Validation of asynchronous circuit specifications using IF/CADP

Dominique Borrione[1], Menouer Boubekeur[1], Laurent Mounier[2],
Marc Renaudin[1], Antoine Sirianni[1]

[1] TIMA, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France
[2] VERIMAG, Centre Equation, 2 avenue de Vignate, 38610 Gières, France
{Surname.Name}@imag.fr

Abstract

This work addresses the analysis and validation of modular CHP specifications for asynchronous circuits, using formalisms and tools coming from the field of distributed software. CHP specifications are translated into an intermediate format (IF) based on communicating extended finite state machines. They are then validated using the IF environment, which provides model checking and bi-simulation tools.

1. Introduction

Asynchronous circuits show interesting potentials in several fields such as the design of numeric operators, smart cards and low power circuits [1]. An asynchronous circuit can be seen as a set of communicating processes, which read data on input ports, perform some computation, and finally write on output ports. In our work, asynchronous circuit specifications are written in CHP, an enriched version of the CSP-based language initially developed by Alain Martin [2].
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 rigorous design and validation methods. This paper describes the automatic validation of asynchronous specifications written in CHP, prior to their synthesis with the TAST design flow [12, 14].

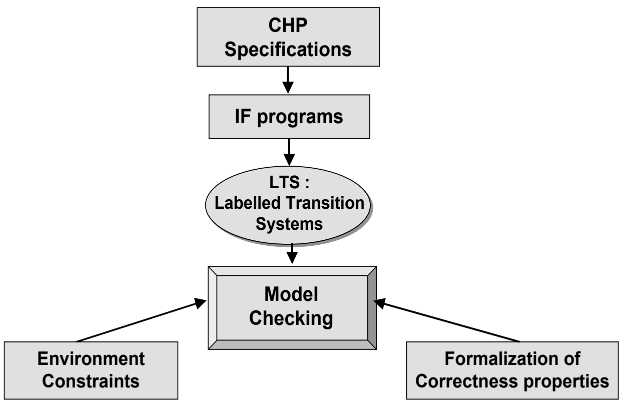
CHP Specifications
IF programs
LTS : Labelled Transition Systems
Model Checking
Environment Constraints
Formalization of Correctness properties

Figure 1: Formal verification flow for CHP

To this aim, we use formalisms and tools coming from the field of software validation, in particular distributed systems, whose execution model is similar to the asynchronous circuits one. We start from an asynchronous specification written in CHP and compile it into the IF format. Resulting IF programs are compiled towards a LTS and eventually submitted to the CADP toolset for verification (Figure 1).
This paper is organized as follows. Section 2 reviews the TAST design flow from CHP. Section 3 describes the validation of concurrent systems with the IF environment. Section 4 discusses the translation of CHP specifications into the IF format, with an emphasis on the CHP concepts that have no direct correspondence in IF. As a case study, section 5 presents the application of our method to an asynchronous FIR filter. Finally, we review related works and present our conclusions.

2. The TAST Design Flow

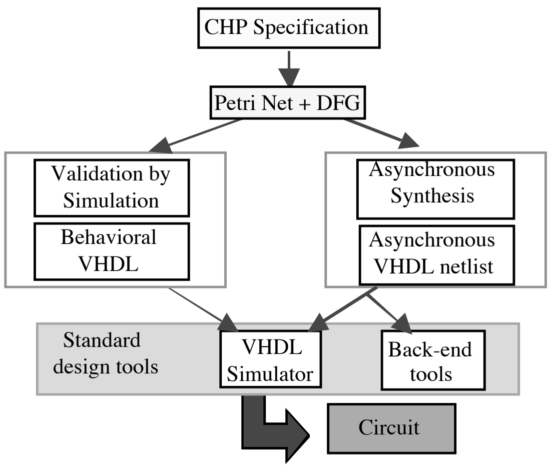

Figure 2: the TAST design flow

In the TAST asynchronous design flow [14], the compiler translates CHP programs into Petri Nets (PN) and Data Flow Graphs (DFG) (Figure 2). The PN model of a CHP specification is translated to behavioral VHDL for simulation purposes. The synthesizer performs process decomposition and refinements on the PN formalization, depending on the selected architectural target: micro-pipeline, quasi delay insensitive circuit (QDI), or synchronous circuit. A dedicated compiler produces a structural gate network, in source VHDL, for simulation and back-end processing using commercial CAD tools.

CHP: the TAST Specification Language

CHP Specifications are organized as lists of components. Each component has a name, a communication interface, a declaration part and a statement part. Hierarchy is managed through local component declarations and instance specifications like in VHDL.

Data types are of three kinds: unsigned multi-rail (MR), signed multi-rail (SMR) and single-rail (SR) with base, length and dimension attributes standing for the range of digits, the number of digits in a vector and the number of vectors in an array, respectively.

This allows supporting arbitrary precision bi-dimensional arrays of numbers, thus modeling memories, registers as well as pure protocol signaling (using SR type) independently of the precision of the machine.

Constants are declared at the component level. They are visible throughout the component and cannot be masked by process variables.

Processes communicate via point-to-point, asymmetric, memory-less message passing, and compatible protocol links. Channels have a name, an encoding and a type. They connect ports of opposite directions (in/out) and opposite protocols (active/passive). Data can be encoded in two ways: DI (one of n code) used for delay insensitive style synthesis and BD for bundled data (binary code) for micro-pipeline style synthesis. Although arrays are supported, channels can only carry out vector values.

The component statement part consists of instances and processes. Instances are similar to VHDL.

Processes have a name, a communication interface, a declaration part and a statement part. Variables are local to their processes and are dynamically initialized when a process starts its execution. Process ports can be connected to local channels or to the component ports. Basic process statements are communication actions and variable assignments. Parallel and sequential compositions, guarded commands, loop and choice operations are the main construction mechanisms to build processes.

Control structures can be either deterministic (the environment must provide mutually exclusive guards) or non deterministic (several guards may be true, only one is elected randomly).

Note that concurrent statements can assign the same variable, or access the same channel, within a process. It may be desirable to check the absence of such behaviors in a given design context.

3. The IF validation environment

IF[10] is a software environment developed in Verimag for the formal specification and validation of asynchronous systems. It provides both a description language (the so-called IF intermediate format), and a set of integrated validation tools. This environment is motivated by two main objectives:

- Gather complementary formal validation techniques (interactive simulation, model-checking, test case generation, etc.) into a single open environment, independent of any high-level description language;
- Support several representation levels of a system behavior: a syntactic level, expressed by the IF intermediate format, and a semantic level in terms of labeled transition systems.

This environment has already been used for software validation in various application domains: safety critical systems, telecommunication protocols, etc. [18].

3.1 The IF intermediate format

A IF system description consists of a set of communicating processes. Each process can access (private) local data and (shared) global data. Inter-process communication can be performed by message passing (through asynchronous buffers), by *rendez-vous* (through gates), or via shared data.

Several predefined data types are proposed (Boolean, integer, enumerated types, arrays, records, etc.) together with abstract data types definition facilities (their concrete implementation being provided in C).

From a syntactic point of view, each process is described as an extended automaton with explicit control states and transitions. Each transition is labelled by an atomic execution step and may include a Boolean guard, an inter-process communication, and assignments to local or global data. The execution model is asynchronous: internal process steps are performed independently from each other, and their parallel composition is expressed by interleaving. However a notion of *unstable* control states allows tuning the atomicity level of process transitions: sequences of transitions between unstable states are considered atomic.

Rather than a "user oriented" description language, IF is a general intermediate format for asynchronous systems. Its main advantages are its expressiveness, flexibility, formal operational semantics, and its well-accepted underlying automaton model. Several high-level specification languages are already automatically translated into IF, such as SDL [10] and UML profiles [19].

3.2 The IF validation toolbox

The IF toolbox is partitioned into two layers:

- The *syntactic layer* provides source level operating tools based on static analysis. They allow performing static optimizations (dead code elimination, live variable analysis) and program slicing.
- The *semantic layer* relies on the so-called model-based validation technique. A generic simulation engine builds the labeled transition system (LTS) expressing the exhaustive behaviour of an IF description. Various validation tools can then operate on this LTS to check whether it satisfies a given property. Particular available tools are the ones offered by the CADP toolbox [7], namely a general temporal logic verifier, and a bi-simulation checker.

In model-based validation of asynchronous systems one of the major concern is to avoid (or at least limit) the state explosion problem (the size of the underlying LTS). In the IF toolbox this problem is tackled at several levels:

- static optimization and program slicing are very efficient since performed at the source level;

- the IF simulation engine is able to produce "reduced" LTS with respect to a preorder relation between execution sequences (still preserving some trace properties);
- large descriptions can be verified in a compositional way (by generating and reducing in turn the LTS associated to each part of the specification).

4. Translation of CHP descriptions into IF

In this section, the semantics of the CHP language and its translation into the IF format are briefly presented.

4.1. Concepts in direct correspondence

The CHP notions of *component, port*, *variable* and *process* are in direct correspondence with the notions of *system*, *gate*, *var* and *process* in IF.
All ports and declared local variables must be typed in both formalisms, but many CHP types give implementation information (e.g. one-hot coding) and their translation must be decided individually.
The three simple statements found in a process body are identical (syntax and semantics) in CHP and IF:

var_Id := expression	variable assignment
In_port ? var_Id	read an input port into a variable
Out_port ! var_Id	write a variable to an output port

4.2. Concepts of CHP that need a transformation step

a) From channels to synchronization expressions
In IF the communication architecture is given by a LOTOS-like synchronization expression. To construct this expression, the headings of all processes must be analyzed to identify the synchronization ports between each pair of processes. To lose no information in the translation phase, comments are added (e.g. Input/Output to distinguish channel direction in the IF system).

b) Process ports
All the component and process *ports* must be declared as *gates* of the IF *system,* even the process ports which are connected to local channels. Thus, ports declared as CHP process ports must be added to the system list of gates if they are not already present (as a component port, or as a port of a previous process).

c) Process body
The block of statements describing the behavior of the process is translated into a set of states and labeled transitions.

4.3. Statements and their composition

In the following, let S1, …Si denote general statements, A1…Aj denote simple assignments, and RW1,…RWi simple read or write communication actions.

a) Sequential composition
In CHP, the sequential composition of S1 and S2, written S1 ; S2 means that S2 starts its execution after the end of S1. To efficiently translate this composition (and reducing the numbers of IF states) we consider three cases, depending on S1 and S2 structure:

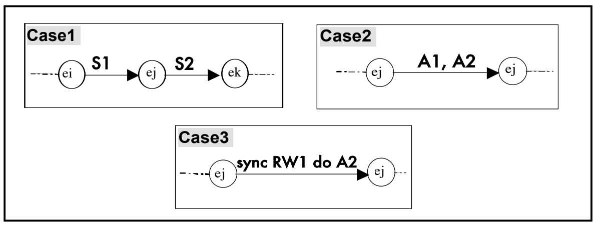

Figure 3 Sequential composition

Case 1: S1 and S2 are possibly complex, or involve more than one communication (general case): intermediate states must be introduced. In IF syntax:

```
from ei do S1 to ej ;
from ej do S2 to ek;
```

Case 2: A1 and A2 are simple assignments: they can all be sequentially executed during a single state transition. In IF syntax: `from ei do A1, A2 to ej;`

Case 3: The first statement RW1 is a communication, considered to synchronize one or more following simple assignments. In IF syntax: `from ei sync RW1 do A2 to ej;`

b) Parallel composition
In CHP, statements separated by commas are concurrent. In IF, no concurrent statements may exist inside a process, parallelism exists only between processes.
To translate the CHP concurrent composition [S1 , S2] the general solution involves the creation of a sub-process *comp* for statement *S2*, and the explicit synchronization of its execution start and completion, as shown on Figure 4. This solution however involves a significant overhead in model size, as it generates all inter-leavings of *comp* with all the model processes.

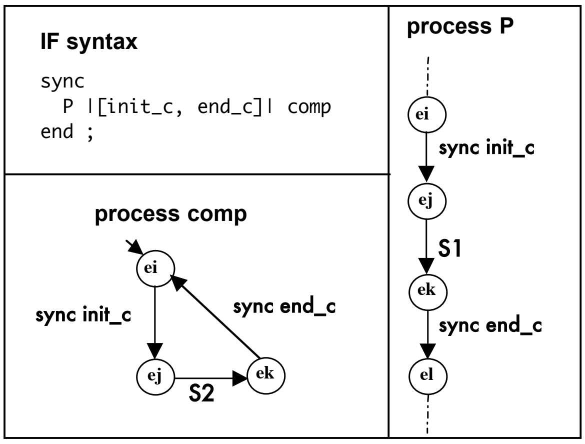

Figure 4 Parallel composition

A more efficient solution is to statically generate a non-deterministic choice between all the possible execution sequences for the concurrent statements, provided these statements initially label a single transition. This keeps the inter-leavings local, and prevents the proliferation of states for the overall model (Figure 5).

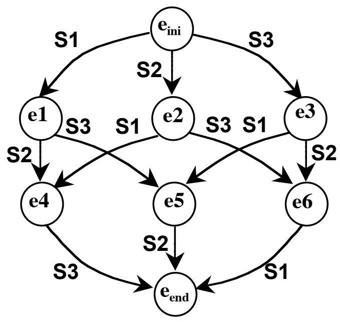

Figure 5: Interleaving generation for [S1, S2, S3]

c) Repetition

The loop statement of CHP allows repeating the execution of a simple or composed statement S. A repeated simple statement labels a transition from a state to the same state (case 1). If S is complex, one or more intermediate states may be generated for its translation, but the first and final states are the same (case 2).

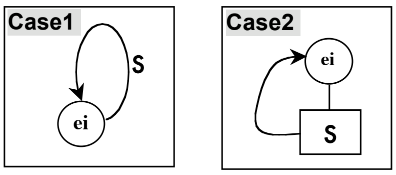

Figure 6: Repetition

d) Deterministic / non deterministic selection

Let Ci be Boolean expressions and Si be simple or composed statements, guarded by Ci. If Ci is false, Si is stalled; if Ci is true, Si is executable.

CHP offers two selection operators. The non-deterministic selection @@ encloses one or more guarded statements, and makes no assumption on the number of guards than can be true. If more than one Ci is true, one among the corresponding Si is randomly selected and executed. The deterministic selection @ imposes on its environment that only one of the Ci be true, a property to be verified. Both are translated as a set of guarded statements that label a set of transitions between the same two states (see Figure 7).

CHP	IF

@[C1 => S1 ; break
C2 => S2 ; break
… … …
Cn => Sn ; break]
	from ei if (C1) do S1 to ej;
	from ei if (C2) do S2 to ej;
	… … …
	from ei if (Cn) do Sn to ej;

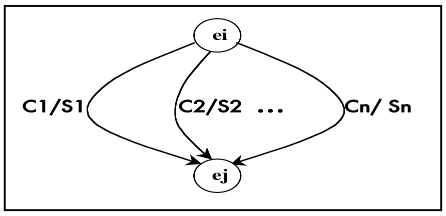

Figure 7: Selection

Repetition and selection may be combined: if one ore more guarded statements end with *loop* instead of *break*, the whole selection block is re-entered upon execution of those statements (Figure 8).

CHP	IF

@[C1 => S1 ; loop
C2 => S2 ; loop
… … …
Cn => Sn ; break]
	from ei if (C1) do S1 to ei;
	from ei if (C2) do S2 to ei;
	… … …
	from ei if (Cn) do Sn to ek;

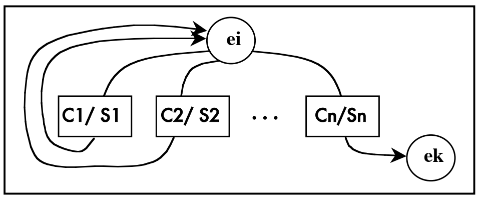

Figure 8□Repetition with selection.

4.4. Example

To illustrate the translation, we consider one typical process: *Mux_3L*, taken from the case study of section 5.

In this process, the control channel (*Ctrl_Round1_L*) is typed MR[3][1], i.e. one-of-three data encoding. Control channel is read in the local variable "Ctrl". According to the value of "Ctrl", one of the two channels (*L0*, *Li_buf2*) is read and its value is written on channel *L16* or *Li_1_buf1*.

CHP code of Multiplexer "Mux_3L"

```
process Mux_3L
PORT ( L0, Li_buf2        : IN DI passive DR[32];
     L16,  Li_1_buf1, Ctrl_round1_l : IN DI passive MR[3]; )
Variable ctrl  :  MR[3];
Variable in1   :  DR[32];
begin
[Ctrl_round1_l ? ctrl;
@[ Ctrl = "0"[3]  =>  L0 ? in1 ; Li_1_buf1 ! in1 ; break
   Ctrl = "1"[3]  =>  Li_buf2 ? in1 ; Li_1_buf1 ! in1 ; break
   Ctrl = "2"[3]  =>  Li_buf2 ? in1 ; L16 ! in1 ; break
  ]; loop ];
end ;
```

The graphical representation of the corresponding IF program is shown on Figure 9.

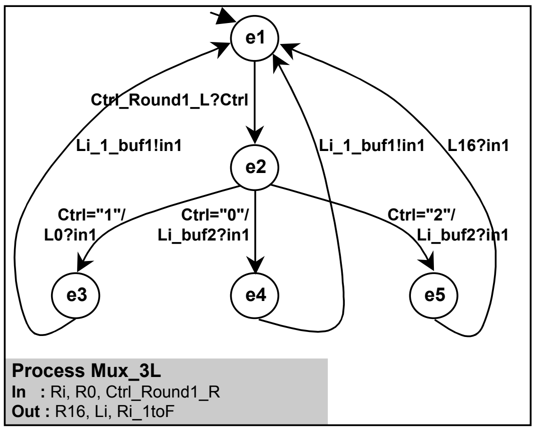

Figure 9□IF representation of Mux_3L

5. Case study : DES Chip

The global architecture of a fully asynchronous DES (Data Encryption Standard) chip is described in Figure10. It is basically an iterative structure, based on three self-timed loops synchronized through communicating channels. Channel Sub-Key synchronises the ciphering data-path with the sub-key computation data-path. CTRL is a set of channels generated by the Controller bloc (a finite state machine) which controls the data-paths along sixteen iterations as specified by the DES algorithm [20].
The 1-bit input channel CRYPT/DECRYPT is used by the Controller to configure the chip and trigger the ciphering. The 64-bit channels DATA and KEY are used to respectively enter the plain text and the key. The ciphered text is output through the 64-bit channel OUTPUT.

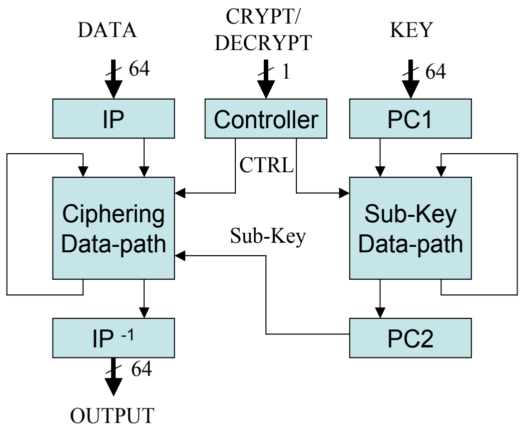

Figure 10: Asynchronous DES Chip architecture

The overall CHP component contains 26 processes (17 in the ciphering data-path, 4 in the sub-key data-path). The translation produces an IF system where each CHP process is represented by an IF process.

5.1. Some verified properties

Characteristic properties of the DES behavior have been expressed in mu-calculus, and automatically verified using CADP, on a SUN Ultra 250 with 1.6 GB memory. The model LTS generation time is 1:05:27.02 and its size is: (3 e+7 transitions, 5.3 e+6 states). The meaning and performances (verification time in h:min:sec; Memory) of typical properties are listed below :
P1: Freedom from deadlock (27:41.23 ; 884 MB)
P2: After reception of the 3 inputs (Key, Data, Decrypt), Output is always produced (27:31.93 ; 865 MB)
P3: The counter of the controller counts correctly. (26:25.64 ; 885 MB)
P4: Each iteration of the ciphering and sub-key data-paths synchronizes correctly. (26:25.65 ; 879 MB)

5.2. Verification by behavior reduction

To verify properties P3 and P4, which relate to the synchronizing channels only, an alternative technique is available. The model behavior is reduced by hiding all the labels which do not relate to CTRL and Sub-Key. This can be obtained by applying property-preserving equivalence reduction techniques which for this model take (11:46.31; 1.71 GB). The resulting LTS is depicted on Figure 11, which shows the cyclic behavior and exhibits the synchronization on channel *Sub-Key*.

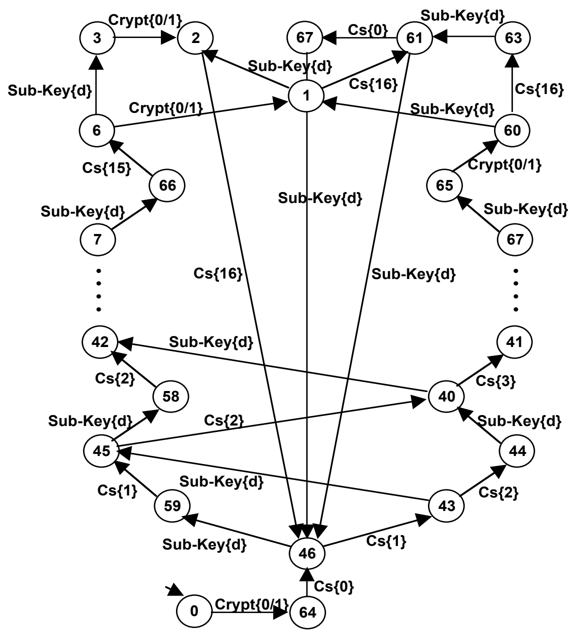

Figure 11: Reduced LTS for property P3

5.3. Handling state explosion

The following techniques have been used during the verification experiments:
- Data abstraction: this is a well known strategy, by which data are reduced to one bit, when their value do not influence the property at hand.
- Explicit generation of interleaving for CHP intra-process concurrent statements (instead of generating synchronized concurrent IF processes, see section 4.3). Without this feature, the model generation faces LTS size explosion.

6. Related works

The verification of asynchronous circuits heavily depends on the design approach: timed or un-timed (delay insensitive) circuits. In the first approach, ATACS is a set of tools that supports the synthesis, analysis, and verification of timed circuits [3]; KRONOS is dedicated to timed automata verification [11]. However, only very small examples have been published using these techniques: the verification of timed systems faces serious complexity problems. In contrast, our work stays at a higher abstraction level, focuses on un-timed specifications and leaves open the choice of the target implementation model.
Concurrent processes are widely used for specifying an un-timed asynchronous behavior; two main directions have been explored: language-based and graph-based.
Graph-based specification methods are used at a low conceptual level. They are painful for the designer, but the synthesized circuit is fast and efficient. Petri nets or Signal

Transition Graph (STG) formalisms are used. Both the circuit specification and its environment assumptions can be modeled using Petri nets or STG. A state encoding is associated to this representation, which allows the application of BDD symbolic model checking techniques [6, 13, 16].
The language-based method eases the designer's task; it allows for modular and high level specifications, at the cost of efficiency in the synthesized result.
Early validation works include experiments with CIRCAL to model micro-pipe lines: the proof is performed on the parallel composition of the implementation and the properties modeled as processes [4]. The use of LOTOS to specify asynchronous circuits has also been suggested, making available the CADP verification tool box [17]. We question the adequacy of LOTOS for circuit specification as no circuit synthesis flow has been built from it. We do keep however the idea of using CADP, but taking as input a hardware design language.
In a previous feasibility study [5], we used an industrial symbolic model checker intended for property checking on RTL designs. We translated the Petri Net produced by TAST as a pseudo synchronous VHDL description, where the pseudo clock thus introduced was only to make each computation cycle a visible state. In essence, our translation performed a static pre-order reduction. This approach gives good results after the communication expansion, but this occurs too late in the design process to validate the initial specifications.

7. Conclusion

We have implemented a first prototype translator that automatically produces the IF model for a CHP specification, along the principles explained in this paper. Preliminary experiments have shown that the IF/CADP toolbox offers a convenient abstraction level for the formal validation of initial CHP specifications.
Essential properties can be proven on the specification, before synthesis decisions are made visible in the design description. This provides a new service to the TAST user. During the architecture exploration process, the designer may use transformations that have not been formally proven correct, and wishes to check that essential properties are retained on the refined architecture; our automatic link to IF/CADP is a possible answer to this requirement.
The perspectives of this work include some improvements to the current version of the translator (e.g. negative numbers are currently not recognized) and its application to many more benchmarks. The scalability of the approach to large circuits, of the size of a 32-bit microprocessor will be measured. It will certainly involve elaborate model reduction strategies, some of which are still not automated. We also intend to work on the combination of symbolic simulation and model checking, applied to asynchronous circuit verification. Finally, replacing the mu-calculus by a more widely accepted property specification language such as the Accelera PSL would ease the designer's access to the verification toolbox.

8. References

[1] M. Renaudin, "Asynchronous Circuits and Systems: a promising design alternative", Microelectronics-Engineering Journal, Elsevier Science, Vol. 54, N° 1-2, Dec 2000, pp. 133-149.

[2] A.J. Martin, "Programming in VLSI: from communicating processes to delay-insensitive circuits", in C.A.R. Hoare, editor, Developments in Concurrency and Communication, UT Year of Programming Series, 1990, Addison-Wesley, p. 1-64.

[3] H. Zheng, E. Mercer, and C. Myers, "Automatic abstraction for verification of timed circuits and systems", Proc. CAV'01, LNCS 2102, Springer, pp. 182-193, July, 2001.

[4] A. Cerone, G. Milne: "A Methodology for the Formal Analysis of Asynchronous Micropipelines", Proc. FMCAD 2000, LNCS N° 1954, Springer Verlag, pp.246-262

[5] D. Borrione et al. "An Approach to the Introduction of Formal Validation in an Asynchronous Circuit Design Flow". Proc. 36th Hawai Int. Conf. on System Sciences (HICSS'03). Jan. 2003

[6] G. Delzanno and A. Podelski,"Model Checking in CLP". Proc. 5th Int. Conf. TACAS'99. R. Cleaveland, ed., Springer Verlag LNCS N°1579, pp.223-239,1999.

[7] http://www.inrialpes.fr/vasy/cadp/

[8] J. Cortadella et al. "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers." IEICE Trans. on Information and Systems, E80-D(3): 315-325, Mar. 97.

[9] K. Van Berkel, "Handshake Circuits – An Asynchronous Architecture for VLSI Programming", Cambridge University Press, 1993, ISBN : 0-521-45254-6

[10] M. Bozga, J.-C. Fernandez, et al. "IF : An Intermediate Representation and Validation Environment for Timed Asynchronous Systems". Proc. FM'99, Toulouse, LNCS, 1999.

[11]M. Bozga, H. Jianmin , O. Maler, S.Yovine,"Verification of Asynchronous Circuits using Timed Automata",Proc. TPTS'02 Workshop, Elsevier Science Pub. April 2002.

[12] A Dinh Duc, L. Fesquet, M. Renaudin, "Synthesis of QDI Asynchronous Circuits from DTL-style Petri-Net", IEEE/ACM Int. Workshop on Logic & Synthesis, New Orleans, June 4-7, 02.

[13] A. Kondratyev, J. Cortadella, M. Kishinevsky, et al.: "Checking signal transition graph implementability by symbolic bdd traversal". Proc. EDTC'95, pp 325-332, Paris, March 95.

[14] M. Renaudin, J.B. Rigaud, A. Dinhduc, A. Rezzag, A. Sirianni, J. Fragoso : "TAST CAD Tools", ASYNC'02 TUTORIAL, ISRN: TIMA--RR-02/04/01—FR, 2002

[15] R. Manohar, T.K. Lee, A.J. Martin. "Projection: a synthesis technique for concurrent systems". 5th Int. Symp. Advanced. Research in Asynchronous Circuits and Systems, Apr. 99.

[16] V. Khomenko and M. Koutny: "Towards An Efficient Algorithm for Unfolding Petri Nets". Proc. CONCUR'01. Springer-Verlag, LNCS N° 2154 (2001) 366-380.

[17] M. Yoeli and A. Ginzburg, "LOTOS-based Verification of Asynchronous Circuits", Tech. Report, Dept. of Computer Science, Technion, Haifa, 2001.

[18] M. Bozga et al. "Automated validation of distributed software using the IF environment", Proc. Workshop on Software Model-checking, El. Notes in TCS vol. 55 Elsevier Science Pub. July 00.

[19] http://www.agedis.de and http://www-omega.imag.fr

[20] NIST, Data Encryption Standard (DES), FIPS PUB 46-3, National Institute of Standards and Technology, Reaffirmed 1999 October 25. http://csrc.nist.gov/csrc/fedstandards.html