School of Electrical, Electronic & Computer Engineering

Desynchronisation Technique using Petri Nets

Sohini Dasgupta, Alex Yakovlev, Victor Khomenko

Technical Report Series

NCL-EECE-MSD-TR-2007-124

November 2007

Contact:

Sohini.Dasgupta@ncl.ac.uk

Alex.Yakovlev@ncl.ac.uk

Victor.Khomenko@ncl.ac.uk

Supported by EPSRC grant GR/S12036

NCL-EECE-MSD-TR-2007-124

School of Electrical, Electronic & Computer Engineering,

Merz Court,

University of Newcastle upon Tyne,

Newcastle upon Tyne, NE1 7RU, UK

`http://async.org.uk/`

Desynchronisation Technique using Petri Nets

Sohini Dasgupta, Alex Yakovlev, Victor Khomenko

November 2007

Abstract

In this report we consider the problem of desynchronising modular synchronous specifications for their realisation into GALS architectures and obtaining simple wrappers that are efficiently synthesisable using existing synthesis tools. The systems are modeled using Petri nets (PN) and the desynchronisation technique is based on the theory of PN Localities. The firing semantics of a globally synchronous system is characterised by maximal firing of input and output transitions. The compartmentisation of a synchronous system is achieved by unbundling the input transitions and allowing the output transitions to fire maximally, in order to enable asynchronous communication in a distributed environment. Our model preserves the two essential correctness properties, namely, semantics preservation and deadlock prevention, during the shift from maximal firing semantics followed by synchronous systems to standard semantics for input transitions and maximal semantics for output transitions followed by GALS architectures. The compartmentisation process is followed by extra signalisation to aid the formation of wrappers and replacement of the global clock by local clocks in each compartment. The wrappers, thus formed, can be efficiently synthesised using existing PN synthesis tools.

1 Introduction

This report introduces a new methodology for the desynchronisation of synchronous systems into globally asynchronous and locally synchronous (GALS) architectures.

In [2], Transition Systems (TS) were used as the specification model to describe the synchronous systems for the purpose of desynchronising the system into GALS architecture. Our previous work exemplified in [4], that the models obtained for each synchronous module can be very large and complex due to the weak handling of concurrency posed by the previous desynchronisation methodology. Concurrency is a prerequisite for the specification of synchronous systems which are required to be equipped to handle asynchronous communication for their GALS deployment. Moreover, these models are translated into Petri nets in order to use existing asynchronous tools for logic synthesis.

Therefore, the complexity of the transition system, obtained from the previous methodology in [4], and the computational complexity of the PN synthesis of these models, specifically the complex translation of the large TSs into PNs using theory of Regions [3][1], form the main motivations for this work. The new technique uses PN as the specification model whose efficient concurrency handling technique makes it one of the most viable models to describe systems for desynchronisation. Moreover, the theory behind the new technique uses the concept of *Localities* which helps in describing the distribution of a synchronous system over asynchronous architectures owing to its strong structural and functional correspondence with GALS architectures.

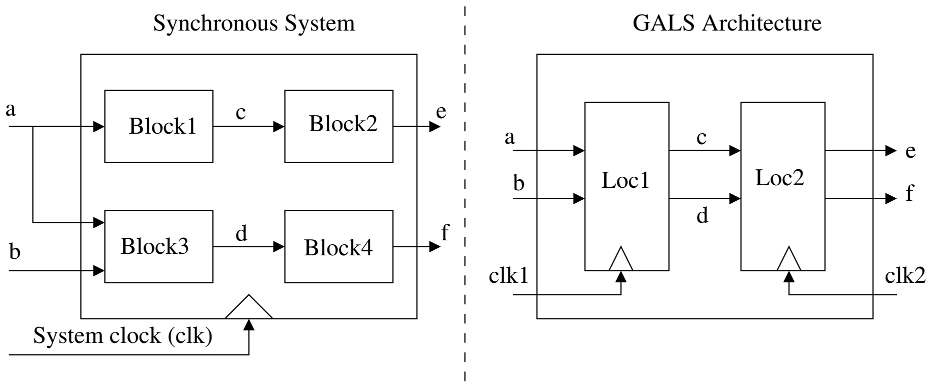


Figure 1: Synchronous system transformation into distributed architecture

A synchronous system consists of components which are associated with a set of input and output signals. Such a system is depicted in Figure 1. Each of these components are governed by activities like sending and receiving control signals as well as exchanging data signals across different components. The actions performed by all the components are controlled by a single global clock.

The theory of *localities* introduce the idea of localising these above mentioned components and hence their associated actions into individual blocks. These blocks are incorporated with some additional ordering constraints on their input and output signals. These constraints enable the individual blocks or localities to behave like independent synchronous systems. Therefore, the global clock can can be eliminated and each locality can be employed with local clocks which govern the actions assigned to them. Such a transformed system is depicted in Figure 1. As is evident from the figure, more than one component can be mapped onto each locality depending on some rules and optimisation criteria, discussed later. These individual localities created would then communicate with each other asynchronously due to the absence of a clock signal in between the localities owing to the removal of the global clock and application of the local clocks.. The technique to obtain a distributed architecture from a globally synchronous system must satisfy two essential correctness properties, namely,

- *semantics preservation of the original synchronous system*: During the execution of each synchronous sequence, components of the synchronous system compute events for the output signals based on the internal signals and the values of the input signals. Within each unit of time, the system is transformed by a maximally concurrent execution of input and output signals. The deployment of such a system into GALS architecture entails unbundling of input signals to aid out-of-order reception of these signals in an asynchronous environment. Therefore, this transformation to form a distributed architecture should preserve the semantics of the original system.

- *prevention of deadlocks*: When the synchronous system is transformed into a GALS architecture, the input transitions that were bound in the original system are unbundled, as previously discussed. This out-of-order reception of signals should not cause the system to enter into a deadlock state. Therefore, there should be additional constraints in the transformed model to avoid such occurrences.

Both the properties have been dealt with in Section 6.

2 Preliminaries

This work uses Petri net models to describe the synchronous systems. This is because all the components and actions carried out by synchronous systems can be directly mapped onto the different elements of a Petri net. For instance, synchronous events are represented on the *transitions* and the trigger conditions are denoted on the *places.* In order to show that a trigger condition is true, the place is equipped with a token. To make a synchronous component transit from one configuration to another is denoted by the firing of transition/transitions. Therefore, PN models are very expressive in describing a synchronous system. A detailed description of such models is presented in Section 4.

2.1 Petri nets

We recall some basic notations concerning Petri nets (see Chapter , Section). A Petri net is a model used to represent systems with concurrency. It is a quadruple $PN = \{P, T, F, \mu_0\}$, where P is a set of *places*, T is a set of *transitions*, F is an *arc* denoting the flow relation $F \subseteq \{(P \times T) \cup (T \times P)\}$ and μ_0 is the *initial marking*. A labelled PN is a PN with a labelling function $L : T \to A$ associating each transition of the net with a name. A labelled Petri net can have a combination of implicit places, where the input and output transitions are named using symbols from the alphabets, connected by arcs and transitions which are labelled with signal transitions $(a+, a-)$ or events (a_req, a_ack).

There exists an arc from $x \in P \cup T$ to $y \in P \cup T$ iff $(x, y) \in F$. The *preset* of a node $x \in P \cup T$ is defined as $\bullet x = \{y \mid (y, x) \in F\}$ and the *postset* as $x\bullet = \{y \mid (x, y) \in F\}$. A *marking* is a mapping $\mu : P \to N$ denoting the number of tokens in each place, $N = \{1, 0\}$ for $1-$*safe* PNs. A transition $t \in T$ is enabled at a marking μ, denoted by $\mu[t >$, if for every $p \in \bullet t$, $\mu(p) > 0$. For a $1-$*safe* PN, the firing of the transition t modifies marking by consuming one token from each of the predecessor places and producing one token to each of the successor places. A marking μ' is reachable from marking μ if there exists a *firing sequence* $\sigma = t_0...t_n$ that transforms μ to $\mu'$and is denoted by $\mu[\sigma > \mu'$. For any $a \in A$, by $\mu[a >$ (or, $\mu[a > \mu'$), it is meant that $\mu[t >$ (or, $\mu[t > \mu'$) for some t with $L(t) = a$.

Given a Petri net N, the *pre-* and *post-multiset* of a transition t are respectively the multiset $pre_N(t)$ and the multiset $post_N(t)$, such that for all $p \in P$, $|p|_{pre_N(t)} = F(p,t)$ and $|p|_{post_N(t)} = F(t,p)$, where $|p|$ denotes the number of tokens present in the place p. Since, all the systems defined in this work are *safe*, $|p| = 1$.

Definition 1. Step

A step is a multiset of transitions $U : T \rightarrow N$, where N is a set of natural numbers.

The steps can be executed in various modes depending on the system that the PN models describe, discussed in Sections 4 and 5.

For a PN to be synthesisable, it is required to satisfy some *Behavioural* and *Structural* properties. We recall an important structural property, namely, *Persistency* because it plays an integral role in our desynchronisation methodology.

Definition 2. Persistency

A Petri net (N, μ_0) is persistent if for any two different transitions t_1, t_2 of N and any reachable marking μ, if t_1 and t_2 are enabled at μ, then the occurrence of one cannot disable the other.

3 Motivation for using Localities

Our initial model starts with the description of a globally synchronous system. In such a system, execution and communication progresses along a sequence of events which are tagged by a global logical clock, i.e., they are active only at certain logical instants. During the execution of such sequences, each of the synchronous system components compute events for the output signals based on the internal signals and the values of the input signals. This global clock paradigm is associated with max firing semantics and transition binding. In order to exemplify the the different firing semantics, a simple example is considered in Figure 2.

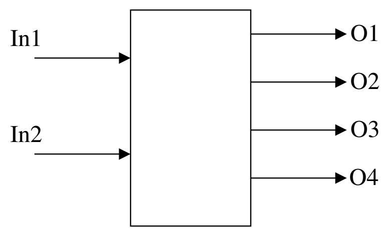


Figure 2: Simple synchronous block

As stated earlier, Petri net models are used to demonstrate the globally synchronous system. In this model the synchronous events are represented on the *transitions* and the trigger conditions are denoted on the *places*. In our PN model of the synchronous system, we presume that every event on the transition are implicitly tagged by the global clock and hence, for simplicity, we do not show the actual clock transition on this model. In such a net there are disjoint sets of inputs I and outputs O and a function l which maps the transitions of the Petri net to the set $I \cup O \cup \{t_{int}\}$, where $t_{int} \notin I \cup O$ is a silent event not observable by the environment. Let the inputs $I1$ and $I2$ be concurrent to each other. Let Figure 3 denote the Petri net representation of the input output dependencies of the system which is shown in Figure 2.

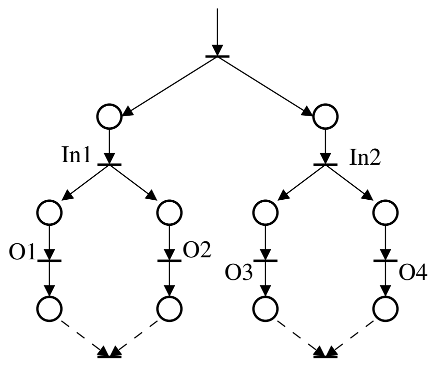
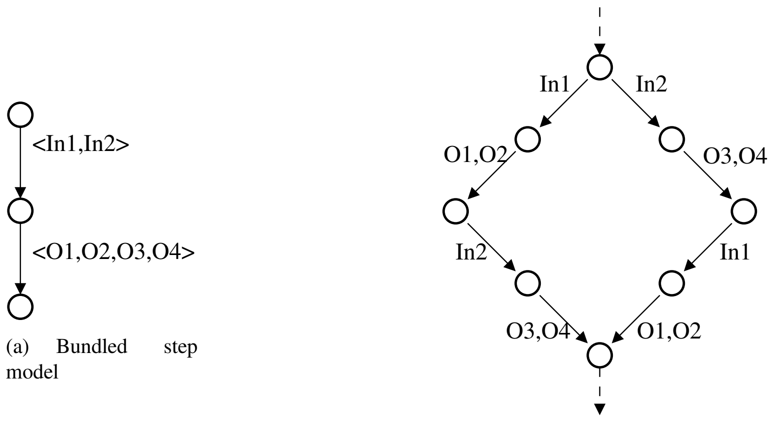


Figure 3: PN model of the synchronous block

As mentioned before, a globally synchronous paradigm is associated with maximal firing semantics. A state graph depicting such a semantic is presented in Figure 4(a). In order to desynchronise the synchronous system into GALS architecture, the input steps are required to unbundled to enable out of order communication. In a synchronous system when two inputs are unbundled and received at deterministic instances of time, Figure 4(b) is obtained.



(a) Bundled step model

(b) unundled input ordered state machine

Figure 4: System model

If a system is globally clocked, the inputs, outputs and the internal signals can be scheduled to fire in persistent (Definition 2) steps since they can be made to arrive at known instances of time.

Such a schedule cannot be maintained in a GALS environment, since the inputs do not arrive at known instances of time. Therefore, in order to realise this system in an asynchronous or GALS environment, additional conditions are required to be added to:

- Prevent the system from entering deadlock states arising from the arrival of out of order inputs
- Exploit the advantages of asynchrony, by allowing the inputs to arrive as and when available leading to increased concurrency.

In order to incorporate the idea of asynchrony, the inputs must be allowed to arrive in any order and at any instant of time. This results in unbundling of inputs as shown in

Figure . Since the input signals cannot be scheduled to arrive at known instants, persistency property cannot be guaranteed. After applying this feature the state machine of the same system takes the form shown in Figure 5. This results in an unknown delay between the inputs. Therefore, from the figure it can be seen that the model has non-persistent steps at state s_1 and s_2.

To exemplify the following example is considered. Let$< In1 >$ arrive first, which causes $< O1, O2 >$ to execute in a maximal step. But before the execution of the maximal step $< O1, O2 >$ is completed, if $In2$ arrives then the system attempts to execute the maximal step $< O1, O2, O3, O4 >$. Therefore, the arrival of $< In2 >$ disables the step $< O1, O2 >$leading to violation of persistency property between the steps $< In2 >$ and $< O1, O2 >$at s_1. Non-persistent steps at the state s_2 can be easily shown in a similar way.

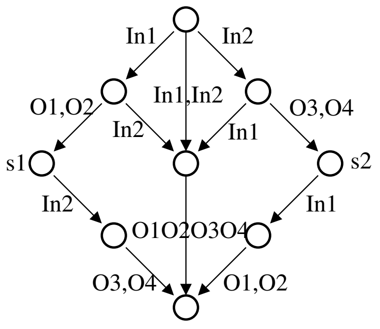


Figure 5: Unbundled out-of-order inputs system model

In order to avoid this situation, the system is not made to follow Max-O semantics globally. If it is possible to partition the system in such a way that none of the transitions are non-persistent in each partition, then the Max-O semantics can be restricted to each partition leading to a correct realisation of a concurrent system. This gives the motivation for the use of *localities.* In order to obtain a correct implementation of a GALS system from a synchronous specification, the synchronous system is required to be partitioned into localities, which are analogous to partitioned blocks. Therefore, the partitioning of the global synchronous system into localities and application of Max-O semantic in each locality, aids the removal of the global clock by guaranteeing the absence of deadlock and the realisation of correct input output dependencies.

3.1 Max-O semantics and validity criteria using Processes

The previous section presented the idea about Max-O semantics used to describe distributed architectures. This notion is required to be handled by the specification models, in this case PN models, used to describe such systems. The standard interleaving semantics for PN does not associate any notion of maximal firing by which a set of transitions are always fired concurrently. Therefore, *maximal output semantics* is introduced which binds sets of output transitions in order to fire them concurrently.

In this section we draw some equivalences between models of PN with *maximal output semantic* and *standard semantics*. The reason for obtaining such equivalences is to use PNs that are behaviourally equivalent under both the semantics due to the feasibility of verification and synthesis. Hence, the model used to represent our system are those that are equivalent under standard and Max-O semantics. Here, we require to define the restrictions that support the above equivalence. This is done with the help of theory of Processes, which was introduced in [8].

A process can be represented as a labelled acyclic graph, with places having at most one incoming and one outgoing arc. The processes can be viewed as subnets of unfolding. Let $\sqsubseteq$ be the prefix relation on processes. The nodes of the processes have identities, i.e. they are not anonymous. Therefore, if $\pi \sqsubseteq \pi^{'}$, then $\pi^{'}$ is a continuation of π rather than some unrelated to π process whose initial part is isomorphic to π.

Definition 3. Behavioural Equivalence

Let $\Sigma = \{P, T, M, \mu\}$ be a Petri net model. Let PN_{STD} be the reachability graph of Σ under standard semantics and PN_{max} be the reachability graph of Σ under the Max-O semantics. Let π and $\pi^{'}$ be set of all finite processes of PN_{STD} and PN_{max}, respectively, then

1. $\gamma^{'} \leq \gamma$
2. $\gamma \sqsubseteq \gamma^{'}$

where, $\gamma \in \pi$ and $\gamma^{'} \in \pi^{'}$.

The standard semantics have interleaved output steps and the Max-O semantics have maximal output steps. Hence, the interleaved semantics will have more permissive steps as compared to Max-O semantics. Therefore, intuitively we can say that the processes of standard semantics are greater than the processes of Max-O semantics.

To prevent the Max-O semantic from having additional events which are not permitted by the standard semantic, our second condition comes into play. Therefore, by enforcing the processes of PN_{max} semantics to be a prefix of some process of PN_{STD}, we address the above issue.

Figure 6 shows an example of a net that is equivalent under both the semantics. In a similar way, the PN models used to describe the synchronous or the distributed systems should be equivalent under both, standard and Max-O semantics.

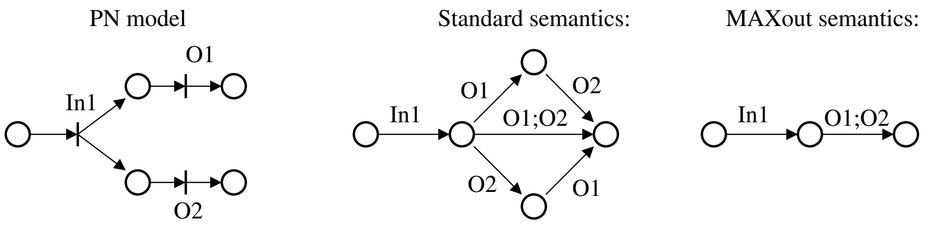


Figure 6: A PN equivalent under the two semantics

4 Synchronous model description

A more complex example is now considered to highlight the main aspects of our desynchronisation methodology. This will be running example for exemplifying the process of GALSification.

Figure. 7 shows a typical synchronous system. There are two inputs $In1$ and $In2$ to the block and seven outputs, namely, $O1, O2, O3, O4, O5, O6$ and $O7$ from the block. The system clock is used to clock the whole system globally. The PN model specification of such a system is shown in Figure 8(a). The state representation of the maximal firing semantics in a globally synchronous environment is shown in Figure

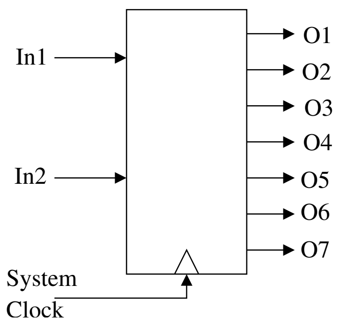


Figure 7: Synchronous Block

8(b).

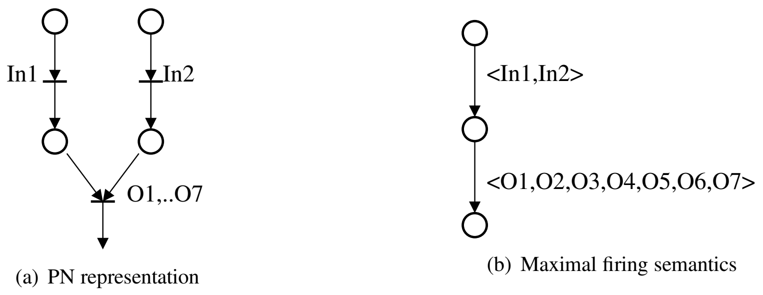


Figure 8: Synchronous model

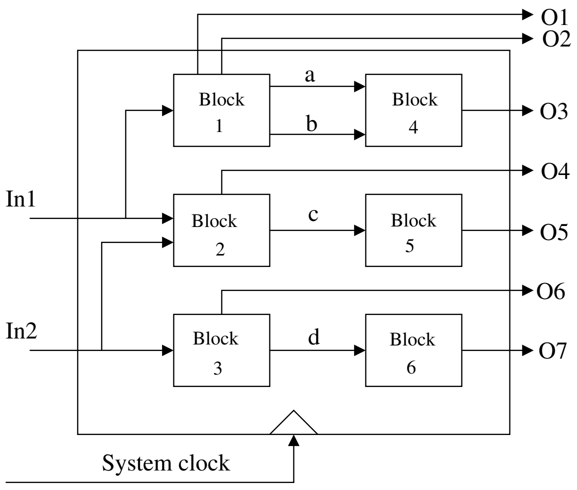
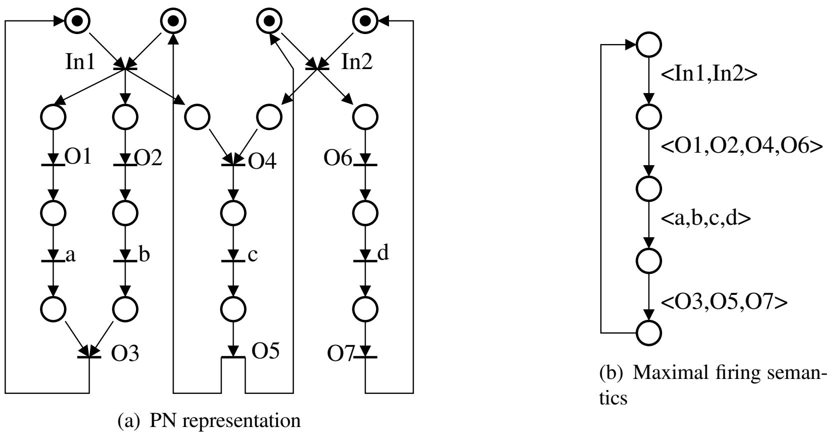


Figure 9: The synchronous system

Each synchronous system can be further divided into smaller computational blocks.

These blocks have their own input signals coming from and outputs going to other similar internal blocks. These signals, when seen from the top level of the single synchronous block, form the internal signals of the circuit. These smaller blocks have their own sets of internal signals. Such a system is exemplified in Figure 9. The signals a, b, c and d form internal signals to the overall synchronous block. A PN representation of such a system is shown in Figure 10(a). The reachability graph of such a system is shown in Figure 10(b). Therefore, the system can be partitioned into small computational blocks as shown in Figure 9. If the system is required to be further partitioned into smaller blocks, then the inputs $In1, In2, a, b, c$ and d can be split to aid the process of locality formation, as discussed earlier.



(a) PN representation

(b) Maximal firing semantics

Figure 10: PN model of the synchronous block

For the formation of localities and to aid asynchronous communication between the localities some transformations are applied at the PN model level. At the granularity of the individual blocks that compose the synchronous system, these internal signals form inputs to and outputs from the internal blocks, i.e a acts as an output from block 1, but behaves as an input for block 4. Since the internal signals are now interpreted as output from one block and input into the next block, transformations are applied on the net to incorporate this communication on the channel, in order to distinguish the outputs from the input signals for desynchronisation. This is necessary to incorporate the idea of localities which have sets of input and output transitions allocated to each locality.

Therefore, output from one locality forms the input to another. To do so, we partition the signal into output and input signals. For example, signal a is partitioned into a_O and a_In. To do this, the model needs to be transformed by inserting new internal signals, for which the transition insertion technique defined in Section 4.1 is used. This refinement leads to a modified PN model of the original system and is depicted in Figure. 11. The shaded blocks denote the insertion of signals in the original system.

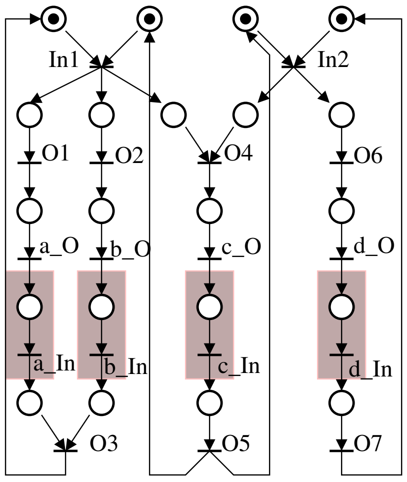


Figure 11: Modified model

4.1 Net Transformations and notion of validity

In order to obtain a distributed PN model of a system, some transformations are required on the model to aid the compartmentisation process. One such transformation is *Signal Insertion.* In this section, signal insertion by transition partitioning, is formally defined. The type of insertion is restricted to *sequential post insertion* because the insertion is to aid the partition of a signal into its output and input counterparts and hence eliminates concurrent insertions.

Definition 4. Transition partitioning

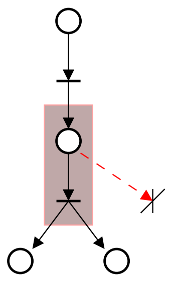
Given a *labelled Petri net* $\Upsilon = (\Sigma, I, O, l)$ where $\Sigma = (P, T, F, \mu_0)$, I is a set of inputs, O is a set of outputs, such that $I \cap O = 0$, l is a function that maps the transitions of the Petri net to the set $I \cup O \cup \{t_{int}\}$, where, $t_{int} \notin I \cup O$, the partition of the transition $t \in T$ yields an LPN $\Upsilon' = (\Sigma', I, O, l)$ with $\Sigma' = (P', T', F', \mu_0)$, where,

- $T' = T \cup \{u\}$, where $u \notin P \cup T$ is a new transition
- $P' = P \cup \{p\}$, where $p \notin P \cup T$ is a new place
- $F' = F \cup (\{t, p\} \cup \{p, u\} \cup \{(u, q) \mid q \in t\bullet\}) \backslash \{(t, q) \mid q \in t\bullet\}$

The notion of validity for signal insertion is straightforward and the transformation can be justified in terms of weak bisimulation which is well studied. Such a notion is presented in [[5], Proposition 5.3].

Conditions of valid transformations There are some restrictions that are required to be followed while inserting the signals.

- The newly inserted places form the interface places between the different localities. Therefore, these places cannot have the token stolen by another transition in conflict. To avoid a transition from stealing the token and resulting in running one locality into a deadlock, situation depicted in Figure. 12(a), should not be allowed. Hence, interface places cannot be choice places.
- If the signal has fan-outs, the buffer should be inserted before the fanout, instead of one buffer in each branch. The later can lead to formation of unnecessary localities due to numerous signal insertions. This is exemplified in Figure. 12(b).


(a) Conflicts

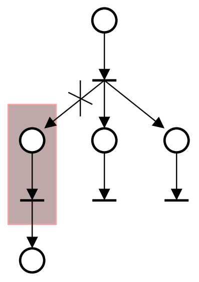

(b) Output fan-outs

Figure 12: Restrictions on transition splitting

Therefore, the allowable examples are shown in Figure 13.

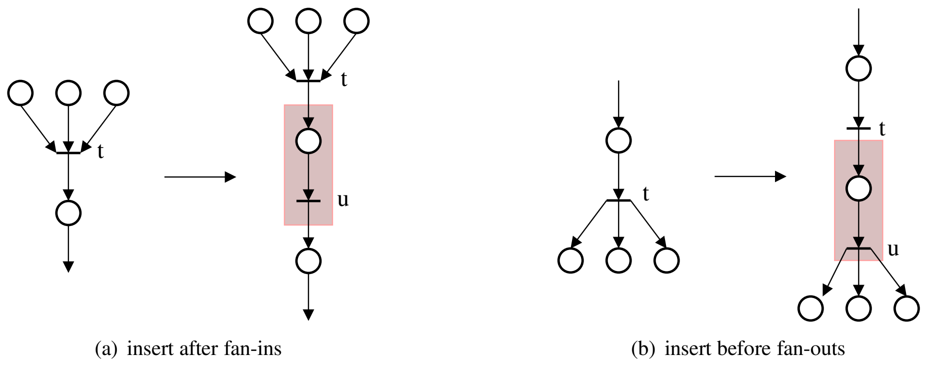


Figure 13: Transition partitioning

Transition re-labelling The transitions t (the transition which is split) and u (the newly inserted transition) are labelled by adding a post-fix $\_O$ to the label of t and $\_In$ to the label of u. This is done to associate meaning to the inserted signals which signify channel communication. Therefore, for the example shown in Figure 10, the transition labelled a is split into a_O, denoting output from block 1 and a_In, denoting input to block 4.

The newly inserted place $t\bullet$, can be regarded as a unit of storage, for instance a finite FIFO. This FIFO stores item of data before transferring it across to the next block.

For a synchronous system, the *Input transitions* should be able to fire as and when the tokens are available. On the reception of the inputs, all the outputs that are dependant on this input are generated together.

These steps thus defined can have different modes of execution. To define the synchronous execution modes we can define the following:

- Free-execution - This means that the events can be executed in any order when the trigger conditions for those events are available at deterministic instances of time, to transit the synchronous component from one configuration or marking to another.

- Seq-execution - This means only one trigger condition is sufficient to execute an event to transit the synchronous component from one configuration or marking to another.

- Max-execution - This means that in each step a maximal multiset of events, denoting the availability all the trigger conditions for the events, must be executed to transit the synchronous component from one configuration or marking to another.

Therefore, from the restrictions on the input/output transitions of a synchronous system, it can be derived that the input and output transitions can made to follow any of the above execution rules.

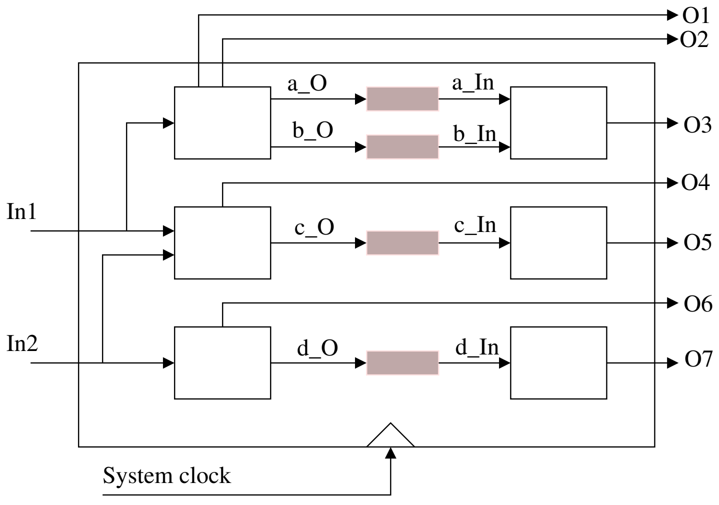


Figure 14: System architecture with storage units

The individual compartments, depicted in Figure 14, can now be viewed as a modular synchronous block with its own input and output signals.Therefore, each of these compartments will have to follow the synchronous behaviour.

Therefore, the original synchronous system can be now defined as a collection of these compartments, given by:

$$\Sigma \quad = \quad (P_1, T_1, F_1, \mu_1) \cup ...(P_n, T_n, F_n, \mu_n)$$

where, $P_1, ...P_n \subseteq P$, such that $P_1 \cap P_2.... \cap P_n \neq 0$, $T_1, ...T_n \subseteq T$, such that

$T_1 \cap T_2 ... \cap T_n \neq 0$, $F_1, ...F_n \subseteq F$ and $\mu_1, ...\mu_n \subseteq \mu_0$

Since each of these compartments are viewed as synchronous blocks, the input and the output transitions from these blocks also follow the same execution rules as discussed above.

5 Petri nets with localities

In order to model a distributed architecture from a synchronous system model, we apply the theory of Petri net with Localities which was originally introduced in [6]. In the previous work, the co-located transitions executed maximally. We extend this by making a distinction between input and output transitions and allowing the input transitions to execute as and when they arrive and restricting the output transitions to execute maximally. This extension is in direct relation to the synchronous behaviour, discussed in the previous sections.

The transitions in the PT-net belong to a fixed unique locality. The allocation of localities to the transitions is achieved my partitioning the PT net using a locality mapping function γ. This means if two transitions return the same value for γ they will be co-located.

A PN with *localities* is a tuple denoted by $NL = (P, T, F, \mu_0, \gamma)$, where the underlying PN is denoted by $_{UND}(NL) = (P, T, F, \mu_0)$ and $\gamma : T \to N$ is the location mapping for the transition set T. $\gamma(t)$ returns an integer value which denotes the locality of the transition t. Initially, for all $t \in T$, $\gamma(t)$ is set to 0, which denotes that the transition is unallocated.

A net can be partitioned into localities giving rise to the formation of smaller nets that constitute the original graph.

Definition 5. Let $\Sigma = \{P, T, F, \mu_0\}$ be an elementary net system. Then, the localisation leads to the division of the net into n smaller nets, denoted by,

$$\Sigma_i = (P_i, T_i, F \cap (P_i \times T_i \cup T_i \times P_i), P_i \lhd \mu_0),$$

for $i = 1$ to n, where n is a set of integers, each $T_i \subseteq T$ so that $(T_1 \cap T_2 \cap ....T_n) \neq \emptyset$ and each $P_i \subseteq P$ so that $(P_1 \cap P_2 \cap ....P_n) \neq \emptyset$, $P_i \lhd \mu_0$ is defined by the following:

If $\mu_0 : P \rightarrow \{0, 1\}$, then $\forall p \in P_i$, $\mu_{0i} : P_i \rightarrow \{0, 1\} | \mu_{0i}(p) = \mu_0(p)$.

Synchronous components versus localities In order to map the synchronous components into localities, the execution modes of the synchronous counterpart should be supported by the localities, thus formed. Therefore, the three execution modes are redefined which are incorporated to support the specification of synchronous behaviour.

Definition 6. Free-enabled

A multiset of transitions U is free-enabled at a marking μ, if $\mu \geq pre_N(U)$. This is denoted by $\mu[U >$.

Definition 7. Seq-enabled

A multiset of transitions U is seq-enabled at a marking μ, if U is free-enabled at marking μ and $|U| = 1$.

Definition 8. Max-enabled

A multiset of transitions U is max-enabled at a marking μ, if U is free-enabled at marking μ and there is no transition t such that $\mu[U + \{t\} >$.

Therefore, it can be seen that each locality is able to emulate the behaviour of the synchronous components, giving rise to semantic preservation when synchronous components are mapped onto the localities.

The next section presents some rules for the allocation of localities in a synchronous system.

6 Notion of partitioning correctness

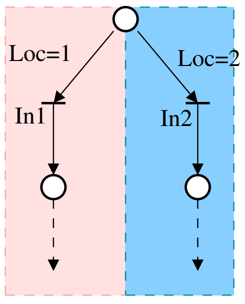
As discussed above, a synchronous system can be desynchronised into a distributed architecture by unbundling the inputs and forming localities. The formation of these localities should satisfy some correctness properties to ensure correct desynchronisation. The partitioning of the GALS deployment of a synchronous system is correct w.r.t the original synchronous system if there is a behavioural equivalence between the GALS system and the initial synchronous specification.

This is formally defined in the following way:

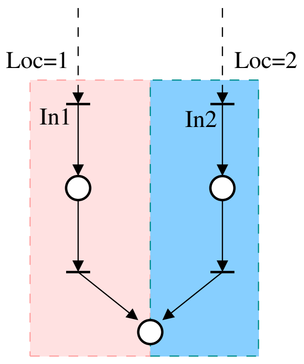
Definition 9. Let $\Sigma = \{P, T, F, \mu_0\}$ be an elementary net system. The partitioning $\Sigma = \Sigma_1 \cup \Sigma_2 \cup ....\Sigma_n$, each belong to locations $L_1, L_2...L_n$, respectively, is correct at a marking μ iff for all steps of transitions $U_1 \subseteq T_1$,..$U_n \subseteq T_n$, where $U_1, ...U_n$ are enabled in $\Sigma_1, ...\Sigma_n$, respectively, the combined step $U_1 \cup U_2 \cup ...U_n$ is enabled in Σ. This denoted as,

$$(\mu \lhd P_1)[U_1 >_{\Sigma_1} \wedge (\mu \lhd P_2)[U_2 >_{\Sigma_2} \wedge ...(\mu \lhd P_n)[U_n >_{\Sigma_n} \Rightarrow \mu[U_1 \cup U_2 \cup ...U_n >_{\Sigma},$$

for all $U_1 \subseteq T_1, U_2 \subseteq T_2 ..., U_n \subseteq T_n$.



(a) Conflict-choice place



(b) conflict-merge place

Figure 15: Conflict between transitions

Conflict Resolution In order to adhere to the above criterion, the locality allocation should satisfy correctness properties for *conflict resolution*. For example, an incorrect partition is shown in Figure 15(a). The net Σ is partitioned into Σ_1 and Σ_2 belonging to localities L_1 and L_2, respectively, so that the transition t_1 is allocated to locality L_1 and t_2 is allocated to L_2 and therefore, $T_1 = t_1$ and $T_2 = t_2$. Now, substituting p for μ, leads to markings $\{p\}[\{t_1\} >_{\Sigma_1}$ and $\{p\}[\{t_2\} >_{\Sigma_2}$ in each of the localities but

$\{p\}[\{t_1, t_2\} >_\Sigma$ is not true. Hence, the partitioning is incorrect. Such an occurrence that leads to an incorrect partition can be similarly shown for Figure 15(b).

The notion imposes the transitions in conflict to be placed in the same locality. The locality optimisation technique can lead to occurrence of such a situation. Hence, care must be taken while inserting the input/output bridges in the partitions. Therefore, the correctness can be guaranteed if the following criterion is satisfied:

Criterion 1. *Let $\Sigma = \{P, T, F, \mu_0\}$ be an elementary net system that has been partitioned into $\Sigma_1, \Sigma_2, ..., \Sigma_n$. If transitions from the partitions do not share preconditions or postconditions , or*

$$\bullet T_1 \cap \bullet T_2 \cap ... \bullet T_n = T_1 \bullet \cap T_2 \bullet \cap ... T_n \bullet = 0$$

then the partitioning is correct.

Step Persistency Another correctness property that the partitioned blocks must satisfy is *Step-persistency.* The reason for identifying and handling non-persistency is already presented in Section 3. The non persistent transitions can be identified and made persistent by executing the following steps.

1. For each output transition in the net, identify the set Out of output transitions that are dependant on more than one input transitions.
2. for each output transition in Out, return the set In of input transitions, on which the output depends.
3. For each input in In, check if it causes more than one output transition.
4. Return the set $persist$ of input signals for which (3) is true.
5. Return the output transition $O1$, such that $O1 \in Out$ and the input that causes it belongs to the set $persist$.
6. Connect the output obtained in (5) with each of the input signals in the set $persist$ through an output-input transition pair as exemplified in Figure .

The signals that are inserted are sets of output-input transition pairs, denoted by Ox and Ix, which behave as internal or silent events for the overall system. These signals satisfy the notion of validity of signal insertion discussed in Section 2. Adding extra signalisation does not introduce any new behaviour in the system. This is because the signals are added as pairs of input/output transitions and emulates a buffer which only introduces some extra delays in the stem without affecting the consistency of the signals. These signals aid the formation of localities which have all inputs arriving before the emission of all the outputs.

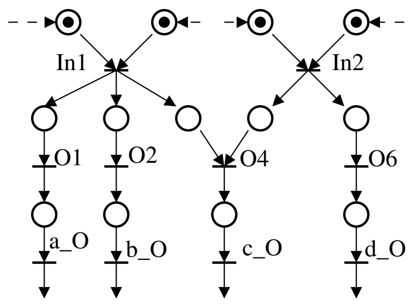


Figure 16: Persistency check

The above is exemplified by taking Figure 16 into consideration. Since $In1$ and $In2$ can arrive with unknown delays, the model needs to be modified to remove non-persistency as described in this section. From (1) we obtain the set $Out := O4$ as it is the output transition that is dependant on more than one input transition. From (2) set In returns $\{In1, In2\}$ which cause the output transition $O4$. For each input $In1$ and $In2$, step (3) returns $true$, since both the inputs cause more than one output signals. Therefore, the set $persist$ in step (4) returns $\{In1, In2\}$. Step (5) returns $O4$. The additional signals, in the form of output-input transition pair, are inserted between $In1$ and $O4$ and between $In2$ and $O4$.

Therefore, at the model level, the system depicted in Figure 11 is transformed into the system depicted in Fig. 24. The block level representation of the part of the circuit, under consideration, is depicted in Figure 18.

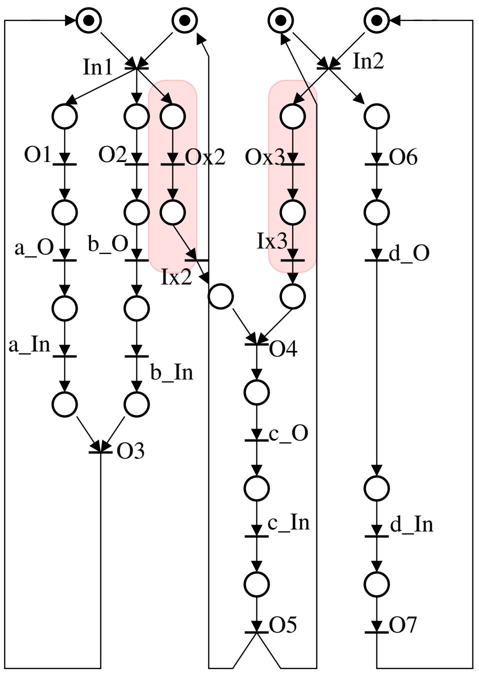


Figure 17: Bridge formation

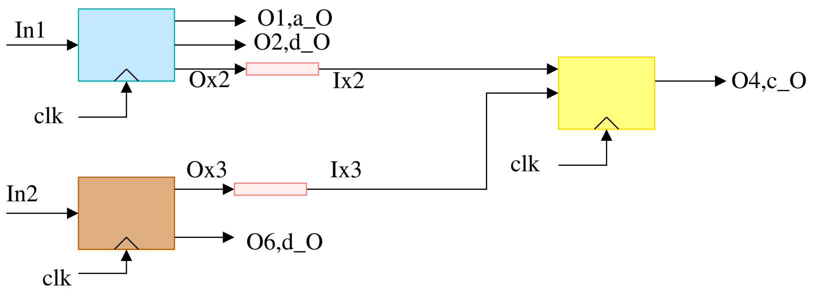


Figure 18: Block level representation

7 Rules for Locality Allocation

Taking the partitioning constraints, obtained from the correctness properties in the previous section into consideration, we present some rules for the allocation of localities. This leads to correct localisation of transitions. The method of deployment of synchronous systems over localities requires the adherence to the following rules:

1. In order to obtain the partition sets of transitions, the information about the loca-

tions of each input and output transitions of the PT-net is required. We derive the localisation of each input and output transitions of the synchronous circuit from the input output dependencies. For example, if output transition y is computed in locality L, then so does the input signals, x in this case, that are required for the execution of y. Therefore, the input x must also be located in L. Such a localisation will directly influence the localisation of internal signals.

2. When allocating localities, all the branches of the choice, i.e. all the transitions of the choice place should be placed in the same locality. Therefore, two transitions in conflict should not be placed in two different localities. Violation of this property leads to an erroneous locality allocation for transitions, discussed earlier in Section 6.

3. In each locality, all the transitions should satisfy the persistency constraint. If the constraint is violated, additional signals are required to be inserted to eliminate non-persistency. This is elaborated in Section 6.

Formally, the above rules lead to the formation of a system that can be defined by the following definition:

Definition 10. Let $\Sigma = \{P, T, F, \mu_0\}$ be an elementary net system. Then, the partitioning leads to the division of the net into n smaller nets, denoted by

$$\Sigma_i = (P_i, T_i, F \cap (P_i \times T_i \cup T_i \times P_i), \mu_0 \lhd P_i),$$

for $i = 1$ to n, where n is a set of integers, each $T_i \subseteq T$ so that $(T_1 \cap T_2 \cap ....T_n) = \emptyset$ and each $P_i \subseteq P$ so that $(P_1 \cap P_2 \cap ....P_n) \neq \emptyset$.

These rules lead to a correct localisation of input/output transitions.

8 Allocation of Localities

This work does not address the problem of finding the optimum localisation of the computations w.r.t the performances of the resulting distributed system. The localisation of all the actions of the synchronous system is derived directly from the localisation of the input and output signals. This section also presents an optimisation for the locality allocation methodology be redistributing transitions over localities to avoid locality overloading arising from large input fan outs.

8.1 Algorithm for locality allocation

In order to allocate localities to the transitions of a system, we require to define some methods which are presented in Algorithm 1. The algorithm implements a locality allocation scheme that is in accordance with the rules presented in the previous section 7.

The algorithm described in this section incorporates a bi-directional subnet traversal in order to allocate localities to the transitions it visits. It takes as input a Petri net model of a synchronous system denoted by Σ. The output of the algorithm is a Petri net model of the synchronous system Σ, with locality information added to each transition in the model.

This algorithm defines the following methods and the functionality of each is described below:

Tr_f: This function denotes the forward subnet traversal. This function traverses the net and adds the transitions to sets T_v and $Tail$, depending on certain conditions. Once the sources have been identified, for each element of the set the net is traversed forward, assigning the each transition visited to the set T_v. If during the traversal a transition is reached, which already belongs to the the set T_v due to a previous net traversal, the traversal is terminated at this node and this transition is added to the set $Tail$. $Tail$ denotes the node where a traversal stops and marks the boundary for a locality. The $Tail$, in contrary to $Source$, forms the output interface for a particular locality. While traversing the net forward, if a transition is reached that belongs to $Source$, then the predecessor transition(s) is added to the set $Tail$.

Alloc_Tail: Once the set of tail transitions have been identified. This function takes the set $Tail$, as a parameter and assigns localities to all the tails of a given source.

Tr_b: This function defines a backward net traversal starting at the tail and terminating at the source. This function finds a set of unallocated transitions on the backward path to the source(s). If a node that belongs to T_v or $Source$ is reached which is already allocated, the locality value is not changed. Therefore, the node retains its original locality, if such a situation arises.

Assign_Loc: This function takes a set of transitions and allocates localities to them depending on the locality values of the *Tail* transitions.

Method:

1. A set of $Source$ is identified which consists of all input transitions of the net.

2. For all the transitions belonging to $Source$, the following steps are executed:

   - Traverse the net forward assigning each transition to the sets T_v or $Tail$.
   - Allocate localities to all the tails.
   - Traverse backward starting from the tail and terminating at the source assigning all the unallocated transitions to the set $LocAssign$, in its path.
   - Finally allocate the transitions in the set $LocAssign$ with the locality of the $tail$ transition chosen from the set $Tail_{new}$, which is a set of assigned $tail$ transitions.

Finally, we obtain a set of all transitions of the net, that are allocated to at most one locality in the system.

Algorithm 1 Allocation of localities

```
function: Allocate_Localities
input: $\Sigma = \{P, T, F, \mu_0\}$
output: $\Sigma_{out} = \{P, T, F, \mu_0, L\}$
$T_v := \emptyset$; $Tail := \emptyset$; $Tail_{new} := \emptyset$; $LocAssign := \emptyset$
for each $t \in T$
  $\gamma(t) = 0$
$Source \leftarrow$ set of all inputs of the system
$n \in N \leftarrow$set of natural numbers
```

```
$n = 0$
for each $t_{in} \in Source$ do
  if $t_{in} \notin T_v$
    $tr\_f(t_{in})$
      for each $t_{tail} \in Tail$ do
        $AllocTail(t_{tail})$
      for each $t_{tail} \in Tail$ do
        $tr\_b(t_{tail})$
      choose $t \in Tail_{new}$
      for each $t_{ua} \in LocAssign$
        $\gamma(t_{ua}) = \gamma(t)$
```

Forward net traversal Once the set of sources is identified, these form the nodes of forward net traversal. In this method, the transitions are assigned to sets T_v (a set of visited transitions) and $Tail$ (a set of tail transitions) depending on the following conditions:

T_v: if the transition passed as a parameter, it is added to the set of visited transitions.

$Tail$: when the transition does not belong to the set $Sources$ and the successor transition belongs to the set $Sources$, then it is added to the set of tail transitions.

The traversal is terminated for a given path when the tail is identified. This method is repeated until all the the tail transitions, for a given source, are identified.

Algorithm 2 Forward net traversal

```
function: tr_f(t)
input: $\Sigma = (P, T, F, \mu_0)$, $Sources$
output: $\Sigma = (P, T, F, \mu_0)$, $T_v \subseteq T$, $Tail \subseteq T$
for each $p \in t\bullet$ do
  for each $t_{succ} \in p\bullet$ do
    $T_v := T_v \cup t$
    if ($t_{succ} \notin T_v$) then
      $T_v := T_v \cup t_{succ}$
    else $Tail := Tail \cup t_{succ}$
    if $(t \notin Sources \,\&\&\; t_{succ} \in Sources)$
      $Tail := Tail \cup t$
    if $(t \notin Tail)$
      if($t_{succ} \notin Tail$)
        tr_f($t_{succ}$)
```

Allocation of localities to tail transitions The previous method identifies a set of tail transitions that could be allocated or unallocated. In this method, all the allocated and unallocated transitions are identified and assigned to sets $Alloc$ and $Nalloc$, respectively. If the $Alloc$ set is null, denoting that all the tail transitions are unallocated, we assign an integer value to all the tail transitions. If $Alloc$ is not null, a tail is randomly chosen (this is because all the tail transitions in the set will have the same locality) from the set and its locality is assigned to all the other tail transitions in the set $Nalloc$. This methodology assigns the same locality values to all the sources that form presets of a set of output transitions.

Algorithm 3 Locality allocation for the tail transitions

```
function: AllocTail($Tail$)
input: $Tail \subseteq T$
output: $Tail_{new} \subseteq T$
$Alloc = \emptyset$
$Nalloc = \emptyset$
for each $t \in Tail$
  if ($\gamma(t) \neq \emptyset$) then
    $Alloc := Alloc \cup t$
    else $Nalloc := Nalloc \cup t$
if $(Alloc == \emptyset)$ then
  $n = n + 1$
  for each $t \in Nalloc$
    $\gamma(t) := n$
    $Tail_{new} := Tail_{new} \cup t$
else
  choose $t_{alloc} \in Alloc$
  for each $t \in Nalloc$
    $\gamma(t) := \gamma(t_{alloc})$
    $Tail_{new} := Tail_{new} \cup t$
```

Backward net traversal In this method, for each of the tail transitions obtained from Algorithm 3 which are an element of the set $Tail_{new}$, the net is traversed in the backward direction to all the unallocated sources that are reached on the backward traversal path. In the path, it assigns all the transitions that are unallocated to a set called $LocAssign$. When the traversal reaches a source that does not belong to the set T_v, the transition is assigned to the set T_v, so that this transition is not need to be dealt with anymore since it has already been allocated a locality owing to another backward traversal.

Algorithm 4 Backward net traversal

```
function tr_b($t$)
input: $\Sigma = (P, T, F, \mu_0)$
output: $\Sigma = (P, T, F, \mu_0)$, $LocAssign \subseteq T$
for each $p \in \bullet t$ do
  for each $t_{pred} \in \bullet p$ do
    if ($\gamma(t_{pred}) == 0$)
      $LocAssign := LocAssign \cup t_{pred}$
    if ($t_{pred} \in Source$)
      $T_v := T_v \cup t_{pred}$
    else $tr\_b(t_{pred})$
```

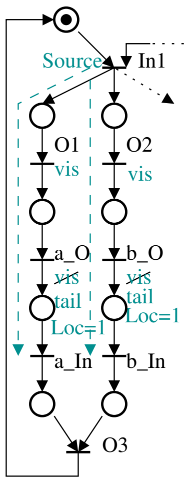
8.2 A simple Example

1. Consider a complete model of a synchronous system Σ in the form shown in Figure 11. This system is partitioned into localities by following the steps of the

algorithm. The steps are elaborated by applying the rules on the system model.

2. Let $Source$ be a set of all input signals that are sent to each internal computational block in the system Σ. It is represented by $Source = \{In1, In2, In3, In4, In5, In6\}$.

3. For each input transition $t_{in} \in Source$, the net is traversed forward, adding each transition visited, to the set T_v, including the source transition t_{in}. For example, in the forward traversal along the first branch for input $In1$, the transitions that are assigned to the set T_v are $In1$, $O1$ and a_O.



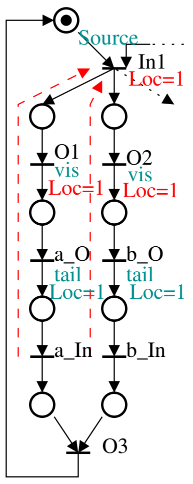
(a) Forward Traversal

Figure 19: Net Traversal

4. The forward traversal assigns transition t to a set $Tail$ if the transition is such that its successor transition, denoted by $t_{succ} \in p\bullet$, where $p \in t\bullet$, belongs to the set $Source$. This is exemplified in Figure 19.

5. This procedure is repeated for each branch of the initial input source. This is depicted in Figure 19.

6. Once all the tails have been identified for a given root, we assign the same locality value to all the tails. For example in Figure 19, transitions labelled a_O and b_O are assigned to the same locality 1.



(a) Backward Traversal

Figure 20: Backward net traversal

8. For each tail, we traverse backward and assign the same locality value to all the transitions till all the possible backward paths are assigned a locality, each path terminating at the transition that belong to the set $Source$. This is shown in Figure 20. On the backward traversal path allocation continues until all the sources in its path have been allocated.

10. The above steps are repeated for all the inputs in $Source$.

The above steps lead to the partition of the system into localities. The partitioned system is shown in Figure 21. The places shared by the localities depict the storage units that form the interface between two localities. Table 1 shows the different localities and the input and the output signals assigned to each locality.

To guarantee partitioning correctness as discussed in Section 6, the net is checked for *step-persistency* before the locality allocation algorithm is applied. Relevant signals are inserted if there is any persistency violation.

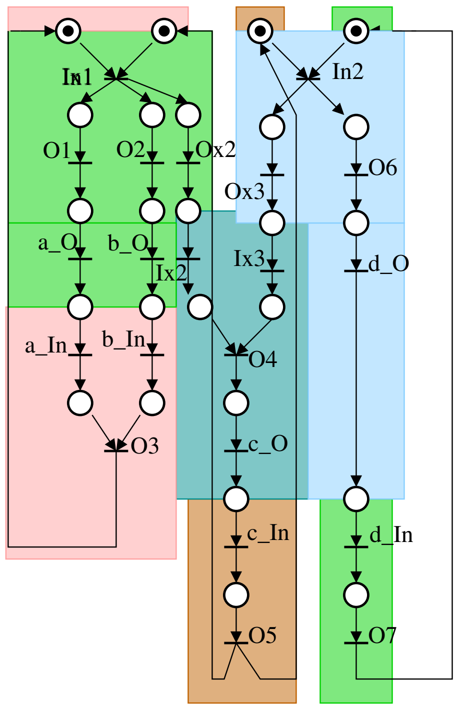


Figure 21: Partition Optimisation

The partitioning algorithm presented, also satisfies the correctness criterion, namely, *conflict-resolution*. Adherence to *conflict-resolution* is exemplified in Figure 22. As can be seen from the figure, the transitions in conflict, namely a_O and b_O are placed in the same locality. Since, *choice places* are contained inside each locality, it can be easily seen that the algorithm also contains the *merge places* inside the localities, owing to the input/output dependency notion used to derive the locality formation.

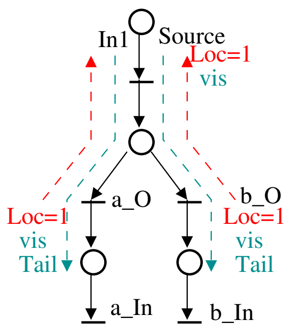


Figure 22: Locality allocation for conflicts

Locality	Inputs	Outputs
$L1$	$In1$	$O1, O2, a\_O, b\_O, Ox2$
$L2$	a_In, b_In	$O3$
$L3$	$In2$	$Ox3, O6, d\_O$
$L4$	$Ix2, Ix3$	$O4, c\_O$
$L5$	c_In	$O5$
$L6$	d_In	$O7$

Table 1: I/O allocation to localities

9 Locality Optimisation

Let us consider the system shown in Figure 23. The rules of locality allocation, allocates localities based on input output dependencies. Therefore, large input fanouts could lead to the formation of large localities. In order to avoid the overloading, the fanouts can be partitioned so that they are allocated to different localities. We do not present an optimum criterion of obtaining as many or as few as possible localities, because it is system dependant. Therefore, depending on a system requirement, large localities can be further partitioned into smaller localities to avoid overloading.

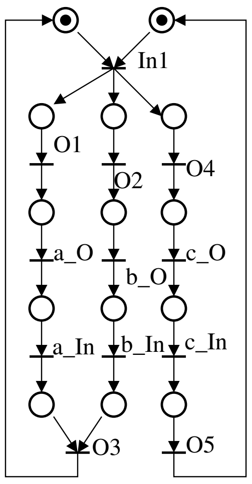


Figure 23: Input fan-outs

This can be done by introducing extra signalisation in the path of the fan-out that requires segregation. A manual signal introduction technique is introduced in this section which is applied to the original un-partitioned system. The signals that are inserted are sets of output-input transition pairs, denoted by Ox and Ix, which behave as internal or silent events for the overall system. Such an insertion is depicted in Figure 24. These signals abide by the notion of validity of signal insertion discussed in Section 4.1. The output signal Ox can only be enabled by $In1$, which requires one or more of its fan-out transition to be relocated to another locality. The transition Ox is followed by Ix which in turn enables the outputs, which were originally activated by $In1$. Hence, these signals enforces the formation of extra localities, in turn reducing the load on any one locality. The algorithm presented in the previous section can now be applied to this transformed system. When the algorithm is applied the following is obtained:

- The transition Ox belongs to the same locality as $In1$.
- The newly inserted input signal Ix and the output transitions that follow along the path are assigned to a new locality, say X.
- If these output signals require other source signals, besides $In1$, for their activation, then these sources are also located in the same locality and so are the output transitions that are dependant on these sources.

Adding extra signalisation does not introduce any new behaviour in the system. This is because the signals are added in pairs and act as buffers which only introduce extra delays in the system without affecting the consistency of the signals. While inserting the signal it has to be made sure that the choice places are not split. The choice branches should be contained in the same locality, as discussed earlier.

10 GALSification

Each of the localities formed have a clock signal that activates that locality. In a synchronous system this clock is the global clock which is sent to each of the localities. At every clock edge new data is read from the input signal ports. These inputs can arrive

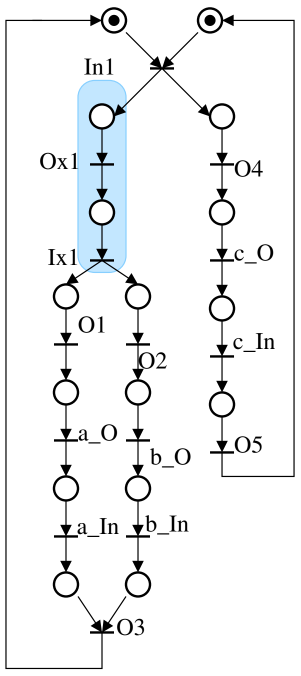


Figure 24: Bridge formation

from other localities or the environment. Due to the delays in the wire, the inputs may arrive at different times.

Due to this phenomenon the order of inputs is not guaranteed. On the contrary, a clock edge triggers all the active input and output transitions maximally.

If the input maximal steps are unbundled and the restrictions on the input and the output transitions, based on the correctness properties, are guaranteed then the global clock which samples these input and output transitions, is no longer a requirement at the locality interfaces. Therefore, we can eliminate the global clock and substitute each locality with local clocks. Therefore, each locality behaves like an independent synchronous system which communicates asynchronously with other localities.

The signals are communicated from one locality to another through an asynchronous domain. Hence, the actions performed in this domain are causal. Hence, the localities have to synchronise with each other while sending and receiving data with the receiver and sender blocks, respectively. This leads to the formation of a GALS architecture.

Therefore, we can deduce that if the I/O conditions are satisfied the globally synchronous system can be translated into a GALS system. Hence, to enforce the I/O conditions in each locality, we impose that the inputs are interleaved to guarantee correctness by allowing them to be received as and when they arrive and outputs are maximal, i.e, if they are active they will fire concurrently. Hence, we obtain the notion of maximal output or Max-O semantics which is obtained by substituting max-enabled transitions of each locality to the outputs generated from them.

11 Implementation of wrappers

Until now, the clock signal in the net has been represented implicitly. The transitions were coupled with the notion of clock. Therefore, the activation of a transition signified that the clock's positive edge for positive logic or negative edge for negative logic, is also active. Firing of the transition activates the negative edge for positive logic and positive edge for negative logic. In a globally clocked architecture, no additional clock circuit is required to control the clock. On the contrary, a clock control circuit is required when local clocks are deployed to the individual localities. This clock control circuit synchronises the signals while crossing the domain from one locality to another.

Hence, the clock transitions need to be shown explicitly to enable this control. Therefore, instead of coupling transitions with the clock information, we introduce explicit clock transitions to signify the positive/negative edges. The formation of the localities enables the treatment of each locality as a black box. Therefore, the process of reading the inputs and the producing the outputs is not dealt with. It is considered that the inputs are read and the outputs produced in one clock cycle. For the sake of simplicity, in the examples the operations are shown to be completed in one clock cycle.

To show the clock control, the signals which denote the availability of inputs and outputs are explicitly depicted. Since, the signals travel from one locality to another, which belong to different clock domains, the transfer from the clocked domain to the unclocked domain and should occur when the clock is inactive. The clock should not be triggered until this transfer is completed. When the transfer is completed the clock can be triggered again to process another set of data. A similar process occurs when a signal is transfered from an unclocked to a clocked domain in the form of inputs. When all the input signals have been received which signifies the availability of input data, the clock is allowed to go high to process the data and produce relevant outputs. The above phenomena elaborates the working of the clock control architecture in the deployment of a synchronous system into a GALS system. This is exemplified in Fig. 25. This is in accordance to the clock control technique presented in [7].

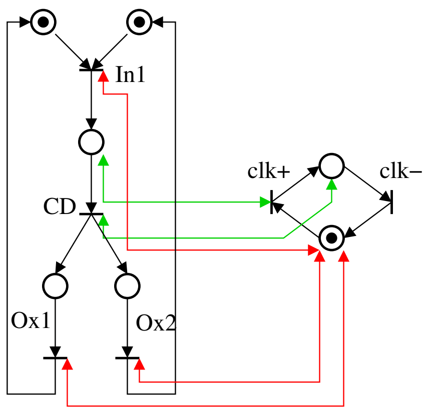


Figure 25: Clock Control

The green arrows depict the operations of the net that triggers the positive edge of the clock. For the input $In1$ to be received, the clock requires to be inactive or low to avoid metastability (Chapter 2, Section). On the reception of the signal, the clock is triggered to process the data. The completion of the process is denoted by the *completion detection* signal CD. When signal CD is received, the clock is lowered for the transfer of outputs.

Such transformations are applied to all the localities to handle local clocks. The insertion of the signal CD does not affect the behaviour of the system as it is an *internal* signal and it adheres to the notion of validity of signal insertions discussed in Section 2. It is inserted after the reception of the input signal and therefore does not delay the input signals.

In a similar way, such implementation is obtained for all the localities. The final model has inputs going in and outputs coming out of each locality to be communicated to the other localities which is done through asynchronous FIFOs. The FIFOs are the interface places that act as storage units for the system. Finally, a system is obtained with localised clocks which communicate with each other using asynchronous FIFOs.

12 Conclusion

This chapter addressed the problem of synthesising delay insensitive wrappers for GALS implementation. It presented an efficient desynchronisation and wrapper synthesis methodology which employed PN as its model of abstraction. The modules thus constructed using PNs were smaller than the ones obtained from the previous method and thus were easily synthesisable, even for large complex circuits. The GALS system was obtained by applying the theory of localities to a synchronous system model preserving the synchronous properties of the input output signals. This chapter also defined and showed consistency between two behaviour preserving transformations, namely, signal insertion and localisation of transitions at different stages of the desynchronisation process.

As a result this chapter presented a desynchronisation methodology with a fast route to synthesis, preserving the IO behaviour of the synchronous systems.

Future Work The synthesis process needs to be automated to reduce design time and designer intervention. The locality allocation can be further optimised to minimise interconnection between localities, yet still containing the choices between signals in the same locality to guarantee correctness, as previously discussed. The protocol used in the proposed algorithm, can be optimised to increase the component speed. We would also take into consideration that the system may require more than one clock cycle for a particular computation. Hence, a counter can be introduced to count the number of clock cycles required for a computation and allow the clock to tick for the required number of cycles while preserving the behaviour of the system.

References

[1] J.Cortadella,M. Kishinevsky, L. Lavagno, A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. In IEEE Transactions, Vol. 47, No. 8, pp. 859-882, 1998.

[2] D.Potop-Butucaru, B. Caillaud and A. Benveniste. Concurrency in Synchronous Systems. In Proceedings of ACSD 2004, Hamilton, Canada, 2004

[3] M. Nielsen, G. Rozenberg, and P. Thiagarajan. Elementary transitions systems. In Theoretical Computer Science, 96:-33, 1992.

[4] S. Dasgupta, D. Potop-Butucaru, B. Caillaud, A. Yakovlev. Moving from Weakly Endochronous Systems to Delay-Insensitive Circuits, Proceedings of the Second Workshop on Globally Asynchronous, Locally Synchronous Design (FMGALS 2005). In Electronic Notes in Theoretical Computer Science, Vol. 146, No.2, January 2006, pp. 81-103, 2005.

[5] A. Madalinski, Interactive Synthesis of Asynchronous Systems based on Partial Order Semantics. PhD thesis, University of Newcastle, 2006.

[6] M. Koutny, M. Pietkiewicz-koutny, Transition Systems of Elementary Net Systems with Localities. In Proceedings of CONCUR, Bonn, Germany, 2006, pp 173-187.

[7] M. Krstic, E. Grass, C. Stahl, Request Driven GALS Technique for Wireless Communication Systems. In Proceedings of International Symposium of Advanced Research in Asynchronous Circuits and Systems, March 2005, pp. 76-85.

[8] V. Khomenko, A. Madalinski, A. Yakovlev, Resolution of Encoding conflicts by Single Insertion and Concurrency Reduction Based on STG Unfoldings. In Proceedings of ACSD, Turku, Finland, 2006, pp 57-66.