Modeling CHP descriptions in Labeled Transitions Systems for an efficient formal validation of asynchronous circuit specifications

Menouer Boubekeur
Dominique Borrione

TIMA-VDS,
46 avenue Félix Viallet,
38031 Grenoble Cedex, France

Laurent Mounier

VERIMAG
Centre Equation,
2 avenue de Vignate
38610 Gières, France

Antoine Sirianni
Marc Renaudin

TIMA-CIS,
46 avenue Félix Viallet,
38031 Grenoble Cedex, France

Abstract

This work addresses the analysis and validation of CHP specifications for asynchronous circuits, using property verification tools. CHP semantics, initially given in terms of Petri Nets, are reformulated as labeled transition systems. Circuit 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. The direct translation must be optimized to delay state explosion. Performance studies are reported.

1. Introduction

With the integration of several complex systems on only one chip, synchronous logic design encounters major problems (distribution of clock, energy, modularity). Asynchronous circuits show interesting potentials in several fields such as the design of microprocessors, smart cards and circuits with low power consumption [1]. The design of effective asynchronous circuits in terms of performance and correctness requires rigorous design and validation methods.
Typically, asynchronous circuits are specified at logic level, using a CCS or a CSP-like formalism [1]. Indeed, an asynchronous circuit can be seen as a set of communicating processes, which read a piece of data starting from input ports, carry out a treatment and finally write on output ports. In our work, asynchronous circuit specifications are written in an enriched version of the CHP (Communicating Hardware Processes) language initially developed by Alain Martin in Caltech [2].

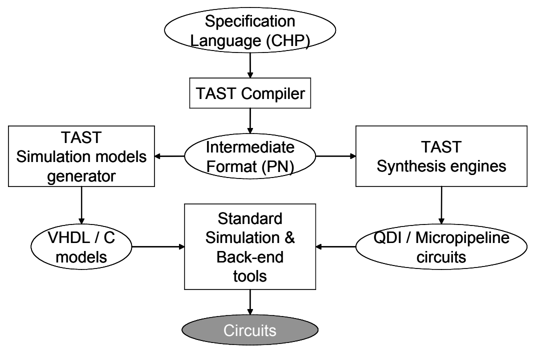

Specification Language (CHP)
TAST Compiler
TAST Simulation models generator
Intermediate Format (PN)
TAST Synthesis engines
VHDL / C models
Standard Simulation & Back-end tools
QDI / Micropipeline circuits
Circuits

Figure 1: the TAST design flow

The TAST design flow is shown in the square boxes of Figure 1 [12, 14]. CHP specifications are interpreted in terms of Petri Nets. Process decomposition and refinements are performed on the Petri Net formalization, from which a choice of architectural targets are available: micro-pipeline, quasi delay insensitive circuit, or synchronous circuit. Dedicated synthesis engines produce structural gate networks, in source VHDL, for simulation and back-end processing using commercial CAD tools. On the other hand, behavioral VHDL and C models can be generated for validation by simulation. This approach has supported, up to silicon fabrication, the design of two asynchronous microprocessors: ASPRO [3] and MICA [4].

The goal of our work is to introduce formal methods into the asynchronous synthesis flow. A first feasibility study was performed on an industrial symbolic model checker that is routinely used to perform property checking on RTL designs [5]. The approach consisted in translating the Petri Net, interpreted as a finite state machine, as a pseudo synchronous VHDL description, where the only purpose of the pseudo clock thus introduced was to make the result of each computation cycle a visible state. In essence, our translation performed a static pre-order reduction, by replacing all independent event interleavings by data-flow concurrency. Due to the semantics of process communication by signals at RT level, abstract communication channels had to be synthesized prior to property verification. This approach gives good results on models after communication expansions, i.e. after essential design decisions have been taken, among which data encoding and communication protocol. But this is too late in the design process to validate the initial specifications.

Our second approach consists in using formalisms and tools coming from the field of software validation, in particular distributed systems, whose model of execution is similar to the asynchronous circuits one. CHP specifications are analyzed and translated in terms of LTS (labeled transitions systems) with guarded commands. These LTS descriptions explicitly integrate channels and simple "read" and "write" actions. They appear to be more appropriate for the formal validation of initial CHP specifications.

Our work being devoted to specification semantics and verification methods rather than to the development of one more model-checker, we again looked for available existing software. We selected IF [8, 10], an intermediate format developed in Verimag laboratory to describe asynchronous processes and LTS. The construction of the IF descriptions is carried out so as to preserve the asynchronous specifications semantics in terms of choice structures, communication and concurrency. Resulting programs are compiled towards a LTS and eventually submitted to the CADP toolset for verification. To obtain a correct behavior, a set of environment assumptions must be associated to each communication channel, ensuring that each input wire behavior is correct. The environment behavior can be modeled as a concurrent IF process. In this approach, we also consider validating the specifications at lower levels, i.e. after the communications expansion according to a four phase handshake protocol. Then we can perform verification between specifications: before and after the communication expansion.

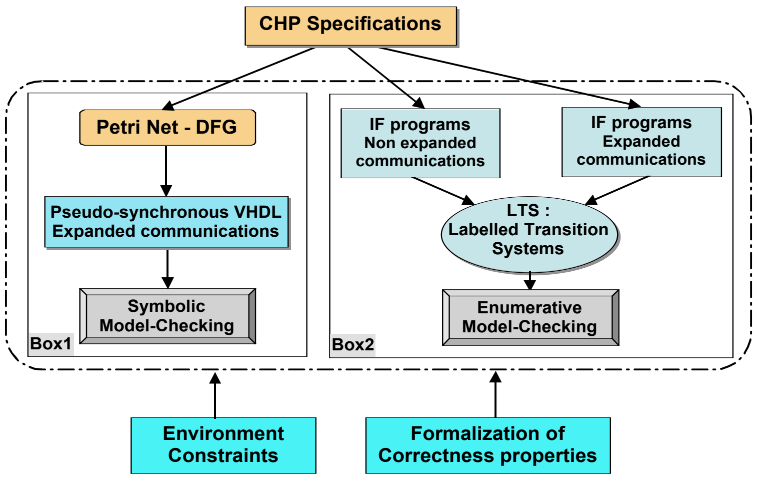

CHP Specifications
Petri Net - DFG
Pseudo-synchronous VHDL Expanded communications
Symbolic Model-Checking
Box1
IF programs Non expanded communications
IF programs Expanded communications
LTS : Labelled Transition Systems
Enumerative Model-Checking
Box2
Environment Constraints
Formalization of Correctness properties

Figure 2: Formal validation flow.

Figure 2 shows the formal validation flow for the two approaches. Box 1 corresponds to the pseudo-synchronous model, checked by symbolic methods. Box 2 displays the use of enumerative methods on asynchronous models. In both cases, we verify safety properties such as freedom from deadlock or livelock, and temporal behavior.

The rest of the paper is organized as follows. Section 2 reviews the essential features of CHP and its semantics both in terms of Petri Net and LTS. Section 3 reports the results of performance studies on the two approaches. In Section 4, as a case study, we present the applicability of the approach on a real-life asynchronous circuit: a four stage Filter. Section 5 ends the paper with our conclusions.

2. Translation from CHP to Petri Nets and IF

Numerous formalisms have been proposed to model the execution of concurrent processes. We consider here two of them, based respectively on Petri Nets and on the IF intermediate format. The operational semantics of these two models can be defined in terms of Labeled Transition Systems (LTS). Then we detail the translation schemes that have been implemented in the tool.

2.1 The Petri Nets and IF models

a) TAST Petri Nets

A Petri Net (PN) is a quadruple $<S, T, F, M_0>$ where S and T are finite disjoint sets of *places* and *transitions*, $F \subseteq (S \times T) \cup (T \times S)$ is an input/output *flow relation*, and $M_0 \in 2^S$ is the *initial marking*. The operational semantics of a PN is classically defined in terms of LTS by using *markings* (i.e., elements of 2^S) to represent global system states. A global transition holds between two markings M and M' if and only if there exists a set of PN transition $\mathbf{t} \subseteq T$, those input places are contained in M, and such that M' can be obtained from M by replacing the input places of $\mathbf{t}$ by their output places. Thus, a PN can easily describe the control flow of a process with concurrent statements.

Data operations can be taken into account on this basic model by extending a PN with a set of variables. Basic statements (data assignments) can then be attached to places, and conditions (or guards) can be attached to transitions. In the TAST PN model, variables are typed and three basic statements are available: variable assignments and value emission or reception via an external communication port.

b) The IF intermediate format

We consider here a subset of the IF intermediate format in which a system description is a quadruple <A, E, G, P> where A is a set of disjoint *extended automata* A_i, E a *composition expression*, G a set of (typed) *global variables* and P a set of *communication ports*. Each extended automaton A_i is a tuple $<Qi, Li, Ri, q_{0i}, Vi>$ where Qi is finite set of *control states*, Li a finite set of *labels*, $Ri \subseteq Qi \times Li \times Qi$ a *transition relation*, $q_{0i} \in Q$ the *initial state* and Vi a set of (typed) local variables. Labels are of the form "g com s" where g is a condition (a Boolean expression over $G \cup Vi$), com is a value emission (p!expression) or a value reception (p?x) via a communication port p of P, and s is a sequence of variable assignments. In addition control states can be declared *unstable*, to better tune the atomicity level of transitions (transition sequences between unstable states are considered atomic).

The communication primitive is based on *rendez-vous*: a value emission (*resp.* reception) on a given communication port p can be executed by automaton Ai if and only if another automaton Aj, synchronized with Ai on port p, is ready to perform a corresponding value reception (*resp.* emission) on the same port. The set of synchronization ports between each automata pair is given in a LOTOS-like algebraic style by the composition expression E. More precisely, this expression is built upon a general binary composition operator Ai |[G]| Aj, allowing to synchronize (by rendez-vous) the execution of automata Ai and Aj on the port set G. The complete operational semantics of the IF model in terms of LTS is detailed in [10].

2.2 CHP components

A TAST CHP component is made of its communication interface, a declaration part and a set of concurrent CHP processes. The communication interface is a directed port list (similar to VHDL) and the declaration part lists the channels used for inter-process communications. Channels interconnect processes through point-to-point, asymmetric, memory less, message passing and compatible protocol links.

The translation schemes we propose from a CHP component to the Petri Net and IF models are the following:

- **from CHP to PN:** each CHP process is translated into a single Petri Net, and inter-process communications are expanded with an explicit communication protocol based on global variables (see section 2.4 below);
- **from CHP to IF:** each CHP process is translated into an IF extended automaton and the channels declaration is translated into a corresponding composition expression.

These two parts are detailed in the rest of the section.

2.3 CHP Processes

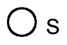
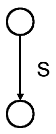
The abstract syntax of a TAST CHP process is given below:

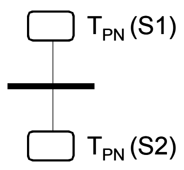
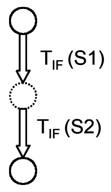
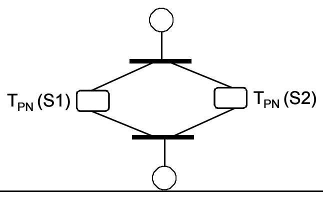
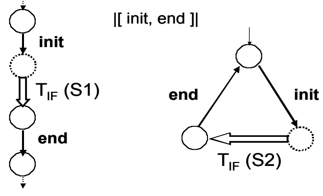
Process	**::=**	*Interface Declaration** *Statement**
Statement	**::=**	*Basic_Statement* **|** *Sequential_Composition* **|** *Parallel_Composition* **|** *Guarded_Commands*
Basic_Statement	**::=**	`var := expression` **|** `p!expression` **|** `p?var`
Sequential_Composition	**::=**	*Statement* `;` *Statement*
Parallel_Composition	**::=**	*Statement* `,` *Statement*
Guarded_Commands	**::=**	`@[` **(** *Guard* `=>` *Statement* `;` *Termination* **)*** `]`
Termination	**::=**	`loop` **|** `break`
Guard	**::=**	`expression` **|** `port#`

Roughly speaking, a process description consists in a communication interface, a declaration part and a statement part. The communication interface is a list of ports connected to local channels or component ports, and the declaration part is a list of typed local variables. The statement part is inspired from A. Martin's original CHP proposal: a statement can be either a *basic statement* (variable assignment, value emission or reception through a communication port), a *sequential* or *parallel* composition, or a general *guarded commands* block. A *guard* is a Boolean expression which may include a channel *probe* (denoted with the `#` operator).

The translation of the communication interface and data parts into the PN and IF models is rather immediate and we focus here on the translation of the statement part. First, all CHP basic statements have their direct counterparts in both PN and IF. Furthermore, CHP control structures (sequential and parallel composition, guarded commands) can also be easily expressed within these models using compositional "patterns". Therefore, the translation from a CHP process to PN and IF can be defined in a classical way by structural syntactic induction. Formally, we consider two translation functions, T_{PN} and T_{IF}, associating respectively a PN and IF pattern to each CHP term. These functions are (recursively) defined in Table 1 below:

CHP term t	T_{PN} (t)	T_{IF} (t)

Basic statement S:
`var := expression`
`p!expression`
`p?var`
		

Sequential composition: `S1 ; S2`		
Parallel composition: `S1 , S2`		 |[init, end]| init T_{IF} (S1) end end init T_{IF} (S2)

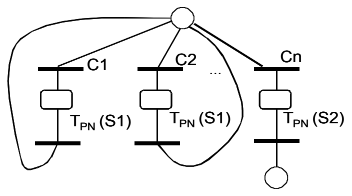
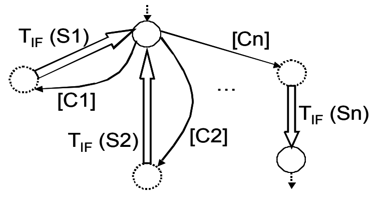
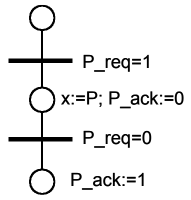
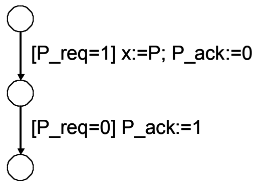
Guarded Commands:
`@[C1 => S1; loop`
`  C2 => S2; loop`
`     ...`
`  Cn => Sn; break]`
	 C1
	C2
	...
	Cn
	T_{PN} (S1)
	T_{PN} (S1)
	T_{PN} (S2)
		 T_{IF} (S1)
		[Cn]
		[C1]
		...
		T_{IF} (Sn)
		T_{IF} (S2)
		[C2]

Table 1: Translation of CHP main statements

The only non-straightforward case is the translation of CHP parallel statement composition into IF (T_{IF} (S1, S2)). Since parallelism is not allowed *inside* IF extended automata we need to introduce an auxiliary automaton for statement S2, to be synchronized with S1 via two extra communication ports init and end.

2.4 Inter–process communications and probes

CHP inter-process communications (value emission p!expression and value reception p?x) are based on a high-level *rendez-vous* mechanism. Their translation into IF is therefore straightforward since the corresponding primitive is available. This makes possible, at reasonable cost, the verification of a circuit design at a specification level. However, to be synthesized on a gate level, this high-level communication primitive has to be expanded into a concrete protocol. The protocol we choose here is a (classical) four-phase handshake protocol based on global variables: for each channel P, a variable P stores the exchanged value, and two Boolean variables P_req and P_ack manage the handshake (in practice P_req and P can be encoded using a single variable). Table 2 shows the translation of this expansion phase into both PN and IF, assuming that the "sender" process initiates the communication (it is *active* on this channel) and that the "receiver" waits for a value (it is *passive* on this channel). On the reverse situation (active reception and passive emission) a dual protocol has to be applied.

Communication expansion	PN	IF
(passive) value reception `p?x`		

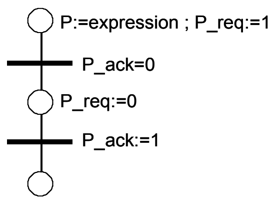
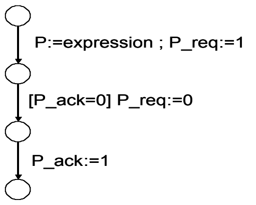
(active) value emission `p!expression`		

Table 2: Communication expansion

This communication expansion also provides a straightforward translation of the CHP *probe* operator. This Boolean operator allows a process to check on a passive channel P if another process is waiting for a communication on P. This check can then simply be done by testing the value of variable P_req.

2.5 Optimizations

The efficiency of the IF verification tools is drastically influenced by the size of the underlying LTS associated to the IF model. It is therefore crucial to keep this size as small as possible. To this purpose, the general translation scheme proposed above can be improved in some particular situations. These optimizations fall into two categories:

1. **Transition atomicity:** IF transitions can be labeled with a guard, a communication and several variable assignments. In some cases this transition structure allows to suppress intermediate control states produced by function T_{IF}, or at least to explicitly declare them as *unstable*. This increases the atomicity level of the transition, and thus reduces the size of the underlying LTS (in particular when this transition is executed in parallel with transitions of other processes). These extra states are the "dotted" ones in Table 1, and they can be suppressed whenever the subsequent statement Si is a *basic* statement.

2. **Explicit interleaving:** The use of an auxiliary IF automaton to translate CHP parallel statement composition is also very "state consuming" (extra communications init and end increase the size of the underlying LTS). Whenever statements S1 and S2 are basic statements, a better solution consists in translating the parallel composition `S1,S2` with an explicit transition interleaving. Note that this interleaving can even be removed whenever S1 and S2 are independent statements (e.g., assignments to distinct local variables). Figure 3 below illustrates this optimized translation of CHP statement `S1,S2` in both situations. The "dotted" state can be suppressed if S2 is a basic variable assignment.
   These optimizations have been integrated into the CHP to IF translation tool.

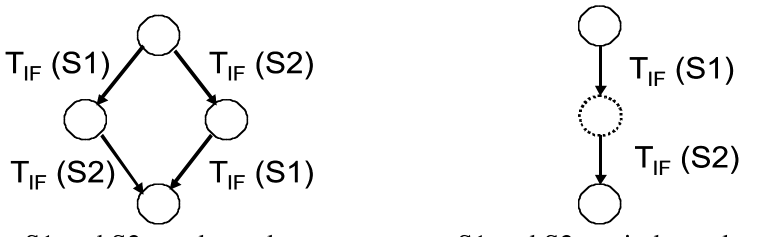


Figure 3: explicit interleaving of parallel composition

3. Performance study

Figure 4 shows the Petri-Net of a selector circuit, that we used as benchmark. The behavior is the following. 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.

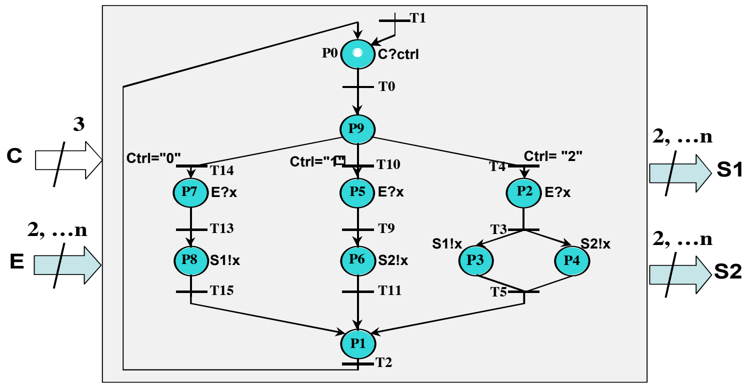


Figure 4: the basic selector

We have varied the bit-width of the data channels (E, S1, S2), the size of control channel C and consequently the number of alternative paths (see Figure 5.a), and the number of concurrent paths (see Figure 5.b). We measured verification time, maximum memory occupancy and BDD/LTS size on two verification systems: a commercial symbolic model checker, and the CADP tool set.

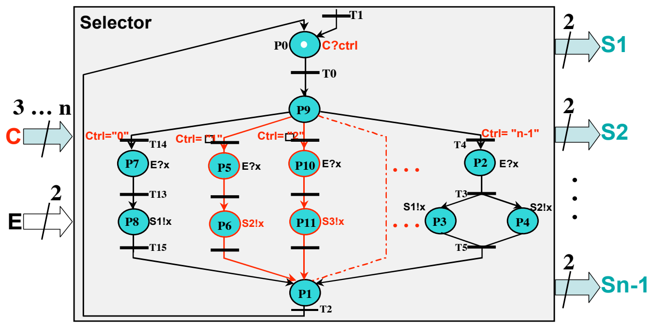
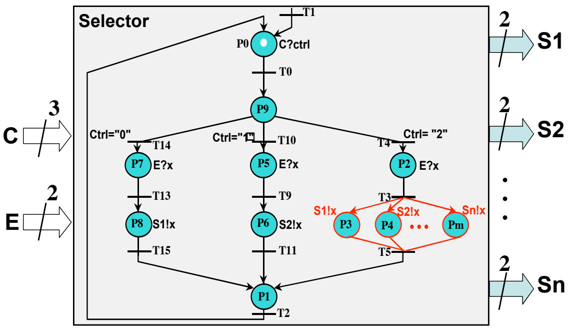



Figure 5.a Varying control size **Figure 5.b Varying number of concurrent paths,**

To conduct the experiment with the first system, we applied the "pseudo-synchronization" principle described in [5], and translated the Petri Net into synthesizable RTL VHDL. At this level, *rendez-vous* on a channel is not available, and communications must be expanded. In the second case, we translated the CHP description into IF, as described in section 2, and produced a model for the two cases: with and without expansion of the communications.

The experiments reported here concern a typical selector characteristic property, expressed in the appropriate property language for each tool: *If an incoming request (C=x"1") arrives, then a write will eventually occur on S1*. The results, computed on a SUN-Blade-100 Sparc with 640 MB memory are displayed in Tables 3,4,5. In the three Tables, for IF/CADP, we measure separately the model generation and the property verification times: the model can be stored and directly reused for another property, and the information was easy to get. For FormalCheck, the verification time includes the construction of the model. Time is expressed in the form mn:sec or mn:sec,ts (ts is tenth of a second).

Table 3 shows the effect of varying the bit width of the input-output data (Figure 4). CADP shows little sensitivity to the data width parameter when communications are not expanded. In the expanded case, both systems show a space and time explosion, the enumerative CADP performs better under 16 bits, and the symbolic FormalCheck performs better above.

Table 4 shows the effect of varying the size of the control input and the number of *alternative* execution paths (Figure 5.a). Once again, CADP performs very well when communications are not expanded, and in the expanded model, 16 bits seems to be the threshold size where the symbolic model of FormalCheck outperforms the LTS enumeration of CADP.

Table 5 gives the results obtained in fixing the data size to a minimum, and in augmenting the number of outputs that are concurrently written (Figure 5.b). Clearly, the amount of concurrent paths is hardest on a system that enumerates all possible event inter-leavings. The memory needed for model

	Data size (E, S1 and S2)	2	4	8	16	24	32
a) Measures without expansion of communications							
CADP/ IF	Size of LTS: trans / states	0.42+e2 / 0.2+e2	1.08+e2 / 0.44+e2	3.12+e2 / 0.88+e2	1.01+e3 / 1.77+e2	2.09+e3 / 2.64+e2	3.55+e3 / 3.52+e2
	Memory size	8168 KB	8168 KB	8176 KB	8184 KB	8224 KB	8232 KB
	Model generation Time	00:00,5	00:00,5	00:00,5	00:00,6	00:00,6	00:00,6
	Verification time	00:00,1	00:00,1	00:00,1	00:00,1	00:00,1	00:00,2
b) Measures with expansion of communications							
CADP/ IF	Size of LTS: trans / states	1.86+e4 / 5.66+e3	5.18+e4 / 1.53+e4	1.65+e5 / 4.77+e4	5.59+e6 / 1.64+e5	1.24+e6 / 3.51+e5	2.15+e6 / 6.06+e5
	Memory size	4400 KB	5624 KB	8160 KB	18 MB	39 MB	83 MB
	Model generation Time	00:02	00:05	00:17	01:27	04:18	09:48
	Verification time	00:00,5	00:01	00:04	00:12	00:29	00:53
Formal Check	Reachable states	2.71 e+03	1.07 e+04	1.71 e+05	4.38 e+07	1.12 e+10	2.87 e+12
	Memory size	3.26 MB	3.84 MB	4.32 MB	7.60 MB	9.21 MB	15.37 MB
	Verification time	00:07	00:09	00:11	00:21	00:27	00:43

Table 3: Varying Data size

	Size of control (C)	2	4	8	16	24	32
a) Measures without expansion of communications							
CADP/ IF	Size of LTS: trans / states	0.42+e2 / 0.2+e2	0.62+e2 / 0.32+e2	1.82+e2 / 0.52+e2	5.84+e2 / 0.98+e2	1.26+e3 / 1.46+e2	2.25+e3 / 1.96+e2
	Memory size	8176 KB	8184 KB	8184 KB	8192 KB	8224 KB	8242 KB
	Model generation Time	00:00,5	00:00,6	00:00,8	00:00,9	00:01	00:01
	Verification time	00:00,1	00:00,1	00:00,1	00:00,1	00:00,1	00:00,1
b) Measures with expansion of communications							
CADP/ IF	Size of LTS: trans / states	1.86+e4 / 5.66+e3	6.37+e4 / 1.68+e4	6.65+e5 / 2.01+e4	2.01+e5 / 6.08+e4	0.07+e5 / 1.23+e5	
	Memory size	4400 MB	5352 MB	5872 MB	9808 MB	19 MB	
	Model generation Time	00:02	00:06	00:08	00:35	01:35	
	Verification time	00:00,6	00:01	00:01	00:04	00:08	
Formal Check	Reachable states	2.71 e+03	8.62 e+03	1.33 e+06	4.30 e+10	9.92 e+23	4.75 e+30
	Memory size	3.26 MB	4.45 MB	7.54 MB	14.04 MB	24.85 MB	38.89 MB
	Verification time	00:07	00:10	00:10	00:41	10:31	21:36

Table 4: Varying size of control channel

	A number of concurrent writings	1	2	4	8	16	24	32
a) Measures without expansion of communications								
CADP/ IF	Size of LTS: trans / states	0.36+e2 / 0.18+e2	0.42+e2 / 0.2+e2	0.98+e2 / 0.46+e2	2.08+e3 / 5.26+e2	1.05+e6 / 1.31+e5		
	Memory size	8168 KB	8168 KB	8192 KB	8768 KB	17 MB		
	Model generation Time	00:00,5	00:00,5	00:00,9	00:15	01:19		
	Verification time	00:00,1	00:00,1	00:00,2	00:00,2	00:15		
b) Measures with expansion of communications								
CADP/ IF	Size of LTS: trans / states	3.2+e4 / 8.78+e3	4.28+e4 / 1.13+e4	5.94+e5 / 1.11+e5	?			
	Memory size	4207 KB	5024 KB	13 MB	>1235 MB			
	Model generation Time	00:03	00:04	00:48	> 156:00			
	Verification time	00:01	00:01	00:11				
Formal Check	Reachable states	5.38 e+03	5.38 e+03	1.07 e+04	1.71 e+05	4.38 e+07	1.12 e+10	2.87 e+12
	Memory size	4.23 MB	3.76 MB	5.56 MB	6.52 MB	11.21 MB	12.94 MB	20.51 MB
	Verification time	00:05	00:05	00:07	00:09	00:13	00:22	00:32

Table 5: Varying concurrent paths number

generation is the limiting performance factor: for instance, in the expanded case, for 8 concurrent output writings, the construction was manually halted after two days, as all the time was spent on swap. In the case of this experiment, the clear performance advantage obtained with FormalCheck should be weighted against the fact that the pseudo-synchronous model benefits from a pre-order reduction of events inter-leavings, optimization that was not performed on the IF model processed by CADP.

These experiments are one more demonstration that brute force model checking will not provide satisfactory answers to the problem of asynchronous verification. As could be expected, essential properties should be checked at the highest possible specification level, and the verification engineer should be prepared to apply a variety of reduction techniques to fight the model size explosion.

The next section shows the application of these principles to the design of an asynchronous filter.

4. Case study: a four-tap FIR filter

Figure 6 shows a four-tap FIR filter, with 8-bit input/output data, and 4 coefficients. This filter is defined by the following transfer function:

$$Output(n) = \sum_{i=0}^{N-1} Coeff(i)\ Input(n-i)\ (N=4)$$

The filter behavior is modeled by 13 CHP processes, whose structure is shown on Fig. 6. FSM and Accumulator contain memorizing elements and are each modeled by two processes for synthesis purposes [14].

4.1 Modeling the Filter in IF

The CHP Component is described by an IF system, and each CHP process is represented by an IF process. To illustrate the translation, we consider one typical process: *Multiplexer (Figure 7).*

In this process, the control channel (*Ad_mux*) is typed MR[4][1], i.e. one-of-four data encoding. Channel *Ad-mux* is read in the local variable *"address"*. According to the value of "*address*", one of the four channels (*Mux1, Mux2, Mux3 or Mux4*) is read and its value is written on channel *out_mux*.

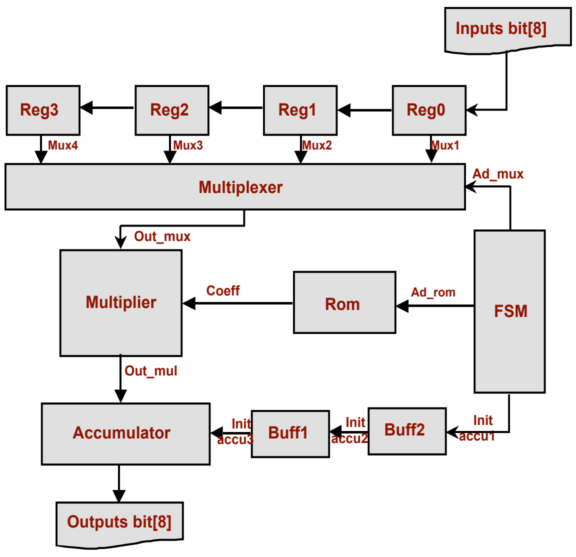
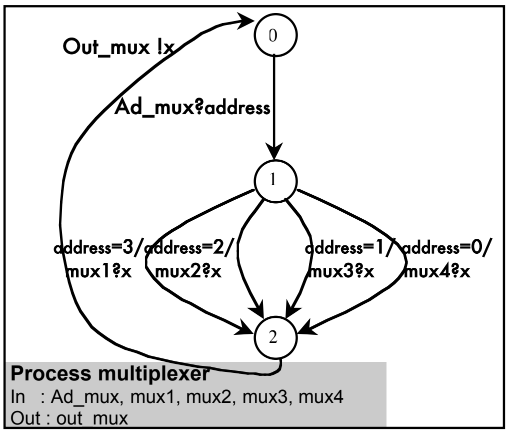


Figure 6: Filter behavioral architecture

```
process multiplexer
port (  Ad_mux  : in DI mr[4][1];
        Mux1, Mux2, Mux3, Mux4    : in DI mr[2];
       out_mux : out DI mr[2][8]
    )
variable address : mr[4][1];
variable x: mr[2][8];
begin
[ @[Ad_mux?address;
        [ address = 3 => Mux1?x; break
          address = 2 => Mux2?x; break
          address = 1 => Mux3?x; break
          address = 0 => Mux4?x; break
        ];
   out_mux!x ] ; loop];
end;
```

(a) CHP code of Multiplexer



(b) IF representation of Multiplexer

Figure 7 : Behavioral specification of Multiplexer

4.2 Some verified properties

Characteristic properties of the filter behavior have been expressed in mu-calculus, and automatically verified using CADP, on a SUN Ultra 250 with 1.6 GB memory. For some of them, their meaning, and overall verification time (LTS generation time + model checking time) are listed below.

P1: Freedom from deadlock
Verification time: 1,77 + 7,27 = 9,04 sec

P2: Every input triggers an output
Verification time: 1, 77 + 9,68 = 11,45 sec

P3: The Multiplexer must receive the inputs (Muxi) in the following order: Mux4, Mux3, Mux2 and Mux1.
Verification time: 1, 77 + 30,55 = 32,32 sec

P4: No new input is read before Mux1 is read (convolution rule).
Verification time: 1, 77 + 8,67 = 10,44 sec

4.3 Verification by behavior reduction

To verify Property P3, which relates to the Mux(i) channels only, an alternative technique is available. The filter behavior is reduced by hiding all the labels which do not relate to the Mux(i) channels. This can be obtained by applying property-preserving equivalence reduction techniques. The resulting LTS is depicted on Figure 8, which shows that the Multiplexer reads the Mux(i) inputs in the correct order.

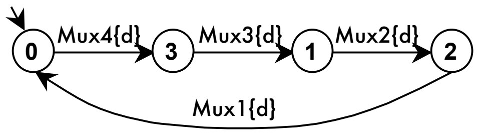


Figure 8: Reduced LTS for property P3

4.4 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 2.5). One order of magnitude was gained in LTS size (from 2,9 e+5 to 7,4 e+4 states, from 1,5 e+6 to 3,2 e+5 transitions).

5. Conclusion

We have implemented a 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 certainly involve elaborate model reduction strategies, some of which are still not automated. Finally, replacing the mu-calculus, the current property notation formalism in CADP, by a more widely accepted property specification language such as the Accelera PSL would ease the designer's access to the verification toolbox.

6. References

[1] 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.

[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] M. Renaudin, P. Vivet, F. Robin, "ASPRO-216□ a standard-cell Q.D.I. 16-bit RISC asynchronous microprocessor", Proc. of the Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC'98), San Diego - USA, 1998, p.□2-31.

[4] A. Abrial, J. Bouvier, M. Renaudin, P. Senn and P. Vivet, "A New Contactless Smart Card IC using On-Chip Antenna and Asynchronous Microcontroller", Journal of Solid-State Circuits, Vol. 36, 2001, pp. 1101-1107.

[5] D. Borrione, M. Boubekeur, E. Dumitrescu, M. Renaudin, 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] M. Bozga, S. Graf, L. Mounier "Automated validation of distributed software using the IF environment", Proc. Workshop on Software Model-checking, Electronic Notes in Theoretical Computer Science vol. 55 Elsevier Science Publishers July 2000.

[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] Anh Vu Dinh Duc, L. Fesquet, M. Renaudin, "Synthesis of QDI Asynchronous Circuits from DTL-style Petri-Net" IWLS-02, 11th IEEE/ACM Internat. Workshop on Logic & Synthesis, New Orleans, Louisiana, June 4-7, 2002.

[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] Rajit Manohar, Tak-Kwan Lee, and Alain J. Martin. "Projection: A Synthesis Technique for Concurrent Systems". Proc. 5th Int. Symposium on Advanced. Research in Asynchronous Circuits and Systems, April 1999.