Automatic Optimization Techniques for Formal Verification of Asynchronous Circuits

M. Boubekeur and M.P. Schellekens
Centre for Efficiency-Oriented Languages (CEOL)
Department of Computer Science
NUI Cork, Ireland
{m.boubekeur, m.schellekens}@cs.ucc.ie

Abstract— **Even medium size asynchronous circuits may display a complex behavior, due to the combinational explosion in the chronology of events that may happen. It is thus essential to apply automatic optimization techniques to avoid such complexity when formally verifying the correctness of the circuit. This paper presents dedicated techniques for optimization of formal verification of asynchronous circuits, these include for instance: automata reduction, pre-order reduction and automatic abstraction. All these techniques have been implemented and tested in a formal verification environment.**

I. INTRODUCTION

Large Scale Integration (VLSI) technology in the last few years has led to faster, but more complex systems. The synchronous logic design encounters major problems (distribution of clock, energy, modularity) in handling this increasing complexity. Asynchronous circuits show interesting potentials in several fields such as the design of microprocessors, smart cards and circuits with low power consumption.

This work[1] is part of a research work that consists of the analysis and the automatic validation of asynchronous specifications written in CHP[7], prior to their synthesis. The goal is to introduce formal methods into the asynchronous circuit synthesis flow [8], [3].

The verification approach consists in using formalisms and tools coming from the field of software validation. The CADP toolset from INRIA, whose execution model is similar to the asynchronous circuits one, was selected. CHP specifications are analyzed and translated in terms of *Extended Labelled Transitions Systems* (ELTS) with guarded commands. These ELTS descriptions explicitly integrate channels and simple "read" and "write" actions. They appear to be more appropriate for the formal validation of initial CHP specifications. They are validated using the *IF/CADP* environment [5], which provides model checking and bi-simulation tools.

The efficiency of the verification approach is significantly influenced by the size of the underlying ELTS associated with the circuit. It is therefore crucial to keep this size as small as possible. For this purpose, it is thus essential to apply automatic optimization techniques. In this work we propose and implement strategies for the optimization of the generated ELTS models. Figure 1 shows the validation environment. We start from a CHP specification, to achieve some verification tasks. The reduction techniques, including automata reduction, pre-order reduction, explicit generation of interleaving and automatic abstraction, are incorporated in the formal verification environment.

[1]Part of this work was carried out at TIMA laboratory, Grenoble, France.

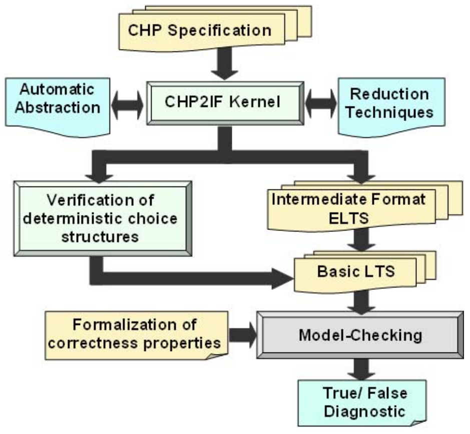

Fig. 1. Formal verification environment

A minimum knowledge of asynchronous circuits and formal verification is needed for the understanding of the presented work. The reader is invited to read [7] and [2]. They describe the asynchronous circuits, the CHP language and the intermediate format IF. [1] contains a description of the formal verification environment for the validation of the asynchronous circuits specifications.

The remainder of the paper is organized as follows: in Section II we give a description of the different optimization techniques. Section III is an example which illustrates the application of the automatic abstraction technique. Finally Section IV gives a conclusion of our work.

II. TECHNIQUES OF OPTIMIZATION

The goal of these techniques is to reduce at the compilation time the number of ELTS states. Generally we are brought to generate, starting from the product of all ELTS, the LTS modelling all the possible executions of the circuit. It is then obvious that any removed state of the ELTS will inevitably

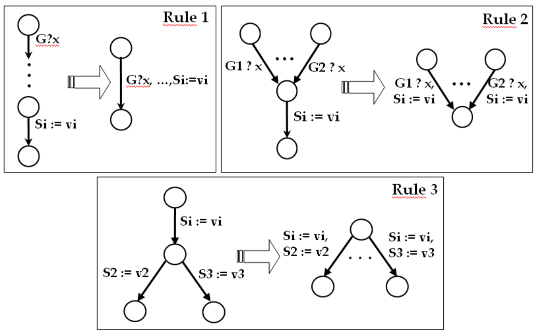

Fig. 2. Reduction rules for compacting ELTS during compilation

reduce the size of the generated LTS model and consequently the time for performing the formal verification phase. In this section we present the different optimization techniques we have implemented in the formal verification environment.

A. Automata reductions

In an ELTS, a transition can be composed of a guard, a synchronization expression and possibly of several assignments. While taking advantage of the expressivity of the ELTS, we have implemented a set of transformation rules allowing the reduction in size of the ELTS generated during the compilation of the CHP specification. This is done while preserving the semantics of the automata. The rules of reductions are presented in Figure 2 and outlined as follows:

- *Rule 1:* The idea is to compact a set of sequential transitions in only one transition. That is possible in the case of an action of communication followed by one or more sequential assignments.
- *Rule 2:* In the case of a conjunction of several transitions followed by an assignment, the latter is added to each preceding transition.
- *Rule 3:* If an assignment is followed by a disjunction of several assignments, then the first assignment is added at the beginning of all the transitions which succeed it.

We note that each transformation rule removes at least a state, and strictly preserves the same action sequences.

B. Automatic abstraction

The abstraction is a well-known technique to deal with the problem of states explosion. It consists of building an abstract model starting from the original model, called a concrete model, such that each action of the concrete model can be simulated by an action in the abstract model.

In the verification process, we are often interested in checking the control aspects since the verification of the functional properties, calculation for example, is difficult and very expensive. An intuitive manner to check these control aspects more easily consists of abstracting the data parameters. Accordingly, certain ports or variables are not used in the control aspects of a process (the way the processes behave), they are then not-relevant for the verification of the control properties, but their simple presence generates an enormous over-cost of calculation and memory at the time of the formal verification phase.

The idea is to modify the parameters (variables or ports) which are not essential so that they always have a constant value. At best, one can completely remove these parameters. To do this, we developed an abstraction function called Nec diminutive of Necessary. The abstraction function Nec expresses that a variable or a port is needed for the verification of a control property.

Definition 1 (Nec: the abstraction function): $Nec(X) = true$ if X is used in a guard; or if X intervenes by dependency in the calculation of the value of a variable y, such as $Nec(y) = true$;

This function is applied to all parameters (variables and ports) of the CHP program by using the dependency rules described in TABLE I.

TABLE I
ABSTRACTION FUCTION RULES

CHP operation	**Rules**
Guard	
$[f(v_1, ..., v_n)]$	$Nec(v_1) \wedge ... \wedge Nec(v_n)$
Communication actions	
$P!(f(v1, ..., vn))$ $P?v$	$Nec(P) \Rightarrow Nec(v1) \wedge ... \wedge Nec(v_n)$ $Nec(v) \Rightarrow Nec(P)$
Assignements	
$v := f(v1, ..., vn)$	$Nec(v) \Rightarrow Nec(v1) \wedge ... \wedge Nec(v_n)$

The resolution of this system of equations is done at the pre-processing compilation time. It thus gives during the generation of the ELTS code all information concerning the variables which are not necessary. After application of the abstraction function, we can build the set Abs.

Definition 2 (Abs: the set of non-necessary parameters): Let X be a variable or a port of the CHP program: $X \in Abs \; if \; Nec(X) = false$

Finally the abstraction technique consists of building the set Abs, then to abstract all ports and variables of this set.

1) Parameterized automatic abstraction:

If one wishes to verify a property which depends on a data variable, this variable should not be abstracted, and of course neither all the variables intervening during its calculation. We envisaged, to answer this need, the implementation of a parameterized automatic abstraction. It is then possible to select a variable or a port as a necessary parameter for the verification of a property.

By allowing this possibility, we improved our abstraction technique, since it becomes not only used for the control aspects. Concretely we eliminate the parameter (variable or port) from the set Abs, this parameter then becomes necessary and consequently cannot be abstracted.

C. Explicit generation of parallelism

As we saw previously, the CHP language allows parallelism within a process whereas the ELTS format (IF) does not allow this functionality. Indeed, this problem could be solved in all cases by creating new concurrent processes which would contain only sequential statements. But this method would generate new communication actions to allow the synchronization of these new processes.

A manner of avoiding the creation of new processes is to explicitly generate the parallelism, between the concurrent actions, according to the interleaving semantic. The method then consists in distinguishing, among the set of concurrent actions, those which can be represented by only one ELTS transition (simple or atomic statement). All the possible interleaving between these actions are explicitly generated (see Figure 3.)

1) Interleaving example: Lets consider these three concurrent atomic statements: S1, S2, S3.

Figure 3 shows the explicit generation of the interleaving between these concurrent actions. We note that the number of states necessary to generate the interleaving is: $1+3+3+1 = 8$.

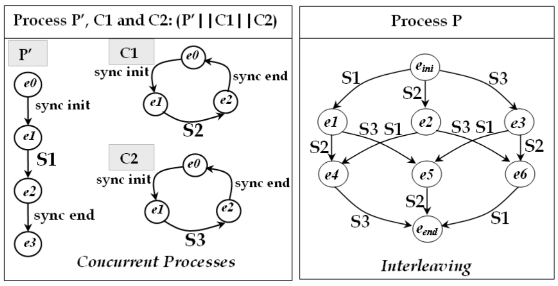

Fig. 3. Explicit generation of interleaving of concurrent actions

2) General case: Let k be the number of transitions. The number of states necessary for the generation of all possible interleaving is 2^k.

The construction of the ELTS modelling the interleaving of k transitions in an optimal way is based on the use of the Triangle of Pascal [4]. To build all possible interleaving of k concurrent actions, it is enough to build the representation of the Triangle of Pascal and add the transitions between the states. Figure 4 illustrates interleaving between 4 instructions: the number of necessary states is 24; the 5 levels are laid out in the following way: "1.4.6.4.1", according to the coefficients of the Triangle of Pascal:

$$C_i^n = \frac{n!}{(n-i)!i!}$$

3) Need for creation of new processes to model parallelism: In certain cases, the treatment of concurrency by explicit generation of interleaving is rather complicated. It is typically the case for a concurrency between serial behaviors. In this case we are obliged to model concurrency by the creation of new concurrent processes synchronized by "extra-ports".

The following CHP code puts several behaviors in parallel which cannot be handled by the explicit generation method. (S1 , (S2 ; S3) , [A # $\Rightarrow$ S4])

The sequential portions have to be extracted in new processes. These new processes are synchronized on the original

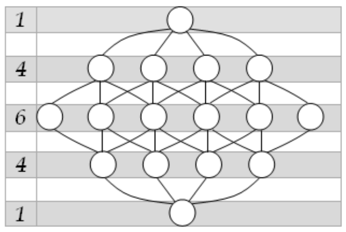

Fig. 4. Method for generation the interleaving

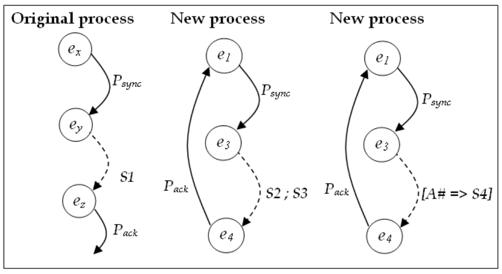

Fig. 5. Example of concurrency that needs creation of new processes

process by the extra-ports: P_{sync} and P_{ack}. The process which contained parallelism recovers one of the expressions (S1 in Figure 5), and synchronizes the new processes so that those are carried out at the appropriate time. It is necessary to make the shared variables global and to specify the synchronized ports correctly.

D. Pre-order Reduction

In order to further reduce the ELTS generated during compilation, we implemented a pre-order reduction technique. In this technique, we consider all the concurrent assignments. We extract all the assignments which do not contain dependency between variables. These concurrent assignments are then replaced by only one sequential execution.

The following CHP code presents a set of sequences to realize in parallel. We want to extract the sequences which will not change semantics independently of the execution.

(1)	(2)	(3)	(4)	(5)
x:=10,	y:=x+1,	z:=w,	A!x,	(C?t; B!t)

The first two statements cannot be withdrawn, because they are in interdependence on variable x: according to the sequence of execution (1); (2) or (2); (1), the result at the end of parallelism is not the same one.

The statement (3) can be extracted, because neither variable z, nor the variable w are dependent on another assignment. The statement (4) should not be extracted because it is a communication action, and we think that it would be necessary to carry out very complex checks to verify dependency between communication actions. Indeed, contrary to the variables which are local, all the ports being global makes it more difficult to analyse their dependency. The expression (5) cannot be extracted because it is not an atomic action. In conclusion, on this example, we can carry out sequentially the statement (3), followed by the interleaving of the expressions (1), (2), (4) and (5).

III. Application on an example

To highlight the previous optimization techniques, we present here the application of the more sophisticated optimization technique, the automatic abstraction, on a typical arbiter (Figure 6). The circuit behaves in the following way: input channel C is read in a local variable $ctrl$ which is tested using a deterministic choice structure. The value read from input channel E is propagated to output channel $S1$ if $ctrl$ is 0, to $S2$ if $ctrl$ is 1, and to both $S1$ and $S2$ in parallel if $ctrl$ is 3.

```
COMPONENT Det-Selector
PORT (C: IN DI MR[3][1]; E: IN DI MR[2];
 S1, S2 : OUT DI MR[2] )

BEGIN
PROCESS main
PORT (C: IN DI MR[3][1]; E: IN DI MR[2];
 S1, S2 : OUT DI MR[2] )
variable x : MR[2];
variable ctrl : MR[3][1];
BEGIN
 [C?ctrl;
  @[Ctrl = "0"[3] => E?x; S1!x; break
     Ctrl = "1"[3] => E?x; S2!x; break
     Ctrl = "2"[3] => E?x; S1!x, S2!x; break]; loop];
END;
END;
```

Fig. 6. CHP code for the Arbiter

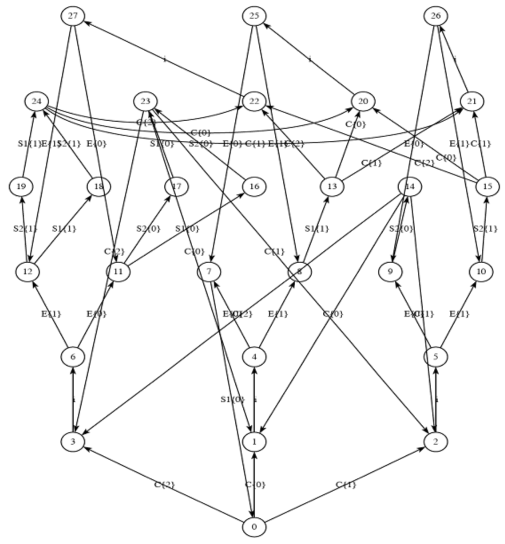

Fig. 7. LTS of the execution model for the asynchronous arbiter

The generated LTS for the asynchronous arbiter without abstraction of the data is described in Figure 7. After execution of the automatic abstraction, we obtain the LTS of Figure 8. By application of the abstraction function (Nec), only one port is decided to be necessary, the control port C. The remaining ports are considered to be abstracted, $Abs = \{E, S1, S2\}$. The data ports of the set Abs are then abstracted by an enumerated type of one value $\{D\}$.

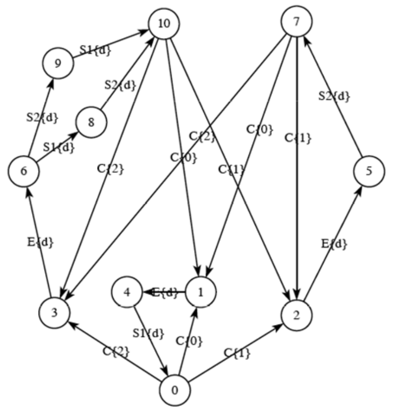

Fig. 8. LTS of the asynchronous arbiter generated by abstraction

We observe that we gain a considerable size factor. This abstraction preserves the properties of the arbiter. Of course the formal verification phase is more efficient on the reduced model.

IV. Conclusion

In this paper we presented an implementation of efficient techniques for optimization of the formal verification of asynchronous circuits. These techniques allowed the formal verification of real-world asynchronous circuits, e.g. the results of the verification of *Data Encryption Standard* (DES) chip [6] are reported in [1]. These techniques are entirely integrated in the formal verification environment.

Acknowledgment

Dr. M. Boubekeur would like to thank Prof. D. Borrione, Prof. M. Renaudin and Dr. L. Mounier for their contributions to this work and for many stimulating and helpful discussions.

References

[1] D. Borrione, M. Boubekeur, et al. "VLSI-SOC: From Systems to Chips". Chapter:"Validation of Asynchronous Specifications using IF/CADP", Kluwer-Springer, 2006, ISBN: 0-387-33402-5.
[2] M. Bozga, J.Cl. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, L. Mounier. IF : A Validation Environment for Timed Asynchronous Systems. Proceedings of CAV00 (Chicago, USA) July 2000.
[3] A. V. DinhDuc, "Synthse Automatique de Circuits Asynchrone QDI". PhD thesis, INP of Grenoble, 2003.
[4] R. L. Graham, D. E. Knuth, and O. Patashnik, Binomial Coefficients. Ch. 5 in Concrete Mathematics: A Foundation for Computer Science, 2nd ed. Reading, MA: Addison-Wesley, pp. 153-242, 1994.
[5] http://www.inrialpes.fr/vasy/cadp/
[6] NIST, Data Encryption Standard (DES), FIPS PUB 46-3, National Institute of Standards and Technology, Reaffirmed October 25, 1999. http ://csrc.nist.gov/csrc/fedstandards.html
[7] M. Renaudin, "Asynchronous Circuits and Systems : a Promising Design Alternative". In "MIGAS 2000", special issue Microelectronics-Engineering Journal, Elsevier Science, Vol. 54, N 1-2, December 2000, pp. 133-149.
[8] M. Renaudin, J.B. Rigaud, et al. "TAST CAD Tools". ASYNC02 TUTORIAL, ISRN: TIMARR-02/04/01FR, 2002.