# Desynchronisation Technique using Petri nets

Sohini Dasgupta[1] and Alex Yakovlev[2]
[1]University of Manchester, [2]Newcastle University

**Abstract**

In this paper 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 partitioning of a synchronous system is achieved by unbundling the input transitions and allowing the output transitions to fire in maximal steps, in order to enable asynchronous communication in a distributed environment. Our model satisfies the two essential correctness properties, namely, semantics preservation and deadlock prevention, during the shift from maximal firing semantics, followed by synchronous systems, to standard interleaving semantics for input transitions and maximal step firing semantics for output transitions, followed by GALS architectures. The formation of localities is supported by adding internal signals which are necessary for building wrappers in the localities that will generate local clock enables. These wrappers can be subsequently synthesised using PN based synthesis tools.

## 1 Introduction

This paper introduces a new methodology for the desynchronisation of synchronous systems into globally asynchronous and locally synchronous (GALS) architectures. Different formal techniques have been proposed for GALS in the last ten years, e.g. [1, 2]. Previously in [3], Transition Systems ($TS$) were used as the specification model to describe the synchronous systems for the purpose of desynchronising them into GALS architecture. The models obtained for each synchronous module can be very large and complex due to the weak handling of concurrency posed by the desynchronisation methodology used. Concurrency is a prerequisite for the specification of synchronous systems for handling asynchronous communication for their GALS deployment. Moreover, these models are translated into Petri nets in order to use existing asynchronous tools for logic synthesis.

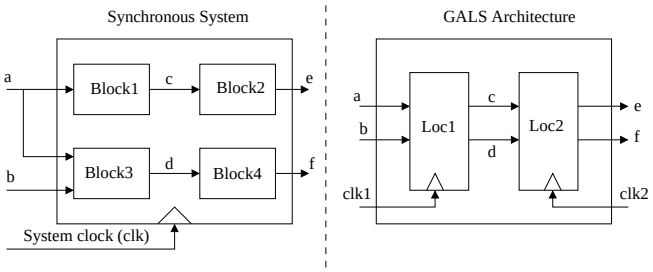



Figure 1: Synchronous system transformation into distributed architecture

Therefore, the complexity of the transition system, obtained from the previous methodology, and the computational complexity of the PN synthesis of these models 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. PN provides a high level system description model where the transitions are mapped to actions on channels or symbols. This technique uses labelled PNs as a way to specify logical dependencies between these actions and the I/O events of the system and its constituent blocks. The notion of synchrony/asynchrony is applied in our framework through the semantics of transition execution. The execution of a net is subject to certain rules which are determined by the way logically enabled transitions are scheduled to fire depending on some conditions on places. These conditions are imposed by the semantics applied to meet the clocking requirements of the system. It is assumed that in a truly synchronous execution, the firing of all logically enabled transitions is done according to the policy of maximal step (*see Definition 1*) semantics. In a true asynchronous execution, we use step semantics with all possible interleavings.

The theory behind the new technique uses the concept of *Localities,* inspired by [5], which helps in describing the distribution of a synchronous system over asynchronous architectures owing to its strong structural and functional correspondence with GALS architectures.

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. In this work we assume that the mechanism for clock generation can be built in one of the known ways:

- using the independent global clock and synchronisers (e.g. based on two-flop structures) associated with each input,
- internal clock generated from the inputs (event or data-driven clocking)

The notion of *localities*[5], in application to biological membrane systems, introduces 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 localities to behave like independent synchronous systems. Therefore, the global clock can be eliminated and each locality can be employed with local clocks which govern the actions assigned to them. Such an occurrence draws a parallel between systems obtained from the proposed desynchronisation scheme and *Endochronous systems*, where a synchronous program can reconstruct its timing from its internal actions and does not require to use the environment as an external stimulus for it.

As figure 1 shows, more than one component can be mapped onto each locality depending on some rules and optimisation criteria, discussed later. These localities created would then communicate with each other asynchronously due to the absence of a clock signal 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 these properties have been dealt with in Section 6 and 7, respectively.

We attempt to present in this paper some preliminary ideas for a method which performs desynchronisation of systems using the modeling language of PNs and their "synchronisation paradigm" expressed semantically in terms of steps and max-parallelism. This method defines important conditions for correct desynchronisation of a Petri net model of a system. These conditions are discussed in the context of the application of the execution semantics of Petri nets. The method has the potential of allowing the designer to move in the design space of a variety of different options for desynchronisation between fully synchronous and fully asynchronous. Particular solutions can be determined by the criteria for desynchronisation, such as performance optimisation. To illustrate a *particular* case of the design solution search in this spectrum, the

paper ends with a sketch of an algorithm which takes a Petri net model of a fully synchronous system and produces a system model with a set of localities determined by the condition of *unbundled* inputs (both primary inputs to the whole system and inputs to individual blocks, formed by the refinement of the system and associated with unbounded delays) and largest possible groups of outputs that can be executed synchronously.

# 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(s). Therefore, PN models are sufficiently expressive in describing a synchronous system. A detailed description of such models is presented in Section 4.

## 2.1 Petri nets and their interpretation in the system desynchronisation context

A Petri net ($PN$) 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 $\mu_0$ is the *initial marking*.

Given a Petri net $N$, the *pre-* and *post-multiset* of a transition $t \in T$ 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 $p$.

There are many semantical aspects and properties available in theory of PNs, but the most essential for our purposes are the following: the semantical notion of a step and the behavioural property of persistency. The former is necessary for expressing the idea of synchronising a group of actions as a bundle (in our interpretation, implemented with the application of a common clock to the corresponding hardware units). The latter is needed for expressing the notion of stability of a bundle of events (i.e., if a clock signal is applied to a set of actions committed to fire in a bundle, this set cannot be changed 'on the fly', otherwise the clock signal can experience hazards).

**Definition 1.** *Step*

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

In a *maximal step semantics,* a PN model evolves through the concurrent firing of transition sets, given the associated external conditions are true.

**Definition 2.** *Persistency*

A Petri net $(N, \mu_0)$ is persistent if for any two different transitions $t_1, t_2$ of $N$ and any reachable marking $\mu$, if $t_1$ and $t_2$ are enabled at $\mu$, then the occurrence of one cannot disable the other.

The notion of *persistency* can be generalised for steps.

In this paper we will consider as the main execution semantics the semantics called *interleaved step semantics* [5], which requires to execute in every marking all possible subsets of enabled transitions if they are not in conflict. For example, for the net shown in Figure 2(b), in the marking which enables both $In1$ and $In2$, there are three possible steps that can be executed, $\{In1\}$, $\{In2\}$, and $\{In1; In2\}$. Each step is usually associated with a separate arc in the reachable state graph. In the following, in order to avoid clutter in notation, we will not normally show the transient steps if the corresponding interleavings are possible. Therefore, as shown in Figure 3, where $In1$ and $In2$ are executed in the interleaved step semantics, we don't depict the arc with the step label $\{In1; In2\}$. We will therefore only show steps where their significance is determined by the use of Maximal step execution.

This paper uses PNs to model systems with actions associated with activations of I/O ports and internal channels. We will not be looking at the computational details of the systems, abstracting them away, and just focusing on the communication aspects. It is therefore convenient to assume that for such a PN 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. Consider a simple example of a system with I/O ports, as in Figure 2(a). Let inputs $I1$ and $I2$ be concurrent to each other. Figure 2(b) denotes the Petri net representation of the input output dependencies of the system which is shown in Figure 2(a).

In the next section we will consider how adding an addition (structural) notion of locality can help us in putting together a convenient modeling framework for analysis of the issues related to system desynchronisation using the Petri net model under the above interpretation of PN transitions.

# 3 Motivation for Desynchronisation Approach based on Localities

The above definition of steps of transitions helps to model the effect of their binding, i.e. synchronisation using clocks. Depending on the semantics of transition step execution, we can formulate different levels of synchronicity, from global, i.e. fully synchronous, down to fully asynchronous. However, we also need some criteria which would impose a particular form of execution policy on the Petri net model of the system. For example, from the point of view of performance we may want to change the policy of (global) maximum step firing parallelism, because such a semantics assumes that the clock signal is only activated after all the events in the previous step have been executed. This leads to the operation with a worst case delay. From the system's viewpoint

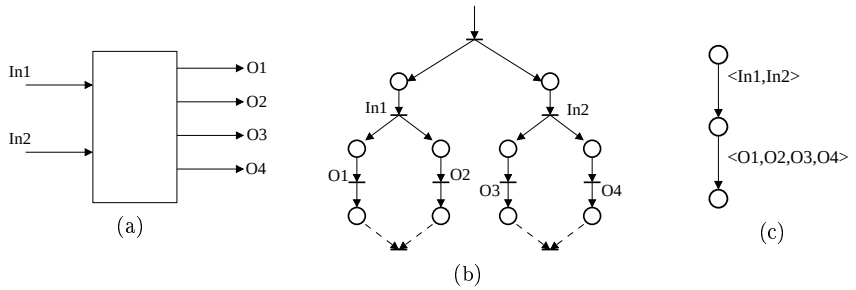



Figure 2: Block and PN representation of a synchronous system

this also implies the problem of distributing global clock, working with global interconnects etc, i.e. all the reasons of why people need to look for ways of moving to GALS systems.

We associate globally synchronous paradigm with maximal firing semantics. A state graph depicting such semantics for the above PN example is presented in Figure 2(c). Suppose we would like to desynchronise the system into a GALS version, for example due to the fact that the variation of delays between $In1$ and $In2$, as well as between $O1$ and $O3$, etc. is large and the global clocking is time-inefficient.

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 3. But the output signals are fired in bundles or *maximal output steps* (or Max-O step semantics, see Section 4). Since the input signals cannot be scheduled to arrive at known instants, persistency property cannot be guaranteed. 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 $s_2$. Namely, 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.

Therefore, if a bundle is altered on the fly, such as $\{O1, O2, O3, O4\}$ to $\{O1, O2\}$ in the above example, it leads to the violation of persistency of a bundle. In other words, events cannot be removed or added into steps as a result of some concurrent action, because it would give rise to unstable bundles.

In order to avoid this situation, the system is not made to follow Maximal Output semantics globally. If it is possible to partition the system in such a way that none of the input transitions and output steps 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* and hence, a way of desynchronisation which satisfies the properties of persistent clocking. In order to obtain a correct implementation

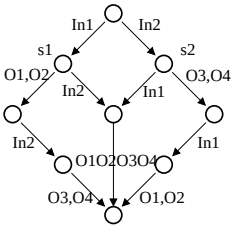



Figure 3: Unbundled out-of-order inputs system model

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 semantics in each locality, aid 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. 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 semantics* and *standard (interleaved step) semantics*. The reason for obtaining such equivalences is to use PNs that are behaviourally equivalent under both semantics due to the feasibility of verification and synthesis using standard tools. Hence, the models 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 can done with the help of theory of Processes, which was introduced in [6]. Avoiding formalisation here we only present the intuitive idea of such a behavioural equivalence.

Let $\Sigma = \{P, T, F, \mu\}$ be a Petri net model. Let $PN_{STD}$ be the prefix of the unfolding of $\Sigma$ under standard semantics and $PN_{max}$ be the prefix of the unfolding of $\Sigma$ under the Max-O semantics.

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

To prevent the Max-O semantic from having additional events which are not permitted by the standard semantics, every process of $PN_{max}$ semantics must be a prefix of some process of $PN_{STD}$.

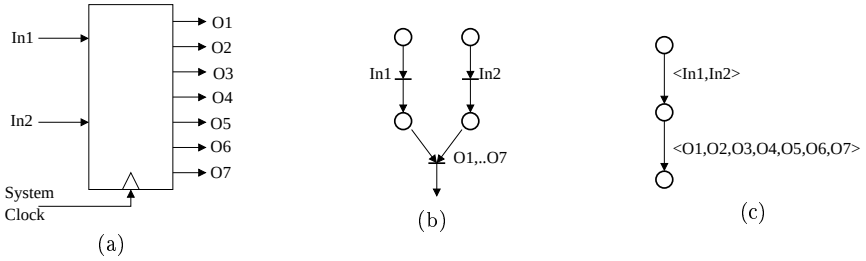



Figure 4: Synchronous block PN and SG model for desynchronisation

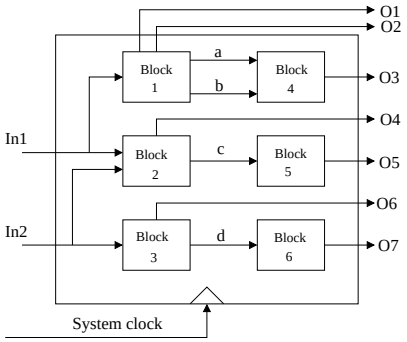



Figure 5: The synchronous system

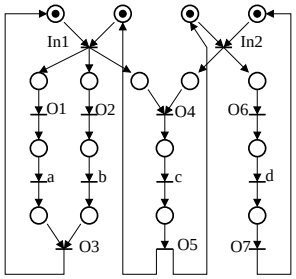
# 4 Synchronous model description

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

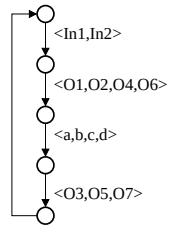
Figure. 4(a) shows a synchronous system. There are two inputs $In1$ and $In2$ to the block and seven outputs, namely, $O1, O2, O3, O4, O5, O6$ and $O7$ from the block. The system clock is used to clock the whole system globally. The PN model specification of such a system is shown in Figure 4(b) (for simplicity, we slightly abuse the labelling convention of our labeled Petri nets, and assign a set of concurrently enabled outputs to one transition instead of using seven separate transitions). The state representation of the maximal firing semantics in a globally synchronous environment is shown in Figure 4(c).

Suppose the synchronous system is further sub-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 shown in Figure 5. 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 6(a). The state graph of such a system is shown in Figure 6(b).

For the formation of localities and to aid asynchronous communication between the localities some transformations are applied at the PN model level. At




(a) PN representation




(b) Maximal firing semantics

Figure 6: PN model of the synchronous block

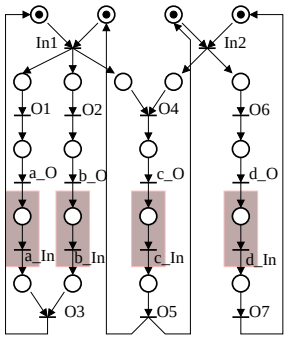



Figure 7: Modified model

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 refined 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. 7. The shaded blocks denote the insertion of signals in the original system.

## 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

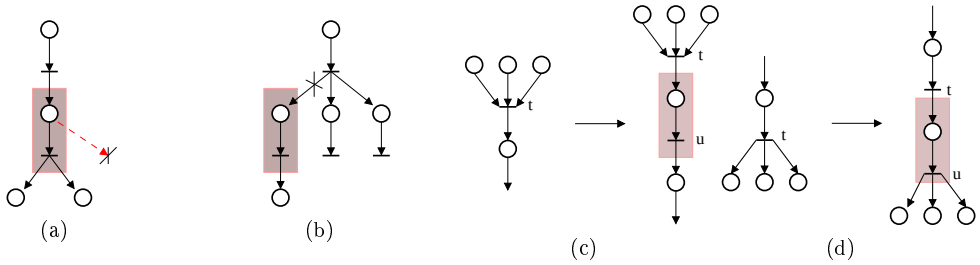



Figure 8: Restrictions on transition splitting

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 3.** *Transition Splitting*

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
- $F^{'} = F \cup (\{t, p\} \cup \{p, u\} \cup \{(u, q) \,|\, q \in t\bullet\}) \backslash \{(t, q) \,|\, q \in t\bullet\}$

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 [[4], 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. 8(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 latter can lead to formation of unnecessary localities due to numerous signal insertions. This is exemplified in Figure. 8(b).

Therefore, the allowable examples for signal insertion after fan-ins and before fan-outs are shown in Figure 8(c) and (d), respectively.

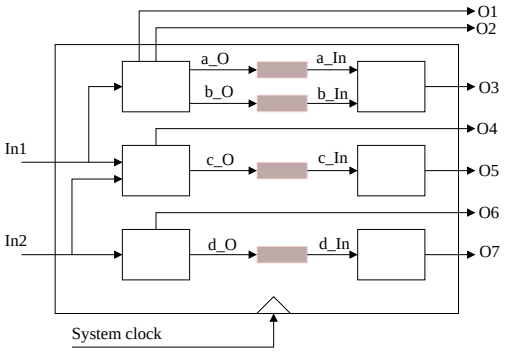



Figure 9: System architecture with storage units

**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 6, 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 buffer. This buffer stores item of data before transferring it across to the next block.

Each of the individual compartments depicted in Figure 9 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, modelled at the $PN$ level by a standard operation of a union of $PN$s, merged on places [10]:

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

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.

## 4.2 A Petri net class

For our initial work , we use a subclass of PNs that follow certain assumptions to aid our desychronisation process. Therefore, our PN models should satisfy the following properties:

***Property 1*** (*Safe*): A PN is 1-safe if for every reachable marking and every place we have $\mu(p) \leq 1$. The PN models used in this work are 1-safe to aid synthesis of the models using existing synthesis tools, as well as to be able to use the theory of localities from [5], which used elementary net structures (it is a known fact that 1-safe nets can be easily modelled by elementary nets).

***Property 2*** (*Choice*): Non-deterministic choice is only allowed between input signals if they are controlled by the environment.

# 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 [5]. 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 and in persistent steps only. This extension is in direct relation to the synchronous behaviour, discussed in the previous sections. Analogies can be drawn between our proposed notion and the notion of Burst-mode circuits, presented in [11, 12], from the point of view of allowing multiple signal changes on each transition and taking into account $I/O$causality. On the other hand, the former can be viewed as a generalisation of Burst-mode circuits. This is because it uses $PN$ as a model of computation which allows the bursts/bundles to be introduced in a flexible way, based on subsets of events bundled in bursts and independent bursts, preserving a level of true concurrency between them.

The transitions in the PN belong to a fixed unique locality. The allocation of localities to the transitions is achieved by partitioning the PT net using a locality mapping function $\gamma$. This means if two transitions return the same value for $\gamma$ 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 $\gamma : T \rightarrow N$ 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.

In general, a net can be partitioned giving rise to the formation of smaller nets that constitute the original graph.

Let $\Sigma = \{P, T, F, \mu_0\}$ be a PN. Then, a partition can lead to the division of the net into $n$ smaller nets which can be denoted by,

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

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 $(P_1 \cap P_2 \cap ....P_n) \neq \emptyset$, $P_i \triangleright \mu_0$ is defined by the following:

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

Depending on the rules applied to the partitioning process, the above general definition can be altered to meet the requirements of a given methodology. The rules applied in our proposed partitioning methodology is presented in 6.

# 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:

Let $\Sigma = \{P, T, F, \mu_0\}$ be a PN. The partitioning $\Sigma = \Sigma_1 \cup \Sigma_2 \cup ....\Sigma_n$, each belong to localities $L_1, L_2...L_n$, respectively, is correct at a marking $\mu$ 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 $\Sigma$. This denoted as,

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

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

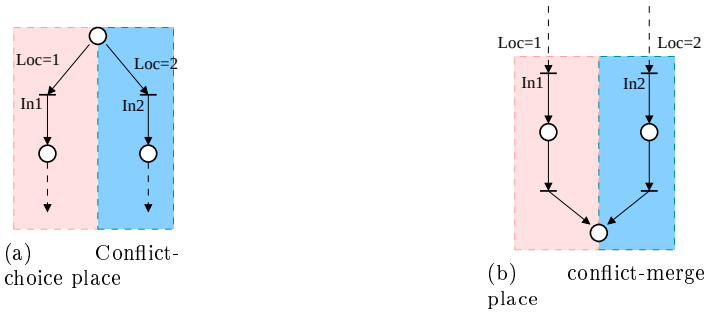



Figure 10: Conflict between transitions

**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 10(a). The net $\Sigma$ is partitioned into $\Sigma_1$ and $\Sigma_2$ 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 $\mu$, 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 10(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 condition holds true:

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

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

then the partitioning is correct.

Therefore, if outputs can disable each other, i.e. they are in conflict, they cannot be in different localities. These outputs must be in the same locality and this conflict should be interpreted as choice which takes place within the locality.

**Note** If there exists a step $\{I1, I2, O1, O2, \}$ enabled under a marking $s$, each action in it is allowed if it is not in conflict with another action in it. If some of the enabled transitions are in conflict and can't fire as a step, the inputs are allowed to be resolved by the environment choice, or within each locality if the conflict is between outputs. In a globally clocked system there are no non-persistent steps because there is no true concurrency (synchronous simultaneous actions). The conflicts in such systems can be resolved statically, in a deterministic way, by scheduling them using some mechanisms such as priorities or cost functions.

As soon as we start desynchronisation, i.e. unbundling, we enter the world where concurrency may introduce hazards due to dynamic conflict resolution or dynamic re-scheduling of steps. This may not be implemented physically using logic blocks with certain min delays (i.e. certain resolution limits).

**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 using the following procedure: 1) For each output transition in the net, identify the set $Out$ of output transitions that are dependant on more than one input transition. 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 shown by the shaded region in Figure 11(b).

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.

Therefore, at the model level, the system depicted in Figure 7 is transformed into the system depicted in Figure 11(b).

# 7 Allocation of Localities

In order to obtain the partition sets of transitions, the information about the locations of each input and output transitions of the PN is required. We derive the localisation of each input and output transitions of the synchronous circuit

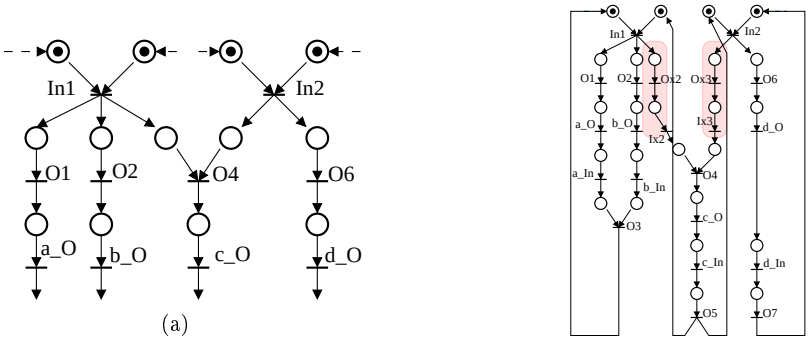



Figure 11: Persistency check

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. Adherence to this property prevents the occurrence of deadlocks, arising due to unavailability of input signals in some localities, since input signals are not shared by the different localities. This locality allocation approach leads us to partitioned blocks denoted by the following definition.

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

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

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 $(P_1 \cap P_2 \cap ....P_n) \neq \emptyset$.

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 by redistributing transitions over localities to avoid locality overloading arising from large input fan outs. In order to allocate localities to the transitions of a system, we require to define some methods which are presented as algorithms in [8, 9].

This algorithm 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 $\Sigma$. The output of the algorithm is a Petri net model of the synchronous system $\Sigma$, with locality information added to each transition in the model.

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. The partitioning algorithm presented in [8], also satisfies the correctness criterion, namely, *conflict-resolution.* This is because the algorithm places all the signals, in con-

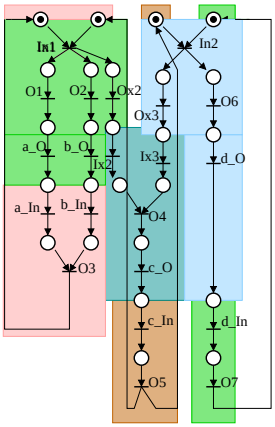



Figure 12: Partitioned model

flict, in the same locality. The application of the algorithm presented in [8] to the system, shown in Fig. 11(a), leads to a partitioned model shown in Fig. 12.

Each of the localities formed as above can either be implemented fully asynchronously or have its internal clock in order to control all the internal computations and generation of output signals. For this appropriate wrappers can be built that will generate local clock enables. These wrappers can be synthesised using existing PN based synthesis tools. More details on synthesis of wrappers can be found in [3, 8].

# 8 Conclusion

This paper addressed the problem of synthesising a GALS system by a desynchronisation methodology which employed PN as its model of abstraction. The granularity of desynchronised systems, thus constructed using PNs, is smaller than the ones obtained from the previous method [3] and thus is easier to automate and apply even for large complex circuits. The GALS system can be obtained by applying the theory of localities to a synchronous system model preserving the synchronous properties of the input output signals.

This work presented a desynchronisation approach with a relatively clear route to automated synthesis, while preserving the IO behaviour of the synchronous systems.

**Future Work** The proposed methodology needs to be automated to reduce design time and designer intervention. The locality allocation can be further optimised to meet various criteria, e.g. to minimise interconnection between localities and to increase the component speed. It could also be possible to apply other ways of unbundling transitions (e.g. not necessarily consider all inputs to be desynchronised) and thus obtain different conditions for persistent steps.

## 9 Acknowlegement

We thank the anonymous referees for their helpful comments. This work has been supported by EPSRC.

## References

[1] C. Carloni, K. McMillan, and A. Sangiovanni-Vincentelli. The Theory of Latency Insensitive Design. *IEEE Trans.* on Computer-Aided Design of Integrated Circuits and Systems, 20(9):1059-1076, 9 2001.

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

[3] S. Dasgupta, D. Potop-Butucaru, B. Caillaud, A. Yakovlev "Moving from Weakly Endochronous Systems to Delay-Insensitive Circuits".In *Proc.* Workshop on FMGALS 2005, ENTCS, Vol. 146, No.2, Jan 2006, pp 81-103.

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

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

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

[7] J. Desel, J. Esparza, "Free Choice Petri Nets". Cambridge University Press, Sept 2005.

[8] S. Dasgupta, A. Yakovlev, "Desynchronisation Technique using Petri nets". Tech. Report Series NCL-EECE-MSD-TR-2007-124, MSD Group, School of EECE, Newcastle Univ., Nov 2007.

[9] S. Dasgupta, "Formal Design and Synthesis of GALS Architectures". PhD thesis, Newcastle Univ, 2008.

[10] G. Berthelot, "Checking Properties of nets using Transformations". Advances in Petri Nets 1985. LNCS, vol. 222. Springer-Verlag, Berlin, 1940.

[11] Steven M. Nowick, "Automatic Synthesis of Burst-Mode Asynchronous Controllers". PhD thesis, Stanford University, Department of Computer Science, 1993.

[12] Steven M. Nowick and David L. Dill, "Synthesis of asynchronous state machines using a local clock". In *Proc.* International Conf. Computer Design (ICCD), pages 192-197.IEEE Computer Society Press, October 1991.