Visualisation of Partial Order Models in VLSI Design Flow

A. Bystrov, M. Koutny and A. Yakovlev

Technical Report Series

October, 2001
Contact:
A.Bystrov@ncl.ac.uk
Maciej.Koutny@ncl.ac.uk
Alex.Yakovlev@ncl.ac.uk

Supported by EPSRC grant GR/M94366, project MOVIE

Copyright©2001 University of Newcastle upon Tyne
Published by the University of Newcastle upon Tyne,
Department of Computing Science, Clarendon Tower, Clarendon Road,
Newcastle upon Tyne, NE1 7RU, UK
Visualisation of Partial Order Models in VLSI Design Flow

A. Bystrov, M. Koutny, A. Yakovlev

15th October 2001

Abstract

This paper presents a new method, algorithms and tool for the visualisation of a finite complete prefix of a Petri net or a signal transition graph. A transformation is defined that converts such a prefix into a two-level model. At the top level, it has a finite state machine, describing modes of operation and transitions between them. At the low level, there are marked graphs, which can be drawn as waveforms, embedded into the top level nodes. The models of both levels are abstractions traditionally used by electronics engineers. The resultant model is completed trace equivalent to the original prefix. Moreover, the branching structure of the latter is preserved as much as possible.

1 Introduction

The growing integration of modern VLSI circuits draws attention to the problems of power consumption, electro-magnetic compatibility, clock skew, robustness and scalability. This results in the increased interest in asynchronous compatibility as they compare favourably to synchronous circuits w.r.t. the above criteria. Unfortunately, the design effort required to implement an asynchronous circuit is high. This is partially due to the lack of efficient visualisation methods and tools to facilitate understanding of models used at different levels of asynchronous circuit development.

In this paper, 1-safe Petri nets (PN) are taken as system specification. System behaviour is densely encoded in a PN. Its extraction is a difficult task for human perception. Since system behaviour is densely encoded in a PN, and so its extraction is a difficult task for human perception, one needs to play the ‘token game’ on a PN to understand its behaviour and to obtain the fundamental relations of concurrency, conflict and precedence. PN and signal transition graph (STG) unfoldings are more convenient for developing such an understanding due to their acyclicity and branching structure. The algorithms of McMillan [7] and Esparza [4] derive a finite complete prefix (FCP), which is a part of the unfolding that contains all reachable markings and relations between them. An FCP is typically larger than the original PN or STG, but significantly smaller than the state space of a system (it is shown not to exceed its size [4]). An early attempt to visualise ‘causality graphs’, which are intuitively similar to PN unfoldings, was made in [12].

The visualisation methods of system specification stage are the main focus of this paper. Figure 1 shows the iterative process of specification refinement. Many aspects of system verification (e.g. PN safety, signal consistency, uniqueness of state coding, etc.) and some of specification correction aspects (e.g. signal insertion to correct state coding problems) can be handled automatically (see, e.g. [3, 10]). However, the convergence of the ‘verify-correct’ cycle is hard to guarantee. Moreover, specification correction algorithms use heuristics and often produce a sub-optimal solution that is inferior w.r.t. the manually introduced changes. This motivates the design of visualisation tools that allow a designer to correct a specification either by hand, or by guiding
a correction tool. Model decomposition and its structural simplification constitutes an important visualisation aspect. It allows the designer to study a particular property of a system, removing irrelevant information and preserving only the needed equivalence; for example, the branching time semantics can be largely ignored in the study of traces [5].

Figure 1: The role of visualisation in the process of constructing a correct specification

Figure 2 shows the place of our visualisation method and tools within the overall asynchronous design toolkit. There are several graphical entry tools for PN and STG graphs (e.g.[1, 2]), as well as several universal graph drawing tools that can be used to visualise the results of specification analysis (e.g.[8, 3]). However, the layout they produce is often virtually unreadable because they do not consider the semantics of a PN, FCP or a SG, in particular, those aspects which are related to the interplay between conflict and concurrency.

They also display entire models rather than their relevant fragments. Our approach to the visualisation, on the other hand, is focused on the design of problem-oriented tools. We have developed the ‘sgPad’ tool, which draws SGs in the form of hyper-cubes in the style of [11], and the ‘unfPad’ tool for FCP visualisation. The decomposition method used in ‘unfPad’ is described in this paper. The output of both tools can be static (a printable postscript format) or dynamic (animated). The two aspects of animation implemented in ‘unfPad’ are the movement through the hierarchy calculated by the same tool, and the simple ‘token game’ which remains an important instrument for interactive model study.

An FCP of a complex system encapsulates three types of relationships between its nodes, viz. causality, concurrency and conflict. Whilst the models based on causality and concurrency, such as timing diagrams, or on causality and conflict, such as finite state machines (FSM), are well known to electronics designers, the raw FCP model might be too complex to manipulate directly. Usually, the designers would create a top-level model of the system, as an FSM defining the operational modes and interaction between them, and the low-level model would be a timing diagram against which the waveforms of signals are then compared. In this paper, a method of representing an
arbitrary FCP as such a two-level model is described.

The desired form of FCP representation is shown in Figure 3. The top-level model is an FSM in the form of a tree. It contains macro-states $A \ldots E$ corresponding to the modes of operation and arcs representing choice options. Internally, a macro-state is associated with a concurrent system execution in which all the non-deterministic choices have been resolved. As such a transition between two macro-states can be associated with the state the arc points to. Internally a macro-state is represented as a concurrent system without choice elements. Figure 3(b) shows an example of timing diagram embedded into a macro-state.

Figure 3: Desired model representation
2 Basic notions

A PN (a net system) is a tuple $\Sigma = (P,T,F,M_0)$ comprising finite disjoint sets of places $P$ and transitions $T$, flow relation $F \subseteq (P \times T) \cup (T \times P)$ and initial marking $M_0$. There is an arc between $x$ and $y$ iff $(x,y) \in F$. The preset of a node $x$ is defined as $\bullet x = \{ y \mid (y,x) \in F \}$, and the postset as $\bullet^* x = \{ y \mid (x,y) \in F \}$. A marking is a mapping $M : P \rightarrow N = \{0,1,\ldots \}$. It is assumed that $\bullet t \neq \emptyset \neq \bullet^* t$ for every transition $t \in T$. The evolution of a PN from the initial marking $M_0$ to a marking $M_n$ by executing transitions results in a firing sequence $\sigma = t_1t_2 \ldots t_n$, where $t_i$ are such that $M_i = t_{i+1}$ for $i = 0, \ldots , n-1$. Moreover, $M_n$ is called a reachable marking.

A Signal Transition Graph (STG) is a PN whose transitions are labelled by signal events, i.e. $STG = (P,T,F,M_0,\lambda)$, where $\lambda : T \rightarrow A \times \{+,-\}$ is a labelling function and $A$ is a set of signals. Usually, the labelling assigns the same label to several transitions. Another specific property of an STG is signal consistency, i.e. when the net is executed no signal can have two transitions of the same polarity without having a transition of the opposite polarity between them. The latter property is difficult to assert without proper visualisation methods and tools.

An occurrence net (ON) is a triple $O = (B,E,G)$, where $B$ are conditions (instances of PN’s places), $E$ are events (instances of PN’s transitions) and $G$ is the flow relation (similar to the F of a PN). The properties required of $O$ are that: (a) $|\bullet b| \leq 1$ for every $b \in B$; (b) $O$ is acyclic; (c) $O$ is finitely preceded; and (d) no element is in conflict with itself (see below).

The fundamental relations defined by an ON are precedence, conflict and concurrency. The flow relation $G$ defines the precedence $\prec$ as an irreflexive transitive closure of $F$. Two nodes are in conflict, $y \not\prec y'$, if there are distinct events $e, e' \in E$ such that $\bullet e \cap \bullet e' \neq \emptyset$ and $(e,y)$ and $(e',y')$ are in the reflexive transitive closure of $G$. Two nodes are concurrent $y \parallel y'$ if neither $y \not\prec y'$ nor $y' \not\prec y$.

A branching process (BP) generated by a PN or STG $\beta = (B,E,G,\pi)$ is an ON extended by a labelling function $\pi$. The labelling function $\pi$ reflects the mapping of the places and transitions of the original net system into conditions and events of the branching process. The following properties are satisfied:

1. $\pi (B) \subseteq P$ and $\pi (E) \subseteq T$.

2. For every $e \in E$, the restriction of $\pi$ to $\bullet e$ is a bijection between $\bullet e$ and $\pi (e)$, and similarly for $\bullet^* e$ and $\pi (\bullet^* e)$ ($\pi$ preserves the environment of transitions).

3. The restriction of $\pi$ to $\min (O) = \{ b \in B \mid \bullet b = \emptyset \}$ is a bijection between $\min (O)$ and $M_0$ (BP starts at the initial marking).

4. For all $e_1, e_2 \in E$, if $\bullet e_1 = \bullet e_2$ and $\pi (e_1) = \pi (e_2)$ then $e_1 = e_2$ (BP is compact).

A prefix of a BP $\beta$ is any branching process $\beta'$ satisfying (a) $\min (\beta') = \min (\beta')$; (b) if $b$ is a condition in $\beta'$, then $\bullet b$ in $\beta$ belongs to $\beta'$; (c) if $e$ is an event in $\beta'$, then $\bullet e \cap \bullet^* e$ in $\beta$ belong to $\beta'$; and (d) $\pi'$ is the restriction of $\pi$ to $\beta'$.

A configuration $C$ of a branching process is a set of events satisfying: (a) if $e \in C$ and $e' \prec e$ then $e' \in C$ (causal closure); and (b) for all $e, e' \in C, \neg(e \# e')$ (no conflicts).

A cut is a maximal w.r.t. set inclusion set of pairwise concurrent conditions.

A BP is complete if for every reachable marking $M$ there is a configuration $C$ satisfying: (a) $Mark (C) = M$; and (b) for every $t$ enabled under $M$ there is a configuration $C \cup \{ e \}$ such that $e \notin C$ and $e$ is labelled by $t$. 

Technical Report, project MOVIE, Oct. 2001, CS Dept., University of Newcastle upon Tyne 4
A prefix that is a complete and contains a finite number of nodes is called a finite complete prefix (FCP) [7, 4].

A casually closed and conflict-free set of events of a BP is a configuration. A local configuration (LC) of an event is the minimal configuration that contains this event.

Other definitions needed in this paper are those of a marked graph (MG) and a finite state machine (FSM). An MG is an ON without choice, i.e. any place has at most one outgoing arc. An FSM is a net system without concurrency. If an FCP is FSM then any two nodes (say \( y \) and \( y' \)) are either in the relation of precedence (\( y \preceq y' \) or \( y' \preceq y \)), or in conflict (\( y\#y' \)).

For analysis of semantic equivalence we will need the following. A complete trace of a finite labelled ON (not necessarily satisfying the restrictions of BP) is a maximal sequence of events belonging to a configuration and compliant with the precedence relation induced by the ON. A complete run of a finite labelled ON is a maximal configuration. Every complete run captures a set of complete traces which are equivalent up to permutation of mutually concurrent events. In this way, runs are equivalent to complete extended (Mazurkiewicz) traces [6]. Two finite labelled ONs are completed trace (run) equivalent if they produce the same sets of complete traces (runs). Such equivalence falls into the category of linear-time semantics, which is weaker than branching-time equivalence, often referred to as bisimulation [5].

3 Decomposition by nodal points

A naive attempt to identify MGs in an FCP is shown in Figure 4(a). The shaded areas are MGs or ‘bands’ labelled as \( B1 \ldots B11 \).

![Figure 4: Separating choice from concurrency](image)

The first disadvantage of this representation is that the bands are not self-contained, i.e. a global state of the FCP may include marked conditions that belong to different bands (e.g. \( c5 \) and
c8). Including c5 in both B2 and B4 results in an overlapping (shown as a dashed line), which is inconvenient for model manipulation. The second disadvantage is that we cannot interpret the top level (a graph where bands are taken for nodes) as a FSM.

A better decomposition is based on the notion of non-concurrent nodal points (NNP). For such we take those events that are not concurrent to any other node. They form the ‘skeleton’ of FSM. The algorithm defining bands by NNP is a simple graph traversal that starts from one NNP and never crosses another NNP. Being applied to all NNPs it generates a set of bands as shown in Figure 4(b). A band has only one input event, which results in a tree structure of the band graph.

It is easy to see that our NNP based decomposition is close to the idea of nodal states [9]. If the postset of a nodal state is a single event, then it is a NNP. If the postset comprises several events, then this can be reduced to the previous case by inserting a single ‘dummy’ event between the cut representing the nodal state and the postset of that state.

The resulting bands are self-contained, i.e. any cut is completely contained in a band. Bands form macro-states of a FSM, which is a desired property of the top-level model representation. The model with bands is bisimilar to the original as it does not change the FCP. A disadvantage of this model is that the content of a band is not necessarily a MG, e.g. the bands B4 and B5 in Figure 4(b) contain choice. The latter problem is addressed in the following section.

4 Decomposition by choice recalculation

In order to convert bands into MGs we propose a technique of choice recalculation. It transforms the original FCP into an equivalent labelled occurrence net in which no choice condition is concurrent to any other node. The equivalence is such that it preserves all reachable markings, firing sequences and complete traces. At some stage the bisimulation may be sacrificed, which is a price to pay for a simpler structure. However, the following two properties are still satisfied (these are necessary and sufficient conditions for the trace equivalence):

Equiv. 1: The set of traces of the original FCP are covered.

Equiv. 2: No new complete traces are created.

The application of this technique to the choice condition c4 is shown in Figure 5. The idea is to push the choice backwards, in the direction opposite to the token flow, to the location that is not concurrent to any other node. Then the bands are formed as in the case of the nodal point decomposition.

The procedure of recalculating a choice condition, denoted later as c*, works in three stages. The first stage is the search for the nearest event which is not concurrent to any FCP node and belongs to the local configuration of the c* condition. Such an event is called the nearest non-concurrent event (NNE) to c*; it always exists and is unique. The second stage creates multiple copies of the subgraph following the NNE and links these copies to the pre-conditions of the NEE; the number of copies is equal to the number of choice options (events that are in the postset of c*). The third stage, the restrict operation, modifies the subgraphs for these copies so that every subgraph instance contains only a single choice option. These stages are combined in the overall choice recalculation procedure so that the equivalence requirements Equiv. 1 and Equiv. 2 are satisfied.

NNE Search. This stage is performed by Algorithm 1. It starts at the c* condition, and then recursively traverses the FCP backwards. In doing so, the algorithm ignores all conditions and
those events which do not create new concurrency, i.e. events with a single outgoing arc. For other
events it checks if they are concurrent to any node in the graph and continues traversal if they are.
Otherwise, the traversal stops and the current event is the NNE of $c^*$. An interesting feature of
this algorithm is that it only needs to traverse through a single preset element of an event, which
follows from standard properties of configurations.

**Subgraph copying.** This stage is performed by a recursive traversal of FCP nodes, starting
from the NNE, found in the previous stage. At every step the algorithm copies a node and calls
itself for every postset element of that node. Flags are used to avoid multiple passes through the
nodes following the events with multiple incoming arcs. A new instance number is given to every
copied node and the original instance number is saved in an additional field of the node object.
Thus a new level of labelling is introduced, as illustrated, e.g., in Figures 11-12. The algorithm
produces an identical copy of the subgraph which follows the given NNE. The new subgraph is
then connected to the preset of the NNE by arcs as shown in Figure 6. If the NNE has more than
one element in its preset, then the algorithm generates extended choice as shown in Figure 6(b).

**Restrict.** The algorithm for this stage, Algorithm 2, is applied separately to each subgraph
copy created in the previous stage, which is identified by means of special flags. It selects one choice
option for the $c^*$ condition and deletes the remaining options. The main task of the algorithm is to
prune choice options of the original $c^*$ condition in the given copy in such a way that the Equiv. 2
property is satisfied.

Pruning the subgraph is relatively easy when the $c^*$ condition is either the only choice in
the subgraph, or in the case of multiple choices, these choices are independent (i.e., executing any
option of one choice does not disable any option of the other choice). Then it is sufficient to remove
the successors of the deleted choice options. In other cases, in order to guarantee the Equiv. 2
Algorithm 1 Search for NNE.

```c
node find_NNE( node nl )
{
    node n2, n3;
    if( nl->type == condition )
    /*conditions don't create or reduce concurrency, skip them*/
    {
        n2=find_NNE(nl->preset);
        /*preset of a condition always contains only 1 element*/
        return n2;
    }
    if( nl->postset.size() < 2 )
    /*an event that doesn't generate concurrency, skip it*/
    {
        n2=find_NNE(nl->postset.begin());
        /*traversing just 1 preset element is sufficient*/
        return n2;
    }
    select_local_configuration(nl);
    select_successors(nl); /*incremental selection, no flag resetting*/
    select_conflict(nl);
    /*unselected nodes are concurrent to nl*/
    for( (graph, n3)
    if( n3 is concurrent to nl ) /* continue traversal */
    {
        n2=find_NNE(nl->postset.begin());
        return n2;
    }
    return nl; /*NNE found, recursively return the NNE */
}
```

Algorithm 2 'Restrict' operation.

```c
void restrict(condition choice, event option)
{
    operate_on_the_selected_subgraph();
    select_conflict( option );
    for( (choice, postset, el)
    if( el != option ) /* all options except the given one */
    {
        select_successors(el); /* select union of successors */
        select_local_configuration(el); /* select union of LCs */
        remove_node( el );
    }
    for( (subgraph, node)
    if( node.flag_successor == true)
    remove_node( node );
    if( node.flag_conflict == true
        && node.flag_local_configuration == true)
    remove_node( node );
}
```
property, some of the predecessors of the deleted choice option (together with their successors) must also be removed. This is illustrated in Figure 7(a), which shows an example of the so called 'controlled choice' (condition e8) having two options controlled by another choice (condition c2). If e* = e8, then removing one of its choice options, e1, and its successors (as shown by dotted lines) results in a net that produces a new complete trace, composed of events e1, e2, e3, e5 and e6. To prevent this, the algorithm also deletes two predecessors of e4, viz. c3 and e2, as shown by dashed lines. Note that the algorithm actually finds the primary choice and removes its options, the options of the controlled choice are removed automatically. In a more complex example, shown in Figure 7(b), the process of pruning involves also successors of the nodes deleted earlier, to guarantee the Equiv. 2 property. Other combinations of choices, such as extended free choice (as in Figure 6(b)) and the so called 'confusion' (when several choices are mutually dependent on each other) are also handled by the algorithm.

![Diagram](image)

**Figure 6: ‘Copy’ operation**

**Figure 7: Controlled choice**

**Overall procedure.** The overall procedure of choice recalculation, implemented in Algorithm 3, processes one choice and one option at a time. It applies the operations of ‘copy’ and ‘restrict’ so that the Equiv. 1 property is satisfied. Equiv. 2 is guaranteed by the ‘copy’ and ‘restrict’ algorithms. The algorithm scans the node set, finds choice conditions, checks if the choice condition is concurrent to any other condition in the net and, if it is, performs the recalculation of
choice in the new location. The case of extended choice is detected at this stage and the conditions that are the members of the extended choice are not counted in the concurrency check.

Algorithm 3 Choice recalculation

```java
recalculate_choice()
{
    for all (graph, node)
        if (node.type == condition && node.postset.size() > 1) /* choice */
            if (is_concurrent(node)) /* ignore extended choice */
                recalculate_choice_1(node);
}

recalculate_choice_1(node choice)
{
    e1 = find_NNE(choice);
    for all (choice.postset, e2)
        /* recalculate option e2 */
        /* copy operation*/
        copy_subgraph(e1);
        /* restrict operation*/
        c1 = find_image(choice); /* image in the new subgraph*/
        e3 = find_image_option(choice, e2, c1); /* e3 in c1 is an image of e2 */
        select_conflict(e3);
        for all (c1.postset, e4)
            if (e4 != e3)
                select_local_configuration(e4);
        for all (graph, n1) /* remove */
            if (n1.flag_local_configuration && n1.flag_conflict)
                remove_with_successors(n1);
}
```

After a candidate for recalculation is found, the algorithm looks for the NNE. Then for every option of this choice the following is done: (i) a copy of the subgraph is created; and (ii) the ‘restrict’ operation is applied to remove all options except the selected one. Eventually, after all options have been processed, the original subgraph is deleted.

The application of the above algorithm to the right part of FCP in Figure 4 is shown in Figure 8. Two concurrent choice conditions, c6 and c7 create four choice combinations. The algorithm is applied first to any of these conditions, c6 in this example, producing two instances of c6 and two of c7. The second run of the algorithm recalculates c7' choice (the first instance of c7) and the third run unfolds c7'' (the second c7 instance).

![Figure 8: Example of choice recalculation](image_url)

Technical Report, project MOVIE, Oct. 2001, CS Dept., University of Newcastle upon Tyne
The resultant graph is larger than the FCP. In the presence of multiple concurrent free choice places its size becomes exponential. However, there is no need to store all the bands. They can be generated dynamically as the designer studies them. The thread of bands starting from the initial event and ending at any band without successors forms a completed MG, which is a subset of the original FCP. This guarantees that any band and any fully ordered set of bands do not exceed the size of the original FCP.

The bands form a choice graph as shown in Figure 9. The choice is now attributed to different conditions. This can be called the 'hypothesis' semantics (a version of 'decorated trace semantics' in [5]). In this semantics the choice is known a priori, i.e. at the condition c3 instead of c6 and c7. The moment at which the choice becomes known can be reflected by means of additional labelling of the nodes, which, however, does not affect the traces generated by the model. This means that the resultant graph is trace-equivalent to the original, though it is not bisimilar. If the choice is fixed in advance then the maximal MG derived from this model exactly matches the MG derived from the original FCP. This is true for any choice combination (for all possible modes of operation), which guarantees that no deadlocks are lost and no new deadlocks are created.

A few more examples give the flavour of the described method. The FCP of VME bus controller in Figure 10 contains a combination of free choice and controlled choice. After choice recalculation two bands are formed, one for the read mode, another for write mode, which is convenient for model interpretation. The model semantics, however, was shifted in the linear time - branching time spectrum [5] towards the linear time.

A model of RGD arbiter in Figure 11 also became completely converted into the linear time semantics due to the inherently concurrent specification. Its left part corresponds to a scenario of the first request winning the arbitration, in the right part the second request wins. The actual choice has been moved from condition (2) to the initial condition (0).

The model of read interface shown in Figure 12, however, partially retains branching structure after choice recalculation. It is clear that the model in Figure 12(b) has a simpler branching structure than the model in Figure 12(a).
5 Conclusions

The paper is a novel attempt to visualise the behaviour of asynchronous systems in a graphical capture that takes into account the semantics of the model represented by the graph. While
similar attempts exist in the state-based representation domain [11], we apply this to a partial order model, provided by the Petri net unfolding. A method of decomposition of a finite complete prefix of the unfolding has been described above and implemented in a tool ‘unfPad’. This method presents an FCP as a two-level model. The top level of this model is a finite state machine, which represents modes of system operation and transitions between them. At the low level, there are marked graphs, which can be drawn as waveforms, embedded into the top level nodes. The models of both levels are abstractions traditionally used by electronics engineers. The resultant model is completed trace equivalent to the original prefix. Moreover, the branching structure of the latter is preserved as much as possible (cf. ‘decorated trace semantics’ in [5]).
References


