Improving Compositional Verification of State-based Models by Reducing Modular Unbalance

Mauricio Varea
m.varea@ecs.soton.ac.uk

Michael Leuschel
mal@ecs.soton.ac.uk

Bashir M. Al-Hashimi
bmah@ecs.soton.ac.uk

Department of Electronics and Computer Science
University of Southampton, SO17 1BJ, UK

Abstract

Compositional Verification is a viable way to tackle the state explosion problem. However, the decomposition of a system into smaller parts is not a trivial problem, and dividing the specification into modules can be regarded as one of the main issues that concerns a compositional approach. This paper concentrates on the application of compositional verification to state-based models, in order to reduce the number of nodes assigned to memory, thus avoiding state explosion and speeding up the verification. Furthermore, we investigate and propose an estimation method that improves the compositional verification process in modular designs, such that the amount of memory required by the process is minimised. This method has been applied to a real-life embedded system, producing meaningful results without the need of data abstraction.

1 Introduction

In the design of large embedded systems, the designer often faces a trade-off as to which validation technique to use in order guarantee that the final implementation will meet the specification. On the one hand, formal verification techniques [4] aim to mathematically prove the correctness of a specification, but suffer from state explosion. On the other, simulation [5], testing [1], and refinement [2] can cope with larger specifications, but they do not cover the entire state space. The key aspect here is that simulation and testing are feasible for large models because they do not attempt to validate the entire specification *at once*. In other words, breaking down the specification into smaller parts, called *modules*, would make the entire system specification formally verifiable, *i.e.*, the model checker will give an answer as to whether or not the specification satisfies a temporal logic property, *in a finite time*. This type of verification, known as modular or compositional verification, is a divide-and-conquer approach that uses natural divisions in the specification in order to partition the structure and, therefore, reduce the number of states involved in the process.

Compositional verification was introduced in [8], aiming to reduce the complexity of large designs validation. Since then, the problem has been studied in several formalisms [11, 15, 20], but not so often applied in computer-aided verification of real-world applications. Only in recent years there has been some increasing research effort towards developing tools and techniques to validate real-life embedded systems by compositional verification [22]. This includes, *e.g.*, microprocessor architectures [16, 21], hardware protocols [23], multimedia SoC [28] and multi-agent systems [17].

The main contribution of this paper is an estimation of the complexity involved in the verification process when a modular approach is taken, which can be used to decide *a priori* which way to partition a state-based model. The estimator introduced in this paper is relevant for the design in terms of memory resources needed for the formal verification process. To the best of our knowledge, such estimator has not been developed in previous work. The concepts introduced in this paper are also applied to a real-world embedded system specification through a control/data-flow unbiased internal design representation, the Dual Flow Net (DFN) model [29, 30].

This work is organised as follows. Section 2 reviews the theory behind compositional verification. In Section 3 we present a motivational example which will show the need for an estimation method. Section 4 introduces a new method for estimating the complexity involved in the verification process, when compositional verification is applied. In Section 5 we illustrate the applicability of the estimation method presented through an Ethernet coprocessor specification. Finally, Section 6 outlines some conclusions and future trends of investigation.

2 Compositional Verification

It is known that a small increment in the size of the model structure $\mathcal{M}$, will result in a several-order-of-magnitude bigger state space to be explored [14]. Indeed, the verification complexity, *i.e.*, cost of applying formal verification to a system description in terms of memory resources, has been proved to be PSPACE-complete [18, 19]. Therefore, by decomposing $\mathcal{M}$ into n modules $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\}$, in such a way that the parallel composition of all these modules $\mathcal{M}_1 \parallel \mathcal{M}_2 \cdots \parallel \mathcal{M}_n$ is equivalent to the original structure $\mathcal{M}$, it is possible to significantly reduce the number of states searched by the model checker, because the complexity is reduced. However, a model checker often uses some heuristic in order to avoid redundancy of larger state spaces and, by exploiting BDD's features, also reduces the amount of BDD nodes allocated in memory. This means that a large number of very small modules $\mathcal{M}_i$ does not necessarily lead to the most efficient way to perform model checking, since they would not share any path in the verification process. In fact, there is a trade-off between having a vast number of small modules, which can have quite a substantial redundancy factor, and having only a few modules that are larger in size, which takes more memory resources due to the exponential rate of increase in the verification complexity.

Mathematically, the compositional verification of a structure $\mathcal{M}$ that has been constructed by the parallel composition (*i.e.*, $\parallel$) of modules $\mathcal{M}_i$, is formulated as:

$$\frac{\begin{array}{l}\mathcal{M}_1 \models \varphi_1 \\ \mathcal{M}_2 \models \varphi_2 \\ \ldots \\ \mathcal{M}_n \models \varphi_n \\ f(\varphi_1, \ldots, \varphi_n) \implies \varphi\end{array}}{\mathcal{M}_1 \parallel \mathcal{M}_2 \cdots \parallel \mathcal{M}_n \models \varphi}$$

which means that proving that each module $\mathcal{M}_i$, $\forall 1 \leqslant i \leqslant n$, satisfies the temporal logic formula φ_i, and also proving that each formula φ_i is related to the property φ (through a logical function f), the conclusion that the entire system $\mathcal{M}$ satisfies the property φ can be drawn.

A key issue in compositional verification is the *assume-guarantee* paradigm [24, 27], which bases its reasoning in separating a system in two parts: the module $\mathcal{M}'$ and the environment $\mathcal{M}''$. Guarantees $\mathcal{G}_i$ are properties of $\mathcal{M}'$ which are verified assuming that $\mathcal{M}''$ satisfies some assumptions $\mathcal{A}_j$. By appropriately combining a set of assumptions $\mathcal{A}$ and a set of guarantees $\mathcal{G}$, it is possible to infer the correctness of the entire system $\mathcal{M}$ without actually building the global state-transition graph.

Let $\mathcal{M}'$ be any module of the system, and $\mathcal{M}''$ its environment. This is, $\mathcal{M} = \mathcal{M}' \parallel \mathcal{M}''$. The assume-guarantee paradigm states that:

$$\frac{\begin{array}{l}\mathcal{M}'' \models \mathcal{A} \\ \mathcal{A} \parallel \mathcal{M}' \models \mathcal{G} \\ \mathcal{M}'' \parallel \mathcal{T}_{\mathcal{G}} \models \varphi\end{array}}{\mathcal{M} \models \varphi}$$

Where $\mathcal{T}_{\mathcal{G}}$ is the "tableau of the property $\mathcal{G}$", as in [20].

3 Motivational Example

Assume that a ten-stage pipeline system needs to be formally verified. The property to be verified is simply that some data put at the beginning of the pipeline will propagate and eventually reach the end. The minimum unit of functionality of the system is each stage itself, but we will suppose that the system can only be partitioned into three separate modules, which will be assigned to three different hardware resources.

Partition	BDD nodes
$\langle 1,2,7 \rangle$	2706
$\langle 1,3,6 \rangle$	2135
$\langle 2,3,5 \rangle$	1912
$\langle 2,4,4 \rangle$	1851
$\langle 3,3,4 \rangle$	1830

Table 1: Sizes of BDD trees for a ten-stage pipeline

Table 1 shows that different ways of partitioning this pipeline, *i.e.*, grouping these pipeline stages, leads to different complexities in terms of BDD nodes allocated to memory. The notation $\langle x,y,z \rangle$ indicates the size of each partition, where $x+y+z$ is the total size of of the system. This variety of partitioning schema raises the vexed question of how to identify which partitions would lead to better results in compositional verification.

4 The Estimation Method

Intuitively, a four-module system which has a partition $\mathcal{P} = \langle 4,4,4,4 \rangle$ is more balanced than the same system using other partition $\mathcal{P}' = \langle 1,9,4,2 \rangle$, where the elements in the tuple indicates the number of elementary units (transitions $t \in T$) in each module. More formally, Definition 1 defines the *modular unbalance* of a Kripke structure which has been partitioned under a partition scheme $\mathcal{P}$.

Definition 1 *The modular unbalance of a Kripke structure $\mathcal{M} = \mathcal{M}_1 \parallel \mathcal{M}_2 \cdots \parallel \mathcal{M}_n$ is given by:*

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (m_i - \overline{m})^2} \qquad (1)$$

where $\overline{m}$ is the statistical mean (average) of the sizes of the modules, i.e., $\overline{m} = m/n$.

It is clear from (1) that the unbalance is defined as the *standard deviation* of the size of each module w.r.t. the average size $\overline{m} = m/n$. This can be seen as making balance analogue to volatility, since standard deviation is one of the most common ways to assess the volatility of a discrete variable. Thus, $\sigma \to 0$ corresponds to a very balanced system while $\sigma \to \infty$ is in accordance to an extremely unbalanced system. A system with $\sigma = 0$ will be called *perfectly balanced*, and a system with the greatest σ possible will be called *perfectly unbalanced*.

We make then the following proposition.

Conjecture 1 *Among all possible partitions of a system $\mathcal{M}$, the one which minimises σ will also minimise the verification complexity.*

The intuition, again, would serve to describe the feature described in Proposition 1. Suppose a system $\mathcal{M}$, which has a partition scheme $\mathcal{P}$, which is perfectly balanced. Then, changing $\mathcal{P}$ into $\mathcal{P}'$ implies that at least one transition $t \in T$ has to be removed from a module $\mathcal{M}_i$ and added into other module $\mathcal{M}_j$. This will lead to a decrement in the complexity verifying $\mathcal{M}_i$ and an increment in the complexity verifying $\mathcal{M}_j$. Since the exponential nature of BDD's state space, it is more likely that the improvement due to the reduction of $\mathcal{M}_i$ is less than the worsening caused by the expansion of $\mathcal{M}_j$. Therefore, the overall increasing of the verification complexity is accompanied by an increase of σ, and vice versa.

Partition	σ	BDD nodes
$\langle 1,2,7 \rangle$	2.6247	2706
$\langle 1,3,6 \rangle$	2.0548	2135
$\langle 2,3,5 \rangle$	1.2472	1912
$\langle 2,4,4 \rangle$	0.9428	1851
$\langle 3,3,4 \rangle$	0.4714	1830

Table 2: Estimating the pipeline complexity

Coming back to the example presented in Section 3, Table 2 show that the more balance, *i.e.* $\sigma \to 0$, the less number of BDD nodes allocated in memory. This leads to an improvement of, in this case, 32% in the resources allocated to the BDD tree, although in all cases the verification is performed over the same system with the same properties. Therefore, this has provided an answer to the question raised earlier and we believe that it is important to include σ as an estimation method to guide the partition of a system which will be further verified by compositional methods.

However, the size of each module is not the only issue that concerns the verification complexity. The coupling among modules is equally important for verification purposes, and being able to find the proper $\mathcal{T}_{\mathcal{G}}$ determines the satisfaction of the results. Due to the internal behaviour of the model checker, which unfolds cycles to perform the verification, all results hitherto presented assume that modules are set to be free of cycles. Otherwise, well balanced partitions may have bigger unfolded structure than others less balanced and, thus, more complex BDD trees. This is illustrated in Figure 1, where the solid line indicates the results shown in Table 2 while the dashed line represents the same pipeline constructed with a cycle in the specifications. It can be observed that, between two partitions $\mathcal{P}$ and $\mathcal{P}'$, with $\sigma < \sigma'$, the complexity of $\mathcal{P}$ will be less than the one from $\mathcal{P}'$, only in the *acyclic* case.

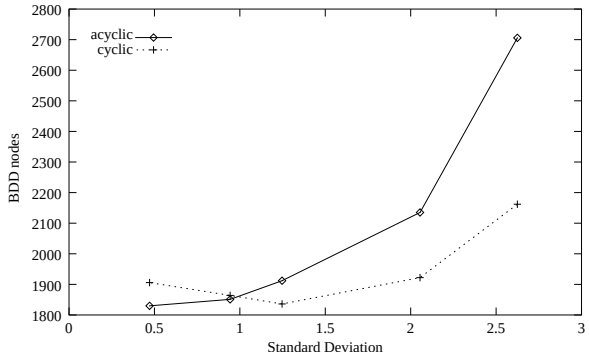

Figure 1: Cyclic and acyclic complexity

An automatic approach to the compositional verification problem would search through a space of possible modular partitions of the DFN model, in order to obtain a the one with minimum σ. Although this approach goes beyond the scope of this paper, we will give an insight of the theoretical limits of our methodology.

It has been proven in [13] that the number of distinct partitions for a modular design is:

$$\#\mathcal{P}(m) \approx \frac{\exp\left[\pi\sqrt{\frac{2m}{3}}\right]}{4m\sqrt{3}} \qquad (2)$$

where $m = |T|$ is the number of transitions in the model. However, we argue that not all $\mathcal{P}_i$, $1 \leqslant i \leqslant \#\mathcal{P}(m)$, are practically possible. There might be cases where the strong dependency among some transitions and the weak dependency among others, may bias the modularisation towards some results. This will restrict the vast exponential approximation given in (2) to a handful of real possibilities.

5 A Real-life Example

This section applies the estimation method proposed to a real-life embedded system, *i.e.*, the Ethernet network's coprocessor, in order to show that the compositional verification of such systems also benefits from the improvements introduced by the method. The Ethernet coprocessor is a chip advocated to transmitting and receiving data frames over a communication channel by means of the CSMA/CD protocol, which is defined in the IEEE 802.3 standard [3]. In order to model the coprocessor, a control/data-flow unbiased internal design representation has been used, and the formal verification process has been carried out with the use of the *Cadence SMV* tool [6]. The platform used was a Sparc Sun-Ultra 10 / 440 MHz with 512 Mb RAM running Solaris 8.

5.1 The Model: Dual Flow Nets

The Dual Flow Net (DFN) model [29, 30] is an extension to the Petri net (PN) model which introduces a combined control/data-flow analysis of systems. This model utilises a set of vertices (P) to represent the state/storage elements in the system, another set of vertices (T) to capture the control flow, and an additional set (Q), not present in PNs, for capturing the transformational elements in the system. A set of arcs $F \subseteq (P \times T) \cup (T \times P) \cup (P \times Q) \cup (Q \times P) \cup (T \times Q) \cup (Q \times T)$ and a weight function $W : F \mapsto \mathbb{Z} \backslash \varnothing$ define the model structure, as well as a marking function defined in the domain of the complex numbers ($\mu : P \mapsto \mathbb{C}$) characterise its behaviour. In addition, there also exist a guard function $G : T \mapsto \sharp \cup \{\top\}$, where $\sharp$ is a finite set of symbols used for comparison, and an offset function $H : Q \mapsto \mathbb{Z}$, which complements the functionality of the model by allowing some arithmetical and logical functions to take place.

Since the DFN model has an enhanced structure, *i.e.*, it contains an additional set (Q) which explicitly captures data activity, this model can be used to verify embedded systems [31]. If a model such as PN was used instead, only the control part would have been verified. Thus, the underlying PN that is visualised when all transformational elements are removed from a DFN model, behaves with the same enabling and firing rule of a classical PN. However, on each transition firing a number of operations take place in the data domain. Further discussion on this model, as well as a formal definition of its principles, the motivation and some introductory examples, can be found in [29].

5.2 An Ethernet network coprocessor

The Ethernet network coprocessor, as studied in [12] and [26], is a highly structured protocol. This makes it very suitable for a benchmark of real-life complexity, mainly if compositional methods are going to be applied.

There have been a few attempts to perform formal verification of this coprocessor. For example, one of the earliest work on the verification of the Ethernet protocol was presented in [25]. This approach has used the SMV tool[1] to verify both the asynchronous and the synchronous model [32] of the Ethernet. Later, another approach directly implemented in C has presented the formal verification of some liveness properties using approximations to cope with the state explosion [10].

The operation of the coprocessor is ruled by the execution unit, exec_unit, which sends the starting memory address to the transmit unit (composed of a frame packager xmit_frame, a direct memory access (DMA) unit dma_xmit), and a serial transmitter $\mathsf{xmit_bit}$) and then enables the DMA unit to operate straight into memory.

The dma_xmit unit directly reads from the successive memory locations in order to obtain destination address, data length, and the actual data, which are then sent to the xmit_frame unit. There are two modes of operation in the dma_xmit unit: *dma_xmit_normal* and *dma_xmit_cancel*, so that this unit normally stays in the first mode but, if a failure occurs in the transmission to xmit_frame, the DMA unit switches to an alternative mode that sets up the environment to restart the transmission process.

The specification presented above has been captured by the DFN model introduced in Section 5.1. Figure 2 shows the complete model consisting of 49 places, 36 transitions and 12 hulls. As commented in Section 2, a model with 36 transitions would imply that there are 19,370 ways to group these transitions into different modules. However, by identifying threads of executions with no cycles in it (*e.g.* $\{t_7, t_{34}, t_8\}$), the number of practically implementable modules is reduced from 36 to just 8, which means that $\#\mathcal{P}(m)$ is also dramatically reduced (to 26). This modules have been denoted by: $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_8$. As a matter of notational convenience, the one place that precedes a module $\mathcal{M}_i$ has been labelled $p_i, \forall 1 < i < 8$, and highlighted in Figure 2. The importance of identifying these places lies on the fact that any token coming into the area of $\mathcal{M}_i$, has to go through p_i. That is the basis for applying modularisation of such DFN models.

Different alternatives for the modularisation are shown in Table 3. The ∞ symbol indicates that state explosion has occurred and the model checker was unable to find a solution in a reasonable amount of time. Further results

[1] Note the difference between the SMV tool from CMU [9] and the Cadence SMV verification suite [6].

σ	BDD nodes
5.290	∞
5.123	∞
4.847	∞
4.583	∞
3.564	2850325
3.000	1128184
2.964	852804
2.742	533619

Table 3: Balance vs. complexity

presented in this section, are based on the row which has the less σ.

In order to prove the correctness of the Ethernet coprocessor by compositional verification, 22 temporal formulas have been used. Table 4 shows the verification results for each property, where the first 9 are *guarantees* $\mathcal{G}_i$, and the remaining 13 are *assumptions* $\mathcal{A}_j$. It can be observed, from the fourth column, the complexity of the verification measured by the amount of BDD nodes allocated into memory.

The behaviour of the model is captured through the guarantee set of properties. For example, when a txstart signal is sent to the DMA unit, this will request access to the CPU (*c.f.* $\mathcal{G}_{\text{acc}}$). Then, the system reads from successive memory locations (*c.f.* $\mathcal{G}_{\text{s1}}$, $\mathcal{G}_{\text{s2}}$ and $\mathcal{G}_{\text{s3}}$), starting from $\text{txaddress}_{[16]}$ (*c.f.* $\mathcal{G}_{\text{from}}$). At this point, the Ethernet coprocessor is ready to transmit $\text{Bdata}_{[16]}$ to xmit_frame, but this is a 8 bit unit. So, we formulate $\mathcal{G}_{\text{high}}$ and $\mathcal{G}_{\text{low}}$ in order to prove that both $\text{Bdata}_{[15..8]}$ and $\text{Bdata}_{[7..0]}$ are transfered to such unit.

However, it is not sufficient to prove these nine $\mathcal{G}_i$. In order to complete the proof, we need to assure *liveness* and declare the order and dependences of the modules mentioned above. By means of $\mathcal{A}_j$ we prove that each module will eventually call another module (*i.e.*, the DFN model is *live*) and we guide the control flow according to some intermediate data values placed in p_{44} and p_{45}. Therefore, the correctness of this DFN model has been proven within 190.45 seconds –as opposed to the state explosion suffered if we were to apply the methodology without any modularity.

6 Conclusions

We have examined the way to break down a complex specification in compositional verification, such that the amount of memory resources used is reduced. For this, we have proposed an estimation method that tackles the modular unbalance of a structure and aims to reduce the complexity by careful selection of the way the system is partitioned. It should be noted that the method and conclusions showed in this paper are not limited to any model in particular and can be applied to many internal design representations that allows modular decomposition. However, the method is slightly restricted by the fact that systems with uneven communication needs (as opposed to the the pipeline example) or systems which include cycles in their behaviour may respond in a different way and, therefore, be optimised by a different partitioning scheme. Thus, although the proposed method might not be optimal, it provides a good indication to efficiently partition a structure in terms of verification time. In order to validate the estimation method, it has been tested by means of a real-life example, showing that it can be successfully applied to models of relatively large complexity.

References

[1] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman. *Digital Systems Testing and Testable Design*. IEEE Computer Society Press, MD 20910, USA, 1990.

[2] Jean-Raymond Abrial. *The B-Book*. Cambridge Press, 1996.

[3] ANSI/IEEE. *Information Processing Systems – Local Area Networks – Part 3: Carrier Sense Multiple Access with colision Detection (CSMA/CD) access method and physical Layer Specifications*. The IEEE, Inc., N.Y., October 1991.

[4] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. McKenzie. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer-Verlag, Berlin, Germany, 2001.

[5] Michael L. Bushnell and Vishwani D. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001.

[6] Cadence. The SMV Model Checker. http://www-cad.eecs.berkeley.edu/∼kenmcmil/smv/, 2001.

[7] *Proceedings of the* 7^{th}*International Conference on Computer Aided Verification*, volume 939 of *Lecture Notes in Computer Science*, Liège, Belgium, 3-5 July 1995. Springer-Verlag.

[8] Edmund M. Clarke, David E. Long, and Kenneth L. McMillan. Compositional Model Checking. In *Proceedings of the* 4^{th}*IEEE Symposium on Logic in Computer Science*, pages 353–362, 1989.

[9] CMU. The SMV System. http://www.cs.cmu.edu/∼modelcheck/smv.html, 1997.

[10] David L. Dill and Howard Wong-Toi. Verification of Real-Time Systems by Successive Over and Under Aproximation. In CAV [7].

Name	Property	Verification Time [sec]	BDD Nodes	Module
$\mathcal{G}_{\text{acc}}$	$\lvert\mu(p_{10})\rvert = 1 \implies \Diamond\lvert\mu(p_{21})\rvert = 1$	8.48 s	10033	$\mathcal{M}_1$
$\mathcal{G}_{\text{from}}$	$\lvert\mu(p_{10})\rvert = 1 \implies \Diamond\angle\mu(p_{12}) = \angle\mu(p_9)$	9.2 s	10256	$\mathcal{M}_1$
$\mathcal{G}_{\text{s1}}$	$\lvert\mu(p_{10})\rvert = 1 \implies \Diamond\angle\mu(p_{13}) = \angle\mu(p_{12})$	9.11 s	10494	$\mathcal{M}_1$
$\mathcal{G}_{\text{s2}}$	$\Diamond\angle\mu(p_{13}) = \angle\mu_0(p_{12}) + 2$	10.82 s	141160	$\mathcal{M}_5$
$\mathcal{G}_{\text{s3}}$	$\Diamond\angle\mu(p_{13}) = \angle\mu_0(p_{12}) + 4$	10.85 s	140948	$\mathcal{M}_5$
$\mathcal{G}_{\text{high}}$	$\Diamond\angle\mu(p_{34}) = \text{high}(\angle\mu(p_{14}))$	8.12 s	5704	$\mathcal{M}_3$
$\mathcal{G}_{\text{low}}$	$\Diamond\angle\mu(p_{34}) = \text{low}(\angle\mu(p_{14}))$	7.68 s	3268	$\mathcal{M}_4$
$\mathcal{G}_{\text{rel}}$	$\lvert\mu(p_8)\rvert = 1 \implies \Diamond\lvert\mu(p_{26})\rvert = 1$	8.14 s	7015	$\mathcal{M}_8$
$\mathcal{G}_{\text{fail}}$	$\lvert\mu(p_{36})\rvert = 1 \implies \Diamond\mathsf{sch} = 23$	8.24 s	9029	$\mathcal{M}_2$
$\mathcal{A}_{\text{start}}$	$\lvert\mu(p_{10})\rvert = 1 \implies \Diamond\lvert\mu(p_3)\rvert = 1$	8.5 s	10017	$\mathcal{M}_1$
$\mathcal{A}_{\text{dest1}}$	$\lvert\mu(p_3)\rvert = 1 \implies \Diamond\lvert\mu(p_4)\rvert = 1$	7.41 s	4012	$\mathcal{M}_3$
$\mathcal{A}_{\text{dest2}}$	$\lvert\mu(p_4)\rvert = 1 \implies \Diamond\lvert\mu(p_5)\rvert = 1$	7.01 s	2365	$\mathcal{M}_4$
$\mathcal{A}_{\text{len-c}}$	$\Diamond\lvert\mu(p_{43})\rvert = 1$	8.14 s	7628	$\mathcal{M}_5$
$\mathcal{A}_{\text{len-d}}$	$\Diamond\angle\mu(p_{32}) = \angle\mu_0(p_{14})$	9.53 s	69283	$\mathcal{M}_5$
$\mathcal{A}_{\text{len-i}}$	$\Diamond\angle\mu(p_{44}) = \text{high}(\angle\mu(p_{32}))$	8.37 s	5506	$\mathcal{M}_5$
$\mathcal{A}_{\text{len-n0}}$	$\mu(p_{44}) = 1 \cdot e^{\mathbf{i}\cdot\alpha}, \alpha \neq 0 \implies \Diamond\lvert\mu(p_6)\rvert = 1$	8.07 s	2629	$\mathcal{M}_5$
$\mathcal{A}_{\text{len-e0}}$	$\mu(p_{44}) = 1 \cdot e^{\mathbf{i}\cdot 0} \implies \Diamond\lvert\mu(p_8)\rvert = 1$	8.14 s	3760	$\mathcal{M}_5$
$\mathcal{A}_{\text{dt0}}$	$\Diamond\lvert\mu(p_{46})\rvert = 1$	8.19 s	10013	$\mathcal{M}_6$
$\mathcal{A}_{\text{dt1}}$	$\angle\mu_0(p_{44}) > \angle\mu_0(p_{45}) \implies \Diamond\lvert\mu(p_7)\rvert = 1$	8.23 s	9598	$\mathcal{M}_6$
$\mathcal{A}_{\text{dt2}}$	$\angle\mu_0(p_{44}) \leqslant \angle\mu_0(p_{45}) \implies \Diamond\lvert\mu(p_8)\rvert = 1$	8.29 s	9612	$\mathcal{M}_6$
$\mathcal{A}_{\text{d2t1}}$	$\angle\mu_0(p_{44}) > \angle\mu_0(p_{45}) \implies \Diamond\lvert\mu(p_6)\rvert = 1$	9.97 s	30585	$\mathcal{M}_7$
$\mathcal{A}_{\text{d2t2}}$	$\angle\mu_0(p_{44}) \leqslant \angle\mu_0(p_{45}) \implies \Diamond\lvert\mu(p_8)\rvert = 1$	9.66 s	30704	$\mathcal{M}_7$

Table 4: Ethernet LTL properties

[11] Orna Grumberg and David E. Long. Model Checking and Modular Verification. *ACM Transaction on Programming Languages and Systems*, 16(3):843–871, 1994.

[12] Rajesh K. Gupta and Giovanni De Micheli. System Synthesis via Hardware-Software Co-Design. Technical Report CSL-TR-92-548, Stanford University, Computer Systems Laboratory, 1992.

[13] G. H. Hardy and S. Ramanujan. Asymptotic Formulae in Combinatory Analysis. *Proceedings of the London Mathematical Society*, 17:75–115, 1918.

[14] David Harel, Orna Kupferman, and Moshe Y. Vardi. On the Complexity of Verifying Concurrent Systems. *Imperial College*, 173:143–161, 2002.

[15] Thomas A. Henzinger and Rajeev Alur. Local Liveness for Compositional Modeling of Fair Reactive Systems. In CAV [7], pages 166–179.

[16] Ranjit Jhala and Kenneth L. McMillan. Microarchitecture Verification by Compositional Model Checking. In *Proceedings of the CAV'01*, volume 2102 of *Lecture Notes in Computer Science*, pages 396–410, Paris, France, 18-22 July 2001. Springer-Verlag.

[17] Catholijn M. Jonker and Jan Treur. Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. *International Journal of Cooperative Information Systems*, 11(1-2):51–91, 2002.

[18] D. Kozen and J. Tiuryn. Logics of Programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 789–840. North Holland, Amsterdam, 1989.

[19] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. Module Checking. *Information and Computation*, 164(2):322–344, 2001.

[20] David E. Long. *Model Checking, Abstraction, and Compositional Verification*. Ph.D. Thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA 15213-3891, USA, July 1993.

[21] Kenneth L. McMillan. Verification of an Implementation of Tomasulo's Algorithm by Compositional Model Checking. In *Proceedings of the CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, Vancouver, BC, Canada, 28 June - 2 July 1998. Springer-Verlag.

[22] Kenneth L. McMillan. A Methodology for Hardware Verification using Compositional Model Checking. *Science of Computer Programming*, 37(1-3):279–309, 2000.

[23] Kenneth L. McMillan. Parameterized Verification of the FLASH Cache Coherence Protocol by Compositional Model Checking. In *Proceedings of the CHARME'01*, volume 2144 of *Lecture Notes in Computer Science*, Livingston, Scotland, UK, 4-7 September 2001. Springer-Verlag.

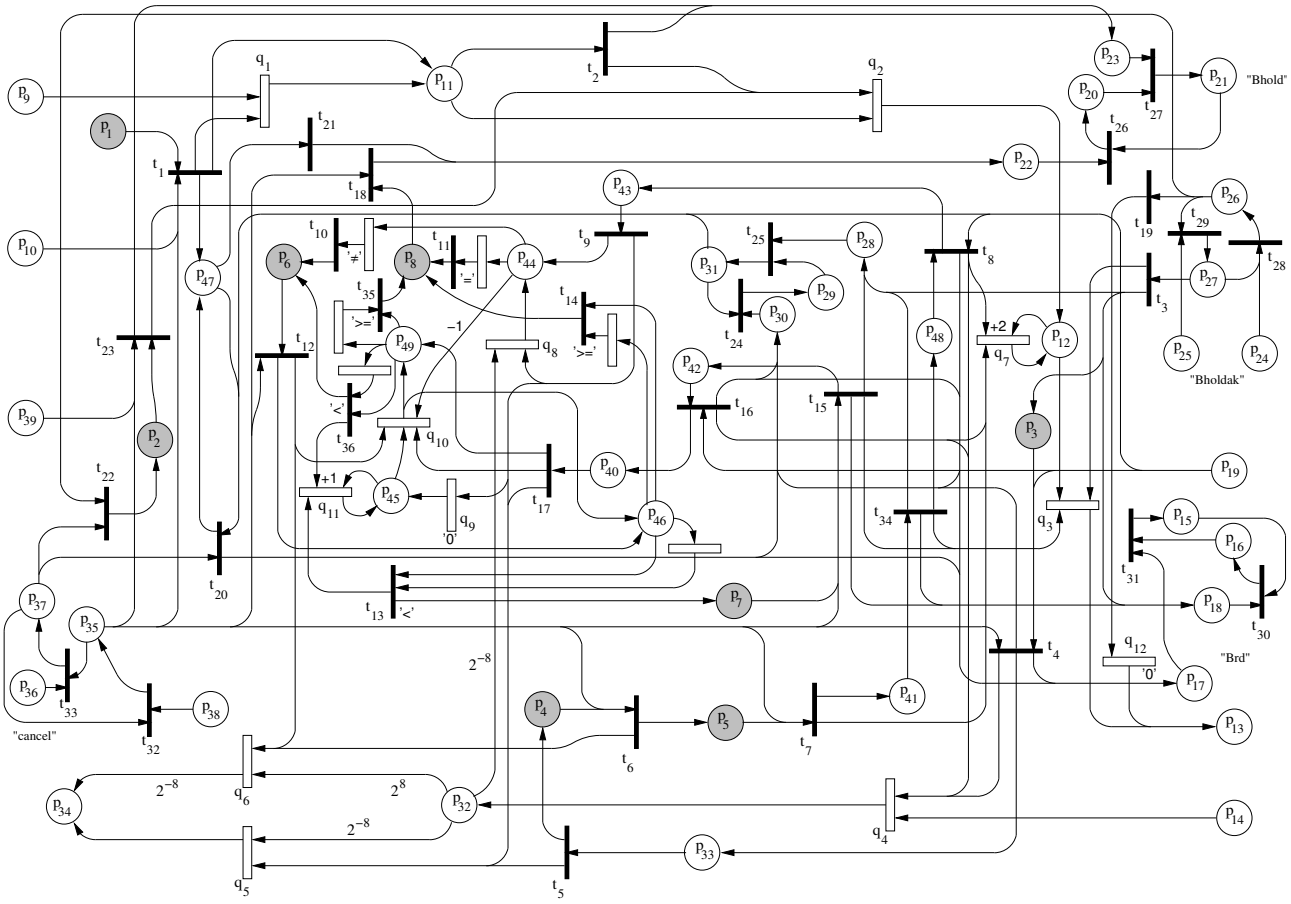

Figure 2: DFN model of the Ethernet coprocessor

[24] Jayadev Misra and K. Mani Chandy. Proofs of Networks of Processes. *IEEE Transaction on Software Engineering*, 7(4):417–426, July 1981.

[25] Vivek G. Naik and A. P. Sistla. Modeling and Verification of a Real Life Protocol Using Symbolic Model Checking. In *Proceedings of the* 6^{th}*International Conference on Computer Aided Verification (CAV)*, 1994.

[26] Sanjiv Narayan and Frank Vahid. Modeling with SpecCharts. Technical Report ICS-TR-90-20, University of California, Irvine, Department of Information and Computer Science, October 1992.

[27] Amir Pnueli. In Transition for Global to Modular Temporal Reasoning. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*, volume 13 of *NATO ASI series. Series F, Computer and System Sciences*. Springer-Verlag, 1984.

[28] Subir K. Roy, Hiroaki Iwashita, and Tsuneo Nakata. Formal Verification based on Assume and Guarantee Aproach - A Case Study. In *Proceedings of the Asia and South Pacific Design Automation Conf. (ASP-DAC)*, pages 77–80, Yokohama, Japan, 25-28 January 2000.

[29] Mauricio Varea. Mixed control/data-flow representation for modelling and verification of embedded systems. MPhil/PhD-Transfer Report, Department of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK, March 2002.

[30] Mauricio Varea and Bashir M. Al-Hashimi. Dual Transitions Petri Net based Modelling Technique for Embedded Systems Specification. In *Proceedings of the DATE'01*, pages 566–71, Munich, Germany, 13-16 March 2001. ACM/IEEE.

[31] Mauricio Varea, Bashir M. Al-Hashimi, Luis A. Cortés, Petru Eles, and Zebo Peng. Symbolic Model Checking of Dual Transition Petri Nets. In *Proceedings of the CODES'02*, pages 43–48, Colorado, USA, 6-8 May 2002.

[32] H. B. Weinberg and L. D. Zuck. Timed Ethernet: Real-time Formal Specification of Ethernet. In *Proceedings of the CONCUR'92*, volume 630 of *Lecture Notes in Computer Science*. Springer-Verlag, August 1992.