, $F_3 = F_4$, $F_4 \subseteq F_5$ and $F_6 = F_7$. No new partitions are created. The fixpoint is reached and the process stops. All the $F_i$ in the same partition modulo the inversion are equivalent, included or inverted. Thus, all corresponding FFs $R_i$ can be matched with the equivalence relation Meq. The matching status is then the one in figure 4.

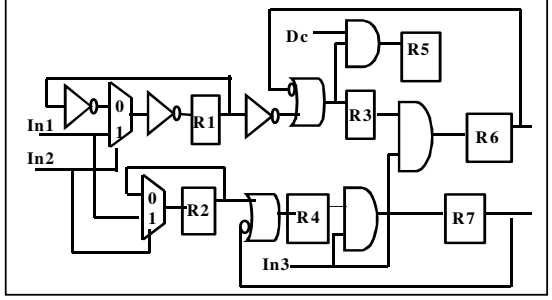
Let suppose that the initial state of the product machine is not known in the example of figure 2. At the beginning of the process, all the $F_i$ are in the same partition ($P_0$, V). The first iteration of the process gives $F_1 \neq \neg F_2$, $F_3 \neq F_4$, $F_4 \not\subset F_5$, $F_6=F_7$. The set F is refined in 6 partitions ($P_0$, $V_0$), ($P_1$, $V_1$), ($P_2$, $V_2$), ($P_3$, $V_3$), ($P_4$, $V_4$), ($P_5$, $V_5$), ($P_6$, $V_6$), (where $P_0 = \{F_1\}$, $P_1 = \{F_2\}$, $P_3 = \{F_3\}$, $P_4 = \{F_4\}$, $P_5 = \{F_5\}$, $P_6 = \{F_6, F_7\}$). The second iteration gives $F_1 \neq \neg F_2$, $F_3 \neq F_4$, $F_4 \not\subset F_5$, $F_6 \neq F_7$. The set R is refined in 7 partitions ($P_0$, $V_0$), ($P_1$, $V_1$), ($P_2$, $V_2$), ($P_3$, $V_3$), ($P_4$, $V_4$), ($P_5$, $V_5$), ($P_6$, $V_6$), ($P_7$, $V_7$ (where $P_0 = \{F_1\}$, $P_1 = \{F_2\}$, $P_3 = \{F_3\}$, $P_4 = \{F_4\}$, $P_5 = \{F_5\}$, $P_6 = \{F_6\}$, $P_7 = \{F_7\}$). At the next iteration the fixpoint is reached. All the $F_i$ are in a different partition at the fixpoint. This implies that no FFs can be matched with Meq. This shows that to now the initial state of the product machine is needed.

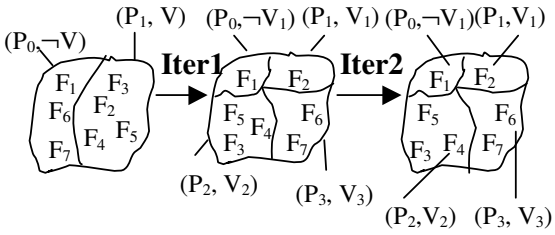
One way to speed up the process is to notice that each input function of the FFs doesn't need to be re-computed at each fixpoint iteration. Indeed for example, the input functions of FFs depending only on the primary inputs don't change.

Moreover when a new class is created, only the input functions of the FFs presents in the TFO of the FFs in the new class need to be computed, other functions remains the same. Another improvement consists not in starting with only two classes, but to use random simulation to create more classes. The experimental results show that those improvements decrease drastically the CPU time.

In [8], it is explained why redundant FFs (as constant or stuck at FFs) should be removed to improve the functional matching and avoid false negative problems. The redundancy removal engine introduced in [8] is combined with the functional matching in a fixpoint algorithm. Indeed removing redundant FFs may lead to have new matching and new matching may lead to have new redundant FFs. The redundant FFs are removed from the unmatched one and the matching status is:

$$MS = \left( \bigcup_{Nceq} C_i \right) \cup \left( \bigcup_{Nunm - Nred} C_j \right)$$




**Init:** $F_1$=0, $F_2$=1, $F_3$=1, $F_4$=1, $F_5$=1, $F_6$=0, $F_7$=0
**Iter1:** $F_1 = \neg((V \wedge \neg In2) \vee (In1 \wedge In2))$,
$F_2 = (V \wedge \neg In2) \vee (In1 \wedge In2)$,
$F_3$=V, $F_4$=V, $F_5$=V∧Dc,
$F_6$=V∧In3, $F_7$=V∧In3.
**Iter2:** $F_1 = \neg((V_1 \wedge \neg In2) \vee (In1 \wedge In2))$,
$F_2 = (V_1 \wedge \neg In2) \vee (In1 \wedge In2)$,
$F_3 = V_1 \vee \neg V_3$, $F_4 = V_1 \vee \neg V_3$, $F_5 = (V_1 \vee \neg V_3) \wedge Dc$,
$F_6 = V_2 \wedge In3$, $F_7 = V_2 \wedge In3$

**Figure 2: Example of functional matching**

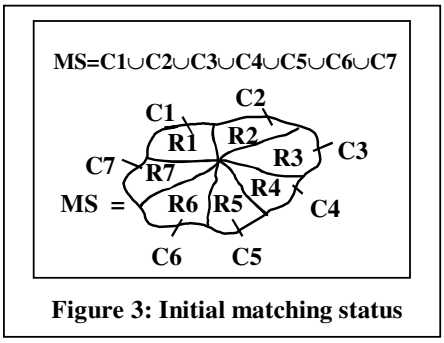



**Figure 3: Initial matching status**

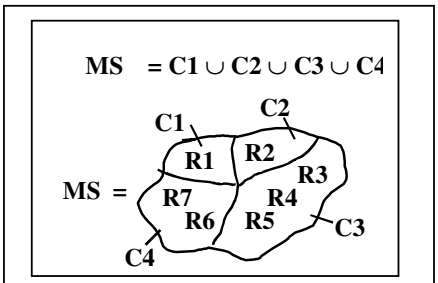



**Figure 4: Final matching status**

For some particular cases, a redundant state removal has been added in our redundancy removal engine. Indeed matching with a one to one correspondence n flip-flop of the reference design with n FFs of the implementation implies that the output value of the n FFs can take $2^n$ values. This can be false if the fsm corresponding to the n FFs have a one hot encoding. This implies that there is $2^n - n$ unreachable values for the output function of the n matched FFs. Let suppose now that FFs $R_{ref}$ have the n FFs of the reference in its support and $R_{impl}$ the n FFs of the implementation. $R_{ref}$ and $R_{impl}$ may not match because their input functions differ in the $2^n - n$ unreachable values.

The problem is illustrated on figure 5. Functional matching matched $R_1$ with $R_5$, $R_2$ with $R_6$, $R_3$ with $R_7$ and left unmatched $R_4$ and $R_8$ because $F_4 \neq F_8$. Match ($R_1$,$R_2$,$R_3$,$R_5$,$R_6$,$R_7$) implies automatically to suppose there is a full range in the output function of those FFs (e.g. matching assume that there is $2^3$ possible output values). This results to have F4≠F8. Applying our redundant state removal, it can be seen that the output value of (R1,R2,R3) can never take the value (1,1,1). This implies directly that F4 = F8 and F4 is functionally matched with F8.

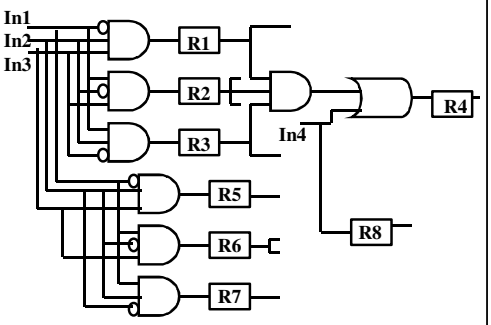



**Figure 5: Example of circuit with redundant states**

# 5. Initial State

As discuss in section 4, the computation of an initial state So is needed for functional matching. General method presented in [13] is based on state traversal and could be very time consuming. In this section, a basic heuristic to compute a synchronization sequences is presented. Experiments done show clearly that for 90% of the industrial benchmark tested, the heuristic could find the right synchronization sequence in few seconds.

The heuristic consists in unrolling the product machine as described in [14]. The unrolling technique consists in computing the product machine at time frame t, then at time t+1…. The notion of time frame model of the product machine is described in [5].

$F_i(t)$ is the function $F_i$ at time frame t, $F_i(t+1)$ at time t+1…..such as $F_i(t)$ is a function of $(V_i, X_t)$, $F_i(t+1)$ a function of $(V_i, X_t, X_{t+1})$…

The time frame at the beginning of the process is noted t. The heuristic unrolls the product machine until time frame T. T is the first time frame where all $F_i(T)$ can be justified separately to a value with a primary inputs sequence between time t and T $(X_t, X_{t+1}, \ldots X_T)$. It is not sure that all FFs can be justified globally at this time frame T. For instance, at time T, two sequences S1 and S2 are found that justifies $F_1(T)$ (resp $F_2(T)$) but not $F_2(T)$ (resp $F_1(T)$). This is why our method is a heuristic and not a general method. The product machine has a synchronization sequence. This implies that here is at least a time frame Tinit where all FFs can be globally initialized. This guarantees to find a time frame T. The weakness of our method is that we may have Tinit > T. But if at time T, there is a sequence, which initializes all FFs (i.e T=Tinit), our procedure guarantees to find it.

Finally, a pattern input sequence, which initializes the maximum of $F_i(T)$, is found using TPG techniques. This input sequence has a length of T-t. If this input sequence could initialize all the N FFs, the synchronization sequence has been found.

The heuristic is illustrated on the design of figure 2. Each flip-flop has a different output variable as at the beginning of the matching e.g. $V_1$, $V_2$….

The unrolling of the product machine gives:

At time frame t:

$F_1(t) = \neg((\neg V_1 \wedge \neg In2_t) \vee (In1_t \wedge In2_t))$

$F_2(t) = (V_2 \wedge \neg In2_t) \vee (In1_t \wedge In2_t)$

$F_3(t) = \neg V_1 \vee \neg V_6$

$F_4(t) = V_2 \vee \neg V_7$

$F_5(t) = (\neg V_1 \vee \neg V_6) \wedge Dc$

$F_6(t) = V_3 \wedge In3_t$

$F_7(t) = V_4 \wedge In3_t$

At time frame t, $R_3$, $R_4$, $R_5$ cannot be justified with the primary inputs, the product machine is computed at next time frame.

At time frame t+1:

$F_1(t+1) = \neg((\neg F_1(t) \wedge \neg In2_{t+1}) \vee (In1_{t+1} \wedge In2_{t+1}))$

$F_2(t+1) = (F_2(t) \wedge \neg In2_{t+1}) \vee (In1_{t+1} \wedge In2_{t+1})$

$F_3(t+1) = \neg F_1(t) \vee \neg F_6(t) = ((\neg V_1 \wedge \neg In2_t) \vee (In1_t \wedge In2_t)) \vee (\neg (V_3 \wedge In3_t))$

$F_4(t+1) = F_2(t) \vee \neg F_7(t) = ((V_2 \wedge \neg In2_t) \vee (In1_t \wedge In2_t)) \vee (\neg(V_4 \wedge In3_t))$

$F_5(t+1) = (\neg F_1(t) \vee \neg F_6(t)) \wedge Dc = ((\neg V_1 \wedge \neg In2_t) \vee (In1_t \wedge In2_t)) \vee (\neg (V_3 \wedge In3_t)) \wedge Dc$

$F_6(t+1) = F_3(t) \wedge In3_{t+1}$

$F_7(t+1) = F_4(t) \wedge In3_{t+1}$

At time frame t+1, all the FFs can be justified with primary inputs ($In1_t$, $In2_t$, $In3_t$, $In1_{t+1}$, $In2_{t+1}$, $In3_{t+1}$), we set T = t+1.

At time frame T, using TPG techniques an input sequences which initialized all the FFs is found. For example a synchronization sequence can be:

($In1_t$, $In2_t$, $In3_t$, $In1_{t+1}$, $In2_{t+1}$, $In3_{t+1}$) = (1,1,0,1,1,0), this implies that an initial state of the circuit could be:

$R_1$=0, $R_2$=1, $R_3$=1, $R_4$=1, $R_5$=1, $R_6$=0, $R_7$=0.

# 6. Non-functional matching

As explained in section 4, functional matching may leave some FFs unmatched due to the function representation issue. In this case, non-functional matching should be combined with functional matching. Indeed matching FFs with non-functional algorithm may lead to new functional matching as described in figure 6. Non-functional techniques are a combination of name

matching simulation, structural matching and support matching.
On the example of figure 6, the output function of the pin 31 of the 32x32 multiplier cannot be computed with BDD techniques. Functional matching is not able to match $R_3$ with $R_7$ and automatically $R_4$ is not matched with $R_8$. Applying non-functional matching, $R_3$ is matched with $R_7$. Then applying functional matching $R_4$ is matched with $R_8$.
The matching status after non-functional matching is:

$$MS = \left(\bigcup_{Nceq} C_i\right) \cup \left(\bigcup_{Ncpeq} C_j\right) \cup \left(\bigcup_{Nunm - Nred} C_k\right)$$

The set $\bigcup_{Ncpeq} C_j$, which comes from non-functional matching, should be proved. Indeed non-functional matching gives only the assumptions that FFs are equivalent. Those assumptions should be proved using more powerful combinational equivalence checking techniques than the one used in the functional matching. On the figure 6, $R_3$ should be proved with $R_7$ to validate the matching. Combinational technique as [3] can be used for this purpose.

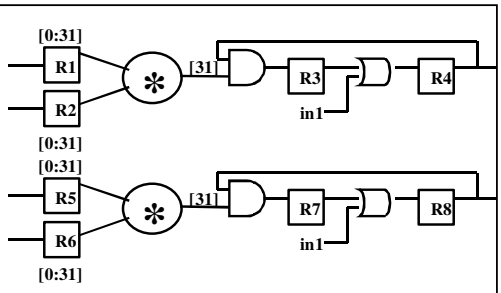



**Figure 6: Example of non-functional matching**

# 7. Retiming procedure

Retiming is one major problem for the matching step. The input function of the retimed FFs is changed during retiming. Thus, it is not possible to find any matching for those FFs using functional or structural matching. Efficient methods as [5][10][11][12] exist to prove retimed circuits. The method [5] uses induction to find a functional nets matching to prove retiming. The method in [10] uses combinational techniques to do it. This method reintroduces the input function of the retimed FFs to the output at next clock cycle. In this section, we present our retiming procedure based on [5] and [10]. The difference with the previous approach is that those methods are applied in subset circuits and not in all the circuit. The other difference is that our retiming procedure chose the best approach regarding the subset circuit structure. [10] is applied when the subset circuit is a pipeline one. [5] is applied for no pipeline one.
The retiming procedure is applied on the last set of the matching status $\bigcup_{Nunm - Nred} C_k$. This set corresponds to the unmatched FFs after functional and non-functional matching and redundancy removal.
To be able to prove complex retiming, the functional matching described in section 4 is extended to net (e.g. not only FFs are matched but also nets).
The construction of the retimed subset circuits consists in finding all matched FFs, nets, primary input and primary output present in the TFO and TFI of all unmatched FFs. The primary inputs of the subset circuits are the matched FFs, nets and the primary input present in the TFI. The new primary outputs are the matched FFs, nets and primary output present in the TFO. Then if the subset circuit is a pipeline circuit, technique as [10] is applied. If the circuit has sequential loops, technique as [5] is applied. Applying retiming methods as [5][10] on subset circuits and not on the product machine make our procedure efficient.
A retimed design (7.a) and its subset circuit (7.b) are presented on figure 7. In the original design (7.a), $R_5$ has been retimed in $R_{11}$ and $R_{12}$. Thus, functional matching is not able to match $R_6$ with $R_{13}$. This implies automatically to not match ($R_3$, $R_4$, $R_9$, $R_{10}$). However ($R_1$,$R_2$,$R_7$,$R_8$) only depends of the primary inputs, they are matched. The retiming procedure builds the subset circuit (7.b) by replacing all FFs in the TFI of the unmatched FFs by primary inputs. The circuit created is not a pipeline one. The method in [5] is applied. This results to match the net n with R5. This leads to $F_6=F_{13}=\neg V_5$ and implies to match $R_6$ with $R_{13}$. Then automatically $R_3$ is matched with $R_9$ and $R_4$ with $R_{10}$. Thus, the outputs of the subset circuit are proved.

Another example of the retiming procedure is given figure 8. In the original circuit (8.a), $R_2$ has been retimed backward then merged with $R_1$. $R_1$, $R_4$ and the network n are matched together by functional matching because they have the same input function. Then the subset circuit is built (8.b). The method [10] is applied on the pipeline subset circuit. The input function of the retimed

FFs $R_2$ and $R_4$ is reintroduced to their output at next clock cycle. This results to match $R_3$ with $R_6$. Thus, the outputs of the subset circuit are proved.

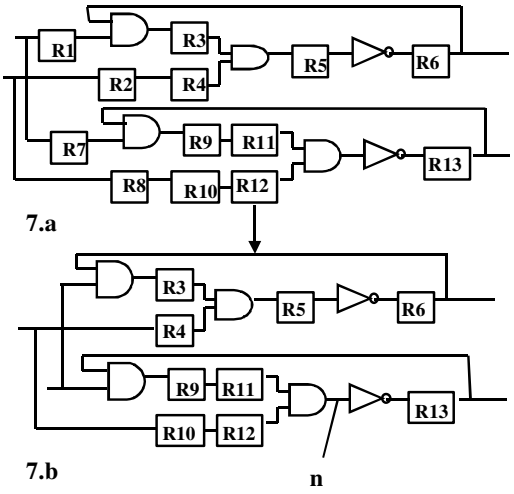



**Figure 7: Example of creation of the subset circuit**

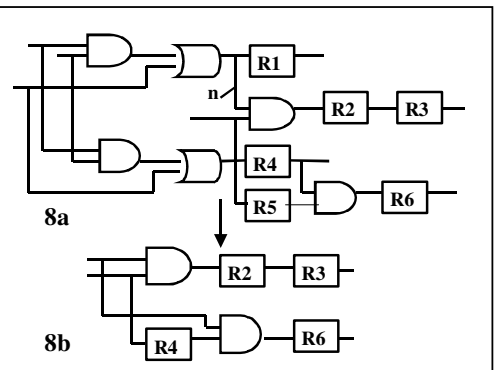



**Figure 8: Example of creation of the subset circuit**

# 8. Experimental results

This section reports the experimental results of our procedure on circuits synthesized and sequentially optimized using Synplify Pro 7.3 tool (www.synplicity.com). We use a corporate BDD package for the functional matching. The FFs matched with non-functional techniques are proved using an improved technique of [3]. All tests are performed using a Dell Precision 833Mhz workstation with a memory limit of 1Gb. We compare our method with the last VIS version on ISCAS89 circuits (Table1) and with an industrial sequential equivalence-checking tool on some industrial designs (Table2). For all tests, we have resumed the matching (complete/incomplete), the rate of matched FFs by the functional and non-functional part of the procedure, the cpu time and if the test is proved or not. The time limit has been set to 1 hour for the comparison with VIS on the ISACS89 benchmarks. For the comparison with the industrial tool, the time limit has been set to 3 days. Note that VIS runs out of time for industrial circuits.

Table 1 shows that our procedure passes quickly 24 tests over 25 where VIS passes only 17 tests. The benchmarks, which failed in Table1 using VIS, are due to that the time limit has been reached. The average rate of FFs matched functionally is 91% and non-functionally 9%. The circuit S38584 does not pass with our procedure because it cannot produce a complete matching for this circuit. Indeed two subset circuits from the retiming procedure are not proved due to function representation issues (BDD blow-up).

Table 2 shows that the proposed matching engine passes 21 test over 23 industrial circuits where the industrial tool passes only 14 tests. The average rate of FFs matched functionally is 69% and non-functionally 31%. The non-passing tests using the industrial tool come from the fact the tests are proved inequivalent due to an incomplete matching (False negative problem). The circuit I17 produces a false negative with our procedure due to an incomplete matching. Our procedure does not find a FFs replication because of BDD size limit. The circuit passes if the replication information is given using the user matching input of our procedure. The circuit I21 does not pass with our method because it does not complete with the time limit of 3 days. This circuit passes using the industrial tool. Our belief is that the industrial tool has a better and faster SAT/BDD solver that ours. This is confirmed with the circuit I20 where both our procedure and the industrial tool pass it but the industrial tool passes it twice faster (2378s) than our method (5288s). It can be noticed that both industrial tool and our matching procedure produce false negative with an incomplete matching. This confirmed the importance of a complete matching for the sequential equivalence-checking problem.

Comparison done with [5], [7] and [14] show that we obtained better performances, but we could not optimize the circuits using the same tools and we did not know the exact machine used for the testing. To give an idea, the circuit S838 is proved on 55.6s with apparently the same optimizations

(same number of FFs after optimizations) with [5] on a HP9000/755, our method proved it on 4.5s. The circuit S1196 is proved in 159.3s with apparently the same optimizations with [7] on a Sun machine, our method proved it on 1.5s. The circuit S35932 with apparently the same optimizations is proved on 75s with [14] on a Sun-sparc5 machine, our method proved it on 9.1s.

# 9. Conclusion

To conclude, the experimental results show a clear advantage to our procedure. The results also show that our procedure is more reliable that an industrial tool (less false negative). These better results are due to several factors. The first one is our incremental approach (every time that new matching is found, all the procedure is run again). The second one is the use of several engines (functional matching, non-functional matching, complex redundancy removal, retiming procedure), which can produce a complete matching in most of the cases. The advantage of our method is also due to our functional matching which handles 69% of the matched FFs for the industrial circuit and 91% for the ISCAS89. Thus, it finds more complex sequential optimizations and produces a complete and better matching than the industrial tool. Our SAT/BDD solver could not handle the circuits, which do not pass with our method (BDDs become too large). This problem is related to our SAT/BDD solver, which is not optimized.

**References**


[1] D.Brand, "Verification of large synthesized designs" in IEEE International Conference on Computer-Aided Design (ICCAD), 1993, pp.534-537

[2] S.M.Reddy, W.Kunz and D.K. Pradhan, "Novel verification framework combining structural and OBDD methods in a synthesis environment" in Design Automation Conference (DAC) 1995, pp414-419

[3] A.Kuelmann and F.Krohm, "Equivalence checking Using Cuts and Heaps" in Design Automation Conference (DAC) 1997, pp, 263-268

[4] S.Rahim, B.Rouzeyre, L.Torres, J.Rampon "Loop problem in sequential equivalence checking" in SAME 2002, pp 52-58

[5] C.vanEijk: Formal Methods for the Verification of Digital Circuits, *Ph.D. Thesis of the Eindhoven University of Technology, Eindhoven, The Netherlands, September 1997*

[6] J.R. Burch and V.Singhal, "Robust Latch Mapping for Combinational Equivalence Checking" in ICCAD, 1998, pp 563-569

[7] S-Y. Huang, K-T.Cheng, K-C Chen and U.Glaeser, "An ATPG-Based Framework for Verifying Sequential Equivalence", in ITC, 1996, pp.865-874

[8] S.Rahim, B.Rouzeyre, L.Torres, J.Rampon "Matching in the presence of don't cares and redundant elements for sequential equivalence checking" in HLDVT, 2003, pp129-135

[9] ] Anastasakis, Damiano, Tony Ma, Stanion, "A Practical and Efficient Method for Compare-point Matching" *DAC 2002, pp 305-310.*

[10] Ranjan, Singhal, Somenzi, Brayton, "Using Combinational Verification for Sequential Circuits", *in DATE. 1999, pp. 138 - 143*

[11] S-Y. Huang, K-T.Cheng, K-C Chen, "On Verifying the Correctness of Retimed Circuits", Proce. Great Lakes Symp, on VLSI, pp.277-281,1996

[12] L.Stok, I.Spillinger, and G.Even, "Improving Initialization throught Reversed Retiming", Proc. ED&TC, pp, 150-154, 1995

[13] Carl Pixley, She-wong Jeong, Gary D.Hachtel "Exact Calculation of Synchronization Sequences Based on Binary Decision Diagrams" in DAC 1992, pp 620-623

[14] Shi-Yu Huang, Kwang-Ting (Tim) Cheng, "AQUILA: A Local BDD-based Equivalence Verifier" in Formal Equivalence Checking and Design Debugging pp90-pp109, Kluwer Academic Publishers.

[15] Janett Mohnke, Paul Molitor, Sharad Malik, "Application of BDDs in Boolean matching techniques for formal logic combinational verification" in STTT, 2000

**Table 1.** Experimental results on sequential and logically optimized ISCAS89 circuits

| Circuit | #FFs | VIS | | Our procedure | | | | |
|---|---|---|---|---|---|---|---|---|
| | orig./opt | Time | Pass | Time(s) | Pass | Matching | FM (%) | NFM (%) |
| S208 | 8/12 | 0.8 | Yes | 0.9 | Yes | Complete | 100 | 0 |
| S298 | 14/29 | 3.2 | Yes | 0.9 | Yes | Complete | 100 | 0 |
| S344 | 15/27 | 18 | Yes | 1.4 | Yes | Complete | 100 | 0 |
| S382 | 21/24 | 24 | Yes | 0.9 | Yes | Complete | 100 | 0 |
| S386 | 6/6 | 10 | Yes | 0.9 | Yes | Complete | 100 | 0 |
| S420 | 16/20 | 82 | Yes | 0.9 | Yes | Complete | 100 | 0 |
| S444 | 21/29 | 15 | Yes | 1.4 | Yes | Complete | 100 | 0 |
| S510 | 6/12 | 2932 | Yes | 1.5 | Yes | Complete | 100 | 0 |
| S526 | 21/32 | 76 | Yes | 1.4 | Yes | Complete | 100 | 0 |
| S641 | 19/17 | 4 | Yes | 1.4 | Yes | Complete | 100 | 0 |
| S713 | 19/17 | 5 | Yes | 1.4 | Yes | Complete | 100 | 0 |
| S832 | 5/5 | 236 | Yes | 1.4 | Yes | Complete | 100 | 0 |
| S838 | 32/74 | - | No | 4.5 | Yes | Complete | 88 | 12 |
| S953 | 29/62 | 186 | Yes | 1.5 | Yes | Complete | 100 | 0 |
| S1196 | 18/18 | 3.8 | Yes | 1.5 | Yes | Complete | 100 | 0 |
| S1238 | 18/18 | 4.3 | Yes | 1.75 | Yes | Complete | 92 | 8 |
| S1423 | 74/154 | - | No | 27.3 | Yes | Complete | 78 | 22 |
| S1488 | 6/6 | 1.1 | Yes | 1.4 | Yes | Complete | 100 | 0 |
| S1494 | 6/6 | 0.9 | Yes | 1.5 | Yes | Complete | 100 | 0 |
| S5378 | 179/253 | - | No | 5.2 | Yes | Complete | 78 | 22 |
| S9234 | 160/62 | - | No | 2.3 | Yes | Complete | 82 | 18 |
| S13207 | 648/431 | - | No | 3.1 | Yes | Complete | 88 | 12 |
| S15850 | 563/304 | - | No | 11 | Yes | Complete | 69 | 31 |
| S35932 | 1728/1728 | - | No | 9.1 | Yes | Complete | 74 | 26 |
| S38584 | 1301/1406 | - | No | 68 | No | Incomplete | 36 | 64 |

**Table 2.** Experimental results on sequential and logically optimized industrial circuits

| $I_i$ | Size | #FFs | Industrial tool | | | Our procedure | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | (K) | orig./opt | Time | Pass | Matching | Time | Pass | Matching | FM | NFM |
| I1 | 36 | 255/277 | 2.2 | Yes | Complete | 2.9 | Yes | Complete | 72 | 28 |
| I2 | 38 | 255/270 | 1.9 | Yes | Complete | 2.2 | Yes | Complete | 100 | 0 |
| I3 | 56 | 158/206 | 5.2 | No | Incomplete | 2.3 | Yes | Complete | 18 | 82 |
| I4 | 24 | 17/18 | 0.9 | Yes | Complete | 1.5 | Yes | Complete | 100 | 0 |
| I5 | 612 | 1437/1437 | 345 | Yes | Complete | 595 | Yes | Complete | 95 | 5 |
| I6 | 68 | 101/105 | 1.5 | Yes | Complete | 1.7 | Yes | Complete | 100 | 0 |
| I7 | 198 | 222/211 | 36 | No | Incomplete | 24 | Yes | Complete | 12 | 88 |
| I8 | 98 | 523/568 | 11 | No | Incomplete | 7.2 | Yes | Complete | 100 | 0 |
| I9 | 88 | 354/219 | 6.3 | No | Incomplete | 5.4 | Yes | Complete | 95 | 5 |
| I10 | 56 | 565/537 | 4.1 | Yes | Complete | 4.6 | Yes | Complete | 100 | 0 |
| I11 | 55 | 381/384 | 3.4 | Yes | Complete | 3.3 | Yes | Complete | 38 | 62 |
| I12 | 278 | 72/72 | 18.2 | Yes | Complete | 20.6 | Yes | Complete | 0 | 100 |
| I13 | 308 | 120/162 | 36.3 | No | Incomplete | 32.5 | Yes | Complete | 34 | 66 |
| I14 | 456 | 996/1251 | 78.6 | No | Incomplete | 56.4 | Yes | Complete | 100 | 0 |
| I15 | 312 | 524/611 | 24.5 | No | Incomplete | 19.7 | Yes | Complete | 100 | 0 |
| I16 | 54 | 48/52 | 3.4 | Yes | Complete | 4.2 | Yes | Complete | 21 | 79 |
| I17 | 356 | 250/246 | 345 | No | Incomplete | 222 | No | Incomplete | 0 | 100 |
| I18 | 76 | 419/439 | 12.5 | Yes | Complete | 18.3 | Yes | Complete | 76 | 24 |
| I19 | 98 | 358/369 | 14.1 | Yes | Complete | 21.2 | Yes | Complete | 82 | 18 |
| I20 | 800 | 357/360 | 2378 | Yes | Complete | 5288 | Yes | Complete | 42 | 58 |
| I21 | 950 | 347/249 | 26378 | Yes | Complete | - | No | Complete | 57 | 43 |
| I22 | 188 | 432/444 | 92.3 | No | Incomplete | 68.1 | Yes | Complete | 76 | 24 |
| I23 | 1280 | 236/237 | 36200 | Yes | Complete | 172800 | Yes | Complete | 75 | 25 |