Chapter 5

The Latch Correspondence Problem

In this chapter, we show that the signature–based methods used for solving the permutation equivalence problem can be easily applied to a problem in *sequential* logic verification. This is the problem of establishing the unknown correspondence for the latch variables (memory elements) of two sequential circuits that have the same encoding of their states. If a correspondence of the latches in the two circuits can be established, then the verification problem reduces to a combinational equivalence check on the combinational circuit defined by the latch boundaries. There are several cases in which the correspondence may exist but is unknown. For example, the names for the latches used in a specification may be different from those used in the implementation due to modifications made by synthesis tools. The combinational equivalence check on the primary output and next state functions can be done using ROBDDs. However, without the correspondence of latches we cannot check if the corresponding ROBDDs of the combinational parts of the two sequential circuits are the same. Thus, we need to establish a correspondence first, similar to the combinational permutation equivalence problem.

The application of signature based methods for this problem is straightforward: derive a signature for each latch variable in order to uniquely identify this latch. In [36], the authors have used this method to identify corresponding latch variables in order to be able to simplify the product machine traversal for sequential verification. This is especially useful when the state encoding of the two machines that have to be verified are identical, but the state variable correspondence is not known. Here, they have used signatures for input variables known from literature [9] to do this and have made the observation that sometimes the task still remains too complex. We think that the ideas presented in this thesis are able to improve upon this, since we do not just combine different input signatures but also develop novel signatures which are especially suited for the latch equivalence problem. Thus, these new signatures give better results, i.e., the set of pairs of latch variables which are candidates for correspondence are determined more precisely.

We start with a problem description in **Section 5.1**. In **Section 5.2**, we describe the differences between establishing unknown input correspondence and establishing unknown latch correspondence, introduce special signature functions for latch variables that can be easily computed on ROBDDs, and explain a solution paradigm which uses these signature functions on an example. Then, we

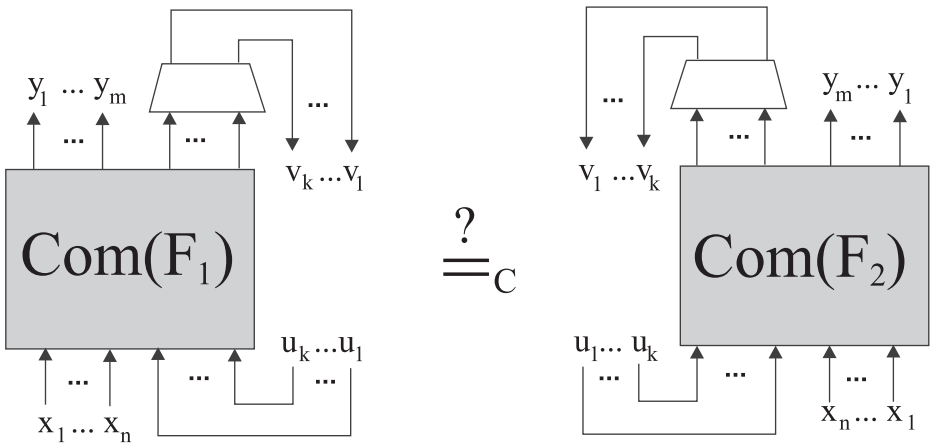

Figure 5.1: The Latch Equivalence Problem

demonstrate the utility of this approach by experimental results in **Section 5.3** and discuss the influence of symmetries in **Section 5.4**. The main results of this chapter are published in [27].

5.1 Problem Description

Let $\mathcal{F}_{n,m,k}$ be the set of all synchronous sequential circuits with n primary input variables, $X = [x_1, x_2, \ldots, x_n]$, m primary output variables, $Y = [y_1, y_2, \ldots, y_m]$, and k latch variables, $L = [l_1, l_2, \ldots, l_k]$. We can consider just the combinational part of a sequential circuit $F \in \mathcal{F}_{n,m,k}$. Therefore, let $U = [u_1, u_2, \ldots, u_k]$ be the set of latches considered as input variables of F, and let $V = [v_1, v_2, \ldots, v_k]$ be the set of latches considered as output variables of F. Then we get the combinational part of F, $Com(F)$ as a Boolean function with $n + k$ input and $m + k$ output variables and can define the *latch permutation equivalence problem.*

Definition 5.1 *The* latch permutation equivalence problem*, L_π (also refered to as the latch correspondence problem) is defined as follows:*

Let be F_1 and $F_2 \in \mathcal{F}_{n,m,k}$. Does there exist a correspondence C between the latch variables of F_1 and F_2, such that the two synchronous sequential circuits have their combinational parts equivalent using this correspondence:

$$Com(F_1) =_C Com(F_2)?$$

This is illustrated in Figure 5.1. Here, we assume that the correspondences between the primary inputs and outputs of F_1 and F_2 is known. The similarity of this problem to the combinational permutation equivalence problem is obvious. We need to find a unique and permutation independent description for each latch variable. The difference is that each latch variable has to be considered as input *and* as output variable. So let us modify the approach used to handle the combinational problem.

5.2 Signatures

In Chapter 3, we define a *signature* as a description of an input variable that is independent of any permutation of all inputs of a Boolean function $f \in \mathcal{B}_{n,1}$. Thus, it can help to solve the combinational permutation equivalence problem, P_π, as follows. With the help of signatures each input variable can be identified independent of permutation, i.e., any possible correspondence between the input variables of two Boolean functions is restricted to a correspondence between variables with the same signature. So, if each variable of a Boolean function f has a unique signature, then there is at most one possible correspondence to the variables of any other Boolean function.

This concept can be used to attack the *latch correspondence problem*, L_π. Here we need to test whether there exist a correspondence between the *latches* of two synchronous sequential circuits, such that the combinational logic used to define them is equivalent under this correspondence.

Let us characterize this new problem and underline the differences from P_π. We assume that the correspondence between the primary inputs of the two circuits is known. So we have the first piece of additional information in comparison to P_π. Similar to P_π, we also assume that we know the correspondence between the primary outputs. These outputs are Boolean functions that depend on the primary input variables as well as the input variables due to the latches. So, we can use the primary outputs to compute signatures for the latch input variables. This is done in exactly the same manner as for the combinational problem P_π. In the following, we will call those signatures *input signatures* and make some remarks regarding them in **Section 5.2.2**. However, latches appear not only as input variables, but also as output variables. So, the most important difference between the combinational problem P_π and the sequential problem L_π is that we can use *latch output signatures* in addition to identify the latches independent of permutation. A latch output signature is a description of a latch, considered as an output variable, that is independent of the permutation of the latches as output as well as input variables. We will explain this in more detail in **Section 5.2.3**.

In **Section 5.2.4**, we demonstrate how we establish a unique possible correspondence for the latches for one of the benchmarks (see Section 5.3), namely *ex4.slif*. A general solution paradigm is given in the next section.

5.2.1 Solution Paradigm

Let us start with describing the solution paradigm on an example of two circuits, F_1 and F_2, with four latch variables $[l_1, l_2, l_3, l_4]$. Let the latches considered as input variables be the variables $[u_1, u_2, u_3, u_4]$, and the latches considered as outputs be the variables $[v_1, v_2, v_3, v_4]$.

First, we look at the latches as input variables and compute an input signature $s_I(F_1, u_i)$ and $s_I(F_2, u_i)$ for $i = 1, 2, 3, 4$ with respect to the primary outputs of F_1 and F_2 respectively. This is done in exactly the same manner as described for the combinational permutation problem P_π in

Section 3.2.2. So let us suppose that we have two lists of input signatures that contain the same elements:

$$\mathcal{L}_I(F_1) = [2, 1, 3, 2] \qquad \text{and} \qquad \mathcal{L}_I(F_2) = [1, 2, 2, 3].$$

Now, based on these signatures we can establish directly that any correspondence between the latches of F_1 and F_2 has to identify latch l_2 of F_1 with latch l_1 of F_2, and latch l_3 with latch l_4. Thus, a correspondence between the latches of F_1 and F_2 has been partially established. In this case, there is a possible correspondence between the latches of F_1 and F_2 for latch equivalence. However, both latch variables, l_1 and l_4 of F_1 could correspond to l_2 as well as to l_3 of F_2. In other words, aliasing occurs between the latch variables l_1 and l_4 of F_1 and between the latch variables l_2 and l_3 of F_2.

Since latch variables are not just input variables but also output variables, we have a second possibility to distinguish between the latches with aliasing. We can consider them as output variables and use latch output signatures to try to uniquely identify them. In our example, we have to do this for the latch output variables v_1 and v_4 of F_1, and for the variables v_2 and v_3 of F_2. So let us assume that

$$\mathcal{L}_O(F_1, [v_1, v_4]) = [4, 5] \qquad \text{and} \qquad \mathcal{L}_O(F_2, [v_2, v_3]) = [5, 4].$$

Now we are done, since we are able to establish a unique correspondence between the latch variables l_1 and l_4 of F_1 and the variables l_2 and l_3 of F_2 as well: latch l_1 of F_1 has to correspond to latch l_3 of F_2, and latch l_4 of F_1 to latch l_2 of F_2.

This gives us the only possible unique correspondence between the latch variables. Furthermore, if we require that signatures have to be elements of an ordered set, we can establish a unique permutation independent order of the latches. In the case of our example, that would be the order $[l_2, l_1, l_4, l_3]$ for the latches of F_1, and the order $[l_1, l_3, l_2, l_4]$ for the latches of F_2.

This example demonstrates that the possibility to consider latch variables as inputs as well as outputs improves our chances to get a unique correspondence between the latches for a possible latch permutation equivalence of two circuits. Of course, it is not guaranteed that such a unique possible correspondence will be obtained. It strongly depends on the input and latch output signatures that we use and on the characteristics of the latches. Thus we focus on the following questions in the rest of this chapter. What input and latch output signatures can we use? What special properties of the problem do we need to take care of? And, what special properties of the latches may cause problems for our signature approach?

5.2.2 Input Signatures

Only some modifications need to be made when considering signatures for input variables as introduced in Section 3.2.3 for use in L_π. Let us consider $F \in \mathcal{F}_{n,m,k}$ again. It has m primary output

functions, $[y_1, \ldots, y_m]$, that we can uniquely identify (see Section 5.1). These output functions depend on the primary input variables $x_1, \ldots, x_n$ and on the latch input variables $u_1, \ldots, u_k$. So we can use them step by step, starting with y_1 up to y_m, to compute input signatures for the variables $u_1, \ldots, u_k$.

Let us describe this on an example. Consider F with two input variables, x_1 and x_2, two outputs, y_1 and y_2, and four latches, l_1, l_2, l_3, and l_4. Then, the two output functions y_1 and y_2 are functions that depend on the set of input variables $[x_1, x_2, u_1, u_2, u_3, u_4]$. Now, let us use the cofactor satisfy count signature function in order to try to separate u_1, u_2, u_3 and u_4. In a first step, we compute this signature using output function y_1. Let us assume we get the partition $[\{u_1, u_3\}, u_4, u_2]$. This means that we can uniquely identify the latch input variables u_2 and u_4 and thus the latches l_2 and l_4. Furthermore, we get a unique partial order of the latches: $[\{l_1, l_3\}, l_4, l_2]$. Now, we can use output function y_2 in a second step to try to distinguish between the latch input variables u_1 and u_3 as well.

Note, that we *cannot* use the latch output variables, $v_1, \ldots, v_k$, for that purpose, since *these* outputs are not uniquely identified as yet.

5.2.3 Latch Output Signatures

Similar to an input signature, we can define a signature for a latch output variable v_i of $F \in \mathcal{F}_{n,m,k}$.

A *signature for a latch output variable* is a value, a vector of values, or a function that provides special information about this latch output variable. This information has to be independent of any permutation of the latch variables of F. In general, such a latch considered as an output depends not only on the *primary* inputs of this sequential circuit, but also on the *latches* considered as input variables. That is why a latch output signature has to be independent of the permutation of these *latch* input variables as well. Furthermore, similar to an input signature, it has to be an element of an ordered set.

Now let us develop latch output signatures that could be useful. Therefore, let us consider the example circuit $F \in \mathcal{F}_{n,m,k}$ with two input variables, x_1 and x_2, two outputs, y_1 and y_2, and four latches, l_1, l_2, l_3 and l_4, again. Suppose, we could not distinguish between the latches l_1 and l_3 considering them as input variables u_1 and u_3. Thus, we still have the partial order of the latches, $[\{l_1, l_3\}, l_4, l_2]$ (see previous subsection). Now, let us consider these latches as *output* variables of F, v_1 and v_3. These output variables represent Boolean functions, that depend on all primary input variables and on the latches considered as input variables.

For simpler notation let us denote the considered latch output variable function as f. We can apply the following kinds of latch output signatures to it.

5.2.3.1 Simple Output Signatures

Signature functions that can be directly developed by applying input signature functions as introduced in Section 3.2.3 to latch output variables are called *simple signature functions* in the

following.

Such a signature function could be for instance:

1. the satisfy count of the output function, $|f|$,
2. the vector of the cofactor satisfy count signatures of the input variables of f sorted by the size of the satisfy counts,

$$sort(|f_{x_1}|, |f_{x_2}|, \ldots, |f_{u_1}|, \ldots, |f_{u_k}|),$$

3. the breakup signature with respect to function f and origin $\mathcal{O} = [0, 0, \ldots, 0]$,

$$[|f^0|, |f^1|, \ldots, |f^{n+k}|].$$

These signature functions satisfy all necessary properties since they are output signatures for f (i.e., for v_1 and v_3, respectively) that are also independent of the permutation of the latch input variables, u_1, u_2, u_3, and u_4.

If these simple output signatures do not break the tie, we have to apply some stronger latch output signatures.

5.2.3.2 Function Signatures for Latch Outputs

Here we use the fact that we can uniquely identify the primary input variables of the circuit. In our example, these are x_1 and x_2. So, any subfunction of a latch output variable f that is independent of the *latch* input variables is a latch output signature. There are several possibilities for this kind of subfunction. Let us consider our example F with the two primary input variables, x_1 and x_2, and the four latch variables considered as inputs, u_1, u_2, u_3, and u_4, again. Subfunctions of a primary output function f of this example that only depend on x_1 and x_2 are for instance $f_{u_1 u_2 u_3 u_4}$, $f_{\bar{u_1}\bar{u_2}\bar{u_3}\bar{u_4}}$, $\forall_{u_1 u_2 u_3 u_4} f$, and $\exists_{u_1 u_2 u_3 u_4} f$.

Such a function contains special information about f, and it is independent of the permutation of the latch output variables as well as of the latch input variables of F. We call this kind of signature a *function output signature.*

In the example, we need to compute function output signatures for the latch output variables v_1 and v_3. Let us consider v_1 and denote the function computed by it as f again.

Furthermore, we can extend the idea of the function signature using exactly the idea of constructing the function signatures for input variables (see Section 3.2.3). For each latch which is uniquely identified at this point, the corresponding latch input variable can be uniquely identified as well. In our example, this is the case for the latches l_2 and l_4. So, subfunctions of v_1 and v_3 that depend on the primary inputs x_1 and x_2 *and* on the latch input variables u_2 and u_4 are latch output signatures — with one minor restriction: we need to reorder the latch input variables u_2 and u_4 independent

of permutation in these subfunctions. Therefore, let us use the order of the latches established by previously used signatures. In the case of our example, the permutation which reorders the input variables of such a subfunction would be: $\pi = (x_1, x_2, u_4, u_2)$ (see the latch order of our example in the previous section). Such a reordered subfunction of v_1 and v_3 has all the properties to be a latch output signature, and we can again try to distinguish between v_1 and v_3. This process can be iterated as long as we can uniquely identify at least one more latch. That is why we call this extended function signature the *iterating function signature*. Our practical experiments have shown that this is a very powerful signature.

5.2.3.3 Canonical Order Signature

There is another strong latch output signature. We call it the *canonical order signature*. For that we use the methods introduced to handle the combinational permutation equivalence problem. Remember, these methods can be used to establish a canonical and permutation independent ordering of the input variables of a Boolean function.

Let us consider the Boolean function f which represents a latch output variable function of F again. On the input variables of this function, the methods used in Chapter 3 and in Chapter 4 can be applied in order to find a canonical permutation independent variable ordering. Suppose $\pi \in \mathcal{P}_{n+k}$ is a permutation of the latch input variables of f which constructs this canonical order, then the canonical order signature is the following function:

$$f^{can} = f \circ \pi.$$

This function is independent of the permutation of *any* input variable of f. Thus it is a latch output signature. Note that finding this canonical ordering can be restricted to the latch input variables because the primary input variables of f are uniquely identified by assumption.

5.2.4 An Example

In this section, we illustrate our solution paradigm with benchmark *ex4.slif* from the LGSynth91 benchmark set [1]. We will not use all signature functions here. However, the general paradigm of finding a unique permutation independent order of the latch variables, as described in the previous sections, will become clear.

Benchmark *ex4.slif* is the description of a sequential circuit with 6 input variables, $v_0, v_1, \ldots, v_5$, 9 output variables, $v20.14, v20.15, \ldots, v20.22$, one clock variable, and 14 latches. Let us call these latches $l_1, l_2, \ldots, l_{14}$ in the order of their appearance in the benchmark description. For more details please see the benchmark description in the LGSynth91 set of benchmarks.

We begin with computing a *simple output signature*: the satisfy count, $|v_i|$ of the latch variable l_i considered as output v_i of the actual circuit. Here, we get the following results:

$$\begin{array}{ll} |v_1| = 0 & |v_2| = 288 \\ |v_3| = 128 & |v_4| = 128 \\ |v_5| = 192 & |v_6| = 64 \\ |v_7| = 128 & |v_8| = 128 \\ |v_9| = 96 & |v_{10}| = 128 \\ |v_{11}| = 128 & |v_{12}| = 192 \\ |v_{13}| = 64 & |v_{14}| = 128. \end{array}$$

Using these signatures we get a partial, permutation independent order of the latches:

$$l_1 \quad \{l_6, l_{13}\} \quad l_9 \quad \{l_3, l_4, l_7, l_8, l_{10}, l_{11}, l_{14}\} \quad \{l_5, l_{12}\} \quad l_2.$$

As you can see, there are three aliasing groups of latches, one of size 7 and two of size 2, for which we have to do further computations. We now consider these latches as input variables, u_i, and use an *input signature* to try to distinguish between these latches. For doing that, we take one primary output function, $v20.i$, after the other and compute the selected input signature with respect to this output function.

Let us take the *cofactor satisfy count signature* as an input signature and start with primary output function $v20.14$. We get the following results for the three aliasing groups:

*group*1	*group*2	*group*3
$\lvert v20.14_{u_6}\rvert = 256$	$\ldots_{u_3} = 0$	$\ldots_{u_5} = 128$
$\ldots_{u_{13}} = 256$	$\ldots_{u_4} = 0$	$\ldots_{u_{12}} = 128$
	$\ldots_{u_7} = 0$	
	$\ldots_{u_8} = 256$	
	$\ldots_{u_{10}} = 0$	
	$\ldots_{u_{11}} = 0$	
	$\ldots_{u_{14}} = 256$	

Based on this we see that the latches of aliasing group 1 and group 3 cannot be distinguished using the cofactor satisfy count signature with respect to primary output function $v20.14$. However, we can split up group 2 in the subgroups $\{l_3, l_4, l_7, l_{10}, l_{11}\}$ and $\{l_8, l_{14}\}$. So, we get a finer partial order for the latches:

$$l_1 \quad \{l_6, l_{13}\} \quad l_9 \quad \{l_3, l_4, l_7, l_{10}, l_{11}\} \quad \{l_8, l_{14}\} \quad \{l_5, l_{12}\} \quad l_2.$$

At this point we have four aliasing groups of latch variables, namely three of size 2 and one of size 5. Now, we can continue with computing the cofactor satisfy count signatures with respect to primary output function $v20.15$, analyzing the new situation with respect to those signatures (i.e., is there a finer partition of the latches?), and so on for all primary outputs — until there is a unique order of the latches.

However, even after using all the primary output variables, we still have just a partial order of the latches:

$$l_1 \quad \{l_6, l_{13}\} \quad l_9 \quad \{l_4, l_{11}\} \quad l_7 \quad l_3 \quad l_{10} \quad l_8 \quad l_{14} \quad \{l_5, l_{12}\} \quad l_2.$$

Now, there are three aliasing groups of size 2. So, let us try and see how the more complex output signatures work. At first, we will use a vector of *function signatures*. This works as follows. We consider the latch variables l_i of the three aliasing groups as output variables v_i again. Then we compute restrictions of such an output variable v_i (output function f^{v_i}), that are independent of the latch input variables. For the purpose of this example, let us take the following two functions:

$$(f^{v_i}_{u_1 u_2 \ldots u_{14}}, f^{v_i}_{\bar{u_1} \bar{u_2} \ldots \bar{u_{14}}}).$$

This is a vector of function signatures for each latch variable of the three remaining aliasing groups of our benchmark circuit. Unfortunately, there is no difference between the function signatures of l_6 and l_{13}, l_4 and l_{11}, and l_5 and l_{12}, respectively. (In our experiments we use six different function signatures.) Applying the *canonical order signature* now, helps to distinguish between latch l_4 and l_{11}, and we get the following partial order for the latches:

$$l_1 \quad \{l_6, l_{13}\} \quad l_9 \quad l_{11} \quad l_4 \quad l_7 \quad l_3 \quad l_{10} \quad l_8 \quad l_{14} \quad \{l_5, l_{12}\} \quad l_2.$$

Let us try the *iterating function signature* next. Here, we use the same functions as for the function signature described above, but with one important difference. The restrictions of an output function f^{v_i} that we compute, is not independent of *all* latch input variables, but only of those that are still in aliasing groups. So, a vector of those restricted functions for latch variable l_i of one of our aliasing groups is:

$$(f^{v_i}_{u_5 u_6 u_{12} u_{13}}, f^{v_i}_{\bar{u_5} \bar{u_6} \bar{u_{12}} \bar{u_{13}}}).$$

However, we need to reorder the unique latch input variables in the restricted functions in order to get an iterating function signature (see Section 5.2). The new order is the permutation independent and unique suborder that we get by the established order of our latch variables:

$$u_1 \quad u_9 \quad u_{11} \quad u_4 \quad u_7 \quad u_3 \quad u_{10} \quad u_8 \quad u_{14} \quad u_2.$$

By reordering the latch input variables in the two functions described above, we get an iterating function signature for latch variable l_i:

$$(g^{v_i}_{u_5 u_6 u_{12} u_{13}}, g^{v_i}_{\bar{u_5} \bar{u_6} \bar{u_{12}} \bar{u_{13}}}),$$

and as our experiments have shown, we finally can establish a unique permutation independent order of all latch variables with the help of these output signatures:

$$l_1 \quad l_6 \quad l_{13} \quad l_9 \quad l_{11} \quad l_4 \quad l_7 \quad l_3 \quad l_{10} \quad l_8 \quad l_{14} \quad l_5 \quad l_{12} \quad l_2.$$

5.3 Experimental Results

We implemented the signatures and ideas presented in the previous sections in the Berkeley SIS–system, release 1.3, in C [34]. To get an understanding about the quality of the signatures we tested a set of 97 benchmarks. These are all *fsmexamples* and all *smexamples* from the LGSynth91 benchmark set for which we could construct the ROBDDs [1]. The experiments were conducted on a SUN Sparcstation 10 with 64 MByte RAM.

The first experiment conducted was to determine the best signature order. There are several input and latch output signatures that we can use to get a unique possible correspondence of the latches. The best order of these signatures is the one with the smallest CPU–time required to obtain unique correspondence. We tried the following orders:

- Use all input signatures first and then all latch output signatures.
- Use all latch output signatures first and then all input signatures.
- Use all those signatures first, for which no exhaustive ROBDD constructions are necessary – do this with input priority and with latch output priority.

We observed that using the latch output signatures first seems to be the better choice. Thus, we decided to use the following order for further investigations.

First compute three simple latch output signatures on each latch output variable f:

- $|f|$,
- $sort(|f_{x_1}|, \ldots, |f_{u_k}|)$,
- breakup signature for f.

Then use some input signatures introduced in Section 3.2.3. These are the satisfy count signatures and the breakup signatures. And finally use the more qualified latch output signatures: function signature, canonical order signature, iterating function signature.

For the *fsmexamples* of the LGSynth91 benchmark set, the signature procedure could establish a unique possible correspondence for all latches of all benchmarks in less than 2 seconds. Here, using the simple latch output signatures was enough, except for benchmark *shiftreg.kiss2*. For this example it was necessary to use the cofactor satisfy count signature function for input variables as well. So, let us concentrate on considering the results of our investigations for the *smexamples*. Table 5.1 presents these results. The first 5 columns include the benchmark characteristics (name, number of inputs, outputs, and latches as well as the number of ROBDD nodes). The next 4 columns show the number of possible correspondences of the latches after using the simple latch output signatures (*so-sigs*), then after using the input signatures in addition (+*i-sigs*), next after using

name	#i	#o	#l	#bdd	# of correspondences with				cpu time
					so-sigs	+i-sigs	+fc-sig (+can-sig)	+itf-sig	(in sec.)
clmA	382	82	33	2211	1				91.2
clmB	382	369	33	1998	1				95.3
daio	2	3	4	21	2	2	1		0.0
ex2	3	3	19	406	$2^4 \cdot 4! \cdot 6!$	$2^7 \cdot 3!$	256	1	10.1
ex3	3	3	10	143	6	1			0.2
ex4	7	10	14	252	$4 \cdot 7!$	8	8(4)	1	2.8
ex5	3	3	9	128	4	1			0.2
ex6	5	8	9	162	1				0.2
ex7	3	3	10	159	4	1			0.2
MinMax4	7	4	12	523	4!	1			0.2
MultiplierB_16	17	1	30	124	$14! \cdot 15!$	$13! \cdot 14!$	1		3.1
MultiplierB_32	32	1	62	249	1				25.4
s1196.bench	14	14	18	2822	1				0.2
s1238	15	15	18	2840	1				0.2
s1423.bench	17	5	74	13657	$4 \cdot 3! \cdot 4!$	$3! \cdot 4!$	6	1	216.0
s1488.bench	8	19	6	489	1				0.0
s1494.bench	8	19	6	484	1				0.0
s208.1.bench	10	1	8	71	1				0.1
s208	12	3	8	82	1				0.1
s27.bench	4	1	3	12	1				0.0
s298.bench	3	6	14	118	2	2	1		0.2
s344.bench	9	11	15	180	$3! \cdot 4!$	4!	1		0.7
s349.bench	9	11	15	178	$3! \cdot 4!$	4!	1		0.7
s382.bench	3	6	21	176	2	2	1		0.3
s386.bench	7	7	6	142	1				0.0
s400.bench	3	6	21	176	2	2	1		0.3
s420.1.bench	18	1	16	203	1				0.7
s444.bench	3	6	21	191	2	2	1		0.3
s510.bench	19	7	6	185	1				0.0
s526.bench	3	6	21	169	2	2	1		0.7
s526n	4	7	21	164	2	2	1		0.7
s641.bench	35	23	19	777	4	1			0.7
s713.bench	35	23	19	777	4	1			0.6
s820.bench	18	19	5	309	1				0.0
s832.bench	18	19	5	309	1				0.0
s838.1.bench	34	1	32	659	1				4.4
s838	36	3	32	323	1				4.6
s953	17	24	29	508	1				0.3
sbc	40	56	28	1689	2	1			0.7

Table 5.1: The Quality of Signatures in L_π

function and canonical order signature (+*fc-sig (+can_sig)*), and finally after using the iterating function signature (+*itf-sig*). The last column includes the CPU time in seconds needed to establish a unique possible correspondence by using these signatures functions. The time 0.0 seconds means that there was no measurable cpu–time. In Section 5.2.4, we demonstrated this process on the

specific benchmark *ex4.slif*.

The results are very promising. We could establish a unique possible correspondence for all latches of each benchmark of our actual set. For about 49% of the benchmarks it was enough to use simple output signatures in order to uniquely identify each of the latches. Applying input signatures helps to solve the problem for a further 18%. And finally, the function signature and the more exhaustive iterating function signature establish a unique possible correspondence for the latches for 33% of all benchmarks. Note, that the canonical order signature could uniquely identify exactly one latch output, that is for *ex4.slif*. Note also that the CPU times are very modest.

5.4 Symmetries in Latch Equivalence

However, applying signatures to solve the latch equivalence problem does not guarantee a complete solution. Similar to the combinational permutation problem, P_π, this approach will fail if any kind of latch symmetry appears in a circuit. Moreover, those symmetries are likely to appear in practice. For example, circuits generated from high level descriptions are likely to have many symmetric (in fact equivalent) state variables. In this case, signatures cannot help (see Chapter 4). Nevertheless, we can extend the signature approach by considering latch symmetry and applying methods used in Chapter 4.

Two latches l_1 and l_2 of $F \in \mathcal{F}_{n,m,k}$ are *symmetric* iff their variables u_1 and u_2 are input symmetric with respect to all primary outputs $y_1, \ldots, y_m$, and with respect to the latch outputs $v_3, \ldots, v_k$. Furthermore, the two functions $v_1(\ldots, u_1, u_2, \ldots)$, and $v_2(\ldots, u_2, u_1, \ldots)$ have to be equal. If this is the case, then the two latches l_1 and l_2 can be exchanged in F without changing the circuit function. It is obvious that this symmetry between two latches can be easily tested by using the known tools in order to test the symmetry of the latch input variables [28]. ROBDDs are used to compare the two latch output functions. It is possible that other kinds of latch symmetry can be defined and handled in a similar way.