
School of Electrical, Electronic & Computer Engineering



**Virtual Self-timed Block Design using Coloured
Petri Nets**

Yuan Chen, Fei Xia, Delong Shang, Alex Yakovlev

**Technical Report Series
NCL-EECE-MSD-TR-2008-134**

August 2008

Contact: yuan.chen1@ncl.ac.uk

Partially Supported by EPSRC grant EP/E044662/1 and EP/C512812/1

NCL-EECE-MSD-TR-2008-134
Copyright c 2008 Newcastle University

School of Electrical, Electronic & Computer Engineering
Merz Court, Newcastle University
Newcastle upon Tyne, NE1 7RU
UK

<http://async.org.uk>

Virtual Self-timed Block Design using Coloured Petri Nets

Yuan Chen, Fei Xia, Delong Shang, Alex Yakovlev,

August 2008

ABSTRACT

In order to increase the power efficiency of IP cores in an SoC, a Self-timed Event Processor (STEP) is designed in this paper to provide power management and event handling for each IP core in a frame of Virtual Self-timed Block (VSB). Following Model Based Design (MBD) method, this paper presents the specification, analysis and verification of a VSB design in detail.

1. INTRODUCTION

With the fast development of semiconductor technologies, all components of a computer system can now be integrated into a single chip forming a System on Chip (SoC). Previous research about *GALS* (Globally Asynchronous Locally Synchronous) [1] has successfully solved asynchronous communication using different types of *ACM* (Asynchronous Communication Mechanism) [2] among heterogeneous integrated IP cores. Large transistor integration also made complex electronic devices energy hungry and the confliction between high power consumption in IP cores and the limited energy that can be supplied by onboard battery became the main restriction to system performance. Although many IP core designs have integrated various low power technologies so as to enable an IP core to be operated in different *modes*, the power/latency cost brought by mode switching transitions [3] requires system-level power control in an SoC context.

Although the GALS architecture has great potential in power saving, system level power management has not been considered by previous GALS design. Without a global clock system, an SoC built in GALS architecture can easily power on/off an IP core or switch it to another mode without interfering the clocks of other clock domains. There have been many studies about system latency and throughput in a GALS based SoC, but how to introduce power control/management into such architectures so as to achieve system performance in low power is still a relatively unexplored area.

Higher integration changes embedded software design as well. When more and more IP cores have been integrated into one chip, task execution in an IP core becomes *nondeterministic* and *concurrent*. In other words, the start moment of a task's execution is unpredictable, and it is highly possible that two or more tasks become ready for execution simultaneously. The concurrency in task execution brings competition for *resources*, like limited battery energy, finite memory space or communication bandwidth, etc. Nondeterminism in task execution requires faster or more real-time resource allocation.

Event driven programming can be used in on-chip software design. An *event* is modelled as something happening or happened and should be responded to by a task. It may mean the availability of request signal or data, or idleness of input/output ports, or enough energy in the battery, depending on different implementations. In this case, resource allocation in an SoC amounts to event handling. A task in an IP core can be executed only after its corresponding event has been handled by the core. Therefore, well-designed dedicated event handlers, which can quickly and properly respond to incoming events, have great importance to system performance in both latency and power.

When on chip nondeterminism and concurrency are taken into consideration, asynchronous circuits have certain advantages in event handler design. Without clock control, an asynchronous event handler can respond to new incoming events without delay, the probability of metastability can be greatly reduced. Furthermore no power is wasted in an asynchronous handler when no state change happens in the system.

Therefore, an asynchronous coprocessor can be designed and used in a GALS based SoC. This coprocessor will not only do asynchronous/synchronous data transform and asynchronous communication for its IP core (thus replacing the traditional wrapper), but also provide event handling and power management for the core. With the coprocessor, every IP core can work as an event driven domain in a highly nondeterministic and concurrent SoC environment. This coprocessor is called as *Self-Timed Event Processor* (or *STEP* in short) and the combination of the coprocessor and its IP will work as a “*Virtual*” *Self-timed Block* (or *VSB* in short) in the GALS architecture [4].

When multiple functions are integrated into a VSB, the VSB design becomes complicated because it should consider not only the design of all function modules, but also the cooperation of these modules in terms of power and latency. The *Model Based Design* (MBD) method, adapted from control system design, has been used in our research as the design flow of a VSB. As power control is the essential function provided by a STEP to its corresponding IP core, the identification of a proper control policy for power is the first step of our design. Stochastic models are used for this identification since these mathematical models are widely used in previous research and highly representative of actual systems. This study is reported in [5] where a policy called Accumulation & Fire (A&F) shows high efficiency and easy realization in power control.

When the control policy has been identified, we focused on the specification, analysis and verification of the VSB design in this paper. The specification of a VSB is given in Section 2. Following this specification, the design of a VSB will be carried out by modelling its behaviour from abstract execution among basic modules in a top level model to detailed processing within every module in low level models. Coloured Petri Nets (CPN) is used for the modelling design of a VSB because its strong representations of systems in which concurrency, communication, and synchronisation play a major role [6]. This modelling design is described in Section 3. And this section also includes the analysis and verification of the VSB design in CPN where concurrent executions in a VSB are highlighted. And conclusion and future work are given in Section 4.

2. VSB Specification

According to the stochastic models given in [5], an IP core can be represented by two factors: Tasks and Modes. Although an infinite length of task queue is assumed in the stochastic model analysis, an IP core can only provide a finite number of task services. A mode defines the power dissipation as well as the processing speed in the core. With such knowledge, we can give a basic functional specification of a VSB in Figure 1.

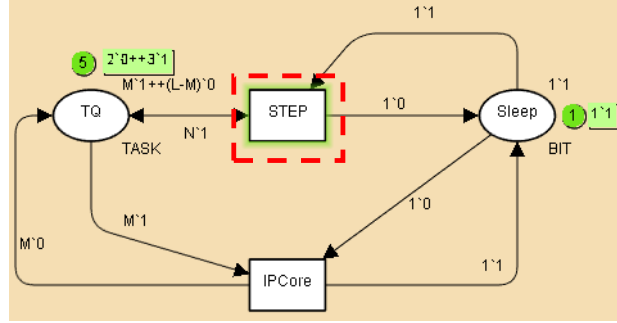


Figure 1: The Specification of a VSB

Although Petri Nets (PN) were used by previous studies to specify different types of ACM [7], we prefer to use high level Coloured Petri Nets (CPN) to give the specification of a VSB in a more concise way. Similarly as the basic Petri Nets (PN), CPN uses *places* to indicate states of the modelled system by means of ellipses (or circles) and uses *transitions* to indicate operations in the system by means of rectangles. For example two places *TQ* and *Sleep*, and two transitions *STEP* and *IPCore* are used in Figure 1. Places and transitions are connected by a set of directed arrows, which are called *arcs*.

A place in a CPN/PN model is used to hold *tokens*, which are represented by the small dots next to each place. An arbitrary distribution of tokens in places is called a *marking*. Different from PN, each token in CPN is attached with some data value (called *token colour*). The data value may be of an arbitrarily complex type. For a given place, all tokens must share the same colour. This colour is called the *colour set* of the place which is written in the right bottom corner of the place.

In CPN, two operators $++$ and $`$ are used for the construction of a multi-set consisting of token colours. The infix operator $`$ takes a nonnegative integer to specify the number of appearances of the element provided as the right argument. The $++$ takes two multi-sets as arguments and returns their union (sum). For example, the tokens $2'0++3'1$ in the *TQ* place describe two tokens with colours (values) of $'0'$ and three tokens with value of $'1'$ respectively (In this paper, a pair of quotations $'$ will be used to quote a colour value when it may be confused with the token number).

The VSB specified in Figure 1 presents an on-off IP core under A&F power control. Two colours have been declared in this figure. Colour *BIT* is declared to describe binary information and have only values $'0'$ and $'1'$. It is the set colour of the place *Sleep*. Initially a $'1'$ token is given to the *Sleep* place (The initial token of a place is described in the upper right side of the place), which indicates the IP core is in its off mode.

The other colour declared in the figure is called *TASK*, which represents tasks that can be executed in the IP core. In this high-level specification, all tasks are taken as identical and the *TASK* colour is declared as *BIT* colour whose token value represents

whether the task is ready for processing (value '1') or not (value '0'). The TASK colour is the set colour of the TQ place (means Task Queue). Since an IP core can only perform a finite number of tasks, two integer constants L and M represent the total number of tasks of the core and the number of valid tasks of the core respectively. Therefore, the initial marking of $M \cdot 1 + (L-M) \cdot 0$ in the TQ place indicates initially M out of L tasks in the core are ready for processing. When M and L are specified as 3 and 5 respectively, the token held in the TQ becomes $2 \cdot 0 + 3 \cdot 1$.

A transition is *enabled* if and only if each of its input places contains at least the number of tokens prescribed by the expression of the corresponding input arc. When a transition is enabled, the corresponding move may take place, which is called the *occurrence* of the transition. As a consequence, tokens from the input places will be removed from the input places and added to the output places after the execution of an occurrence.

The STEP transition is used to describe the power/task control given to the IP core, i.e. the actions performed by the STEP. The expression in the arc from the Sleep place to the STEP transition is written as $1 \cdot 1$, which means this transition is only enabled when the token value in the Sleep place is '1'. It indicates the power control is given to the IP core only when it is in its off mode. The TQ place and the STEP transition are connected by a *double-headed arc*. A double headed arc is shorthand for two directed arcs in opposite directions between two nodes which have the same arc expression. The integer constant N ($N \leq L$) is used to represent the accumulation limit of A&F policy which is implemented in the STEP. Therefore, when N is specified to 2, the STEP transition is enabled (which is highlighted by a dotted rectangle) and its occurrence will change the token colour in the Sleep place to '0'. This occurrence indicates the IP core is activated when there are at least N tasks accumulated.

When the token in the Sleep place becomes '0', the IPCore transition is enabled and its occurrence will first reset all '1' tokens in the TQ place to '0', and then toggle the token in the Sleep place to '1'. This occurrence describes the processing of all ready tasks in the activated IP core, and the shutting down of the core afterwards when no tasks are ready.

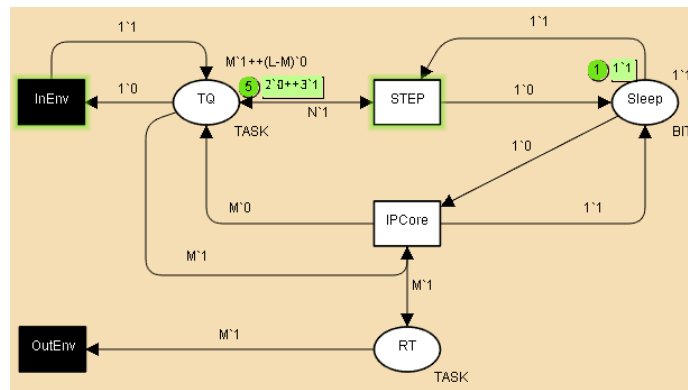


Figure 2: New Specification of a VSB

Although the model in Figure 1 specifies the basic function in a STEP and an IP core in a VSB, it only represents an isolated computation block without interactions with its environment. In Figure 2, we present the relationship between a VSB and its SoC environment. Transitions *InEnv* and *OutEnv* represent the SoC environment and dark shade is used in these two places so as to differentiate them from other places/transitions which represent a VSB. The occurrence of the *InEnv* transition will

responded to by the STEP. Similarly, the place *TQ* represents the status of all tasks that need to be executed in the IP core. An initial token *L'0* is attached to the TQ place. The initial value '0' indicates that no task is ready for execution and all tasks are waiting for their corresponding events.

The transition *EH* (means Event Handler) is used to represent the event handling execution in a STEP. It is enabled when there is at least one '1' EVENT token in the EQ place and one '0' TASK token in the TQ place. The occurrence of this transition will remove one EVENT token from the EQ place, indicating one incoming event has been responded to by the STEP. At the same time, a '0' token in the TQ place is replaced by a '1' token, indicating one more task is ready for execution. When a VSB is designed for data processing, the possible asynchronous/synchronous data transform which used to be performed in an asynchronous wrapper is also included by the occurrence of the EH transition.

The EH transition will be enabled again until no more '0' tokens can be found in the TQ place. In this case, all tasks are ready for execution and further responding to the incoming events in the event handler cannot change the status of the task queue until some tasks are completed in the IP core.

The power control in a STEP is represented by the transition *PM* (means Power Management) in the CPN model. According to A&F policy, the PM transition is enabled only when there are at least *N* '1' tokens in the TQ place. The occurrence of the PM transition will toggle the token value in the *Sleep* place from '1' to '0', which indicates the wakeup processing in the IP core. It will also add one '1' token to the *load* place, which means the IP core will load a new task for execution when the wakeup processing is completed. According to the arc from the Sleep place to the PM transition, the PM transition is only enabled when the token value in the Sleep place is '1'. It is such designed because A&F is useful only when the IP core is sleeping. Disabling the execution in the PM after the IP core is activated will further reduce the power dissipation in a VSB.

The nondeterministic incoming of events make it highly possible that several tasks can become ready before the IP core is woken up. In this case, some scheduling execution is necessary to select one task from all the ready ones for the IP core's execution. Although task scheduling is provided by many IP cores, a *task manager* is designed as a component of a STEP to provide scheduling service. It is not only because hardware scheduling can be many times faster than software scheduling, but also because this design means both task scheduling in the STEP and wakeup processing in the IP core are carried out in parallel for better system latency and power dissipation.

In Figure 3, the transition *TM* is used to represent the execution in the task manager. This transition is enabled when there are more than one TASK '1' token in the TQ place and one BIT '1' token in the load place. Because all tasks are treated as identical in the top level model, the occurrence of this transition will add one TASK '1' token to the *NTask* (means New Task) place, indicating a randomly chosen task is loaded to the IP core. At the same time, one TASK '0' token is added to the TQ place indicating the chosen task in the NTask place has already progressed to the next step.

When one TASK token is available in the NTask place, the *Execution* transition is enabled and the occurrence of this transition indicates the execution of the current task in the IP core and it will add one TASK '1' token to the *RT* (means Result Task)

place. Generally speaking, the completion of one task execution will either release some system resources like I/O port or data bus, or generate some new data or signals. In most cases, the released resources or generated data in one VSB can work as a new event to trigger some other task in the SoC, probably in other VSBs. Therefore, a new component of STEP, named as *output controller*, is needed to prepare a new event when the execution of the current task is completed. In the top level model, the transition *OutCt* is used to represent the executions in the output controller. Its occurrence will add one EVENT '1' token to the *OEQ* (means Output Event Queue) place where events will be sent to the SoC environment. The occurrence of *OutCt* transition will also add one token to the Load place which enables the TM transition to choose another task for the IP core's execution. Note that this cycle implies an assumption of fully sequential execution in the IP core, but can easily be extended to situations when the IP core can handle execution concurrency.

When a new BIT token in the load place is generated but finding no TASK '1' token available in the TQ place, the *Shutdown* transition will be enabled since ready tasks have all been executed. Its occurrence will toggle the token in the Sleep place to '1' which means the IP core has been shut down.

3.1.2 Environmental Set Description

All places and transitions introduced so far construct the top level model of a VSB. In order to check the behaviour of the model and verify the properties, some extra places and transitions are added so as to simulate the asynchronous environment of an SoC. Therefore, the entire top level model can represent an enclosed system. These places and transitions are highlighted by dark shade so as to differentiate from their counterparts describing a VSB.

Transition *Env* (means environment) is used to describe event transferring in an SoC. The occurrence of the *Env* transition indicates the event generated from the current VSB is transferred to its SoC environment. It will be used in some other VSB to enable some task's execution. And the execution of the task will also generate some new events. Eventually this relay of "event transfer – task execution – event transfer" may generate some event to enable a task in the current VSB in turn again, but the time span cost in this relay is nondeterministic.

A CPN function $P()$ and a transition *Env1* are used to model the nondeterministic characteristic of event relays. The CPN function $P()$ is defined as:

fun $P()$ = poisson (2.5)

|

This function uses the random number generator *poisson* provided by *CPN Tools* [8] to generate a random integer number which follows Poisson distribution. The number 2.5 in the function declaration is the rate λ in the Poisson distribution and can be changed according to the feature of the implementation environment.

The expression of the arc from the transition *Env* to the place *EQ* is written as "if $P() > 1$ then 1`1 else 1`0". Therefore, the value of the token generated by the *Env* transition's occurrence depends on the result of $P()$ function. If the function result is less than 1, an EVENT '0' token is added to the *EQ* place indicating the event relay is not completed since an EVENT '0' token can not enable the *EH* transition. Instead, it will enable the *Env1* transition and the latter's occurrence will add one EVENT '0' token to the *OEQ* place. When an EVENT type variable *event* is used in the expression of arc from the *OEQ* place to the *Env* transition, the latter transition will be

enabled no matter what value the token in the OEQ has. Therefore, the token loop in OEQ-Env-EQ-Env1-OEQ represents the event relay in the SoC environment and the moment to jump out of the loop depends on the random result generated by the P() function. When the P() result becomes bigger than 1, an EVENT '1' token added to the EQ place will enable occurrences in the current VSB. We use this token loop and its non-deterministic exit to model the non-deterministic nature of event distribution within an SoC and its effect on any single VSB because at this stage we have no application-specific system level information.

3.1.3 Simulation

CPN Tools [9] is the computer aid software for CPN modelling and analysis. This software provides easy editing, simulation, state space analysis, and performance analysis of CPN models. In this section, we use CPN Tools to build CPN models of a VSB and simulation and state space checking are used for analysis and error checking. Initially we set one '0' token to the Sleep place suggesting the IP core is inactive. Constant L is set to 5, and five '0' TASK tokens are given to the TQ place indicating none of the five tasks are ready for execution. Two EVENT '1' tokens are set to the EQ place showing two incoming events are waiting to be responded to by the STEP. Even when the two events are responded to, the IP core cannot be woken up since N is set to 3. At the same time, two EVENT '0' tokens are added to the OEQ place indicating two events are relaying in the environment. Therefore, the activation of the IP core needs the arrival of at least one EVENT '1' token to be added to the EQ.

With the initial marking, we can observe the behaviour of the top level model using the simulation tool by CPN Tools. All concurrent executions in the system are shown with simulation steps. Sometimes more than one transition is enabled in one step. This simultaneous transition enabling describes the concurrent processing in the corresponding system. The occurrence sequence of multi enabled transitions is random, which represents the nondeterminism in their modelled operations. Since different occurrence sequences may bring different markings, CPN models are highly representative for the behaviour of a system under nondeterministic and concurrency.

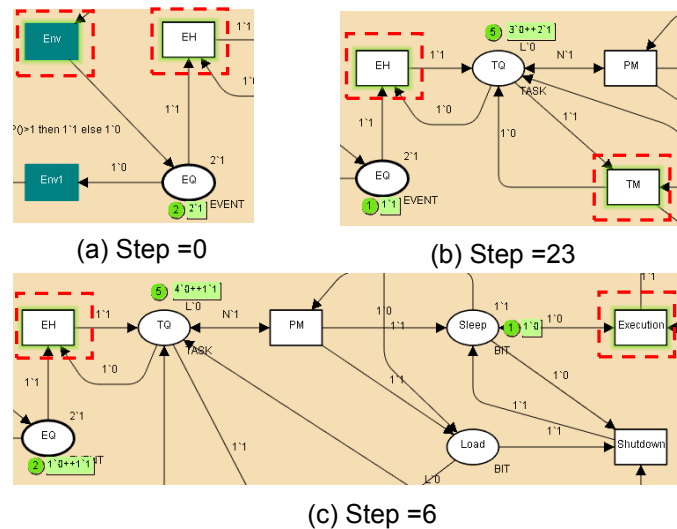


Figure 4: The simulation result of the CPN model

Some typical concurrent executions are shown in Figure 4 where every enabled transition is highlighted by a dotted rectangle. Figure 4(a) is about the concurrent

executions between the current VSB (The EH transition) and the environment (The Env transition). Figure 4(b) indicates the concurrent executions among different components of a STEP (The EH and TM transitions). The concurrent executions between a STEP (The EH transition) and its IP core (The Execution transition) are shown in Figure 4(c).

The simulation can also help users to correct errors in their model design. For example, one double-headed arc is used to connect the Shutdown transition and the TQ place because all '0' TASK tokens will be checked but not consumed when the Shutdown transition occurs. However, designers might miss the arc directing from the Shutdown transition to the TQ place (Figure 5(a)) and the consumption of TASK '0' tokens in the occurrence of Shutdown transition will make further enabling in the EH transition impossible.

If simulation is carried out with the incorrect top level model, it will stop after a certain number of steps because in that case no more transitions will be enabled (called dead marking or dead lock). Therefore, a dead marking in the simulation is used to detect an error. However, because of the randomness brought by the function $P()$, this simulation termination may not happen within a few steps. Five simulations have been carried out when the model has the given error. In these simulations, the CPN Tools took 103, 202, 159, 394, 941 steps respectively to reach the dead marking. Fewer designers will take thousands of steps in the simulation and if they quit in the first several hundreds steps when 941 steps are needed to detect the error, the error will be hidden in the design.

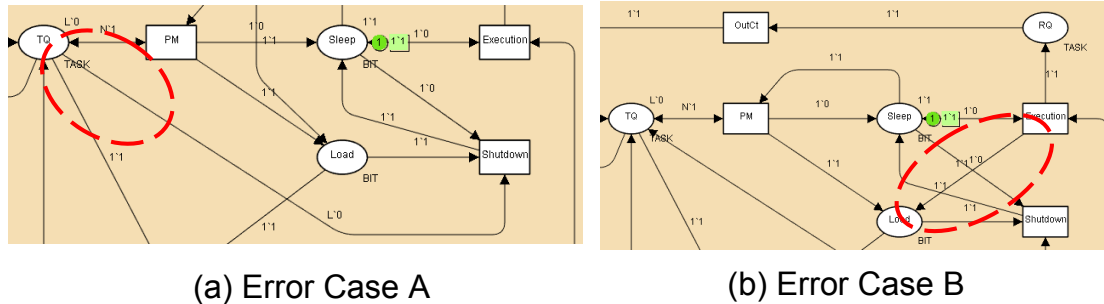


Figure 5: Two Possible Errors in Top Level Model Design

Furthermore, the position of some other errors cannot be detected based on simulation termination. In Figure 5(b), the arc directing from the OutCt transition to the Load place in Figure 3 is changed by the arc leading from the Execution transition. This comes from the initial thought that a load requirement should be given as soon as the execution of the current task is complete. However, if only TASK '0' tokens are available in the TQ place, the Shutdown transition will be enabled simultaneously with the enabling of the OutCt transition. If the Shutdown transition occurs first, the OutCt transition is disabled which means no more executions for new events preparation will be carried out because the IP core is inactive and the corresponding event may be missed or duplicate sent. Unfortunately, this error cannot be found by the method of simulation termination because the concurrent enabling of both OutCt and Shutdown transitions will not make the model reach dead marking. And errors like this will be more easily missing by the designers. Since simulations cannot guarantee the finding of any particular malfunction, we need other more reliable function tool to prove the correctness of the model.

3.1.4 State Space Checking

The state space tool provided by CPN Tools will check all possible executions of the model and present the properties of the full state spaces of the model in a statistical report. Therefore, state space checking has been used on the top level model (as well as other CPN models in the following sections) for error checking and property verification.

When state space checking is done on a top level containing the error marked in Figure 5, the corresponding report is given in Table 1.

Table 1: State Space Report for CPN Model with Error A

Statistics			
Occurrence Graph		Scc Graph	
Nodes:	149	Nodes:	35
Arcs:	387	Arcs:	80
Secs:	0	Secs:	0
Status:	Full		
Boundedness Properties			
Best Integers Bounds	Upper	Lower	
TOP'EQ 1	4	0	
TOP'Load 1	1	0	
TOP'NTask 1	1	0	
TOP'OEQ 1	4	0	
TOP'RT 1	1	0	
TOP'Sleep 1	1	1	
TOP'TQ 1	5	5	
Best Upper Multi-set Bounds			
TOP'EQ 1	2`0++4`1	TOP'Load 1	1`1
TOP'NTask 1	1`1	TOP'OEQ 1	3`0++4`1
TOP'RT 1	1`1	TOP'Sleep 1	1`0++1`1
TOP'TQ 1	5`0++4`1		
Best Lower Multi-set Bounds			
TOP'EQ 1	empty	TOP'Load 1	empty
TOP'NTask 1	empty	TOP'OEQ 1	empty
TOP'RQ 1	empty	TOP'Sleep 1	empty
TOP'TQ 1	1`0		
Home Properties			
Home Markings: [109]			
Liveness Properties			
Dead Markings: [109]			
Dead Transitions Instances: None			
Live Transitions Instances: None			

A full state space is a directed graph, where there is a node for each reachable marking and an arc for each occurring binding element. Therefore, the first part of the state space report is *state space statistics* telling how large the state space is. The next two parts of the state space report contain information about the *boundedness properties*. The boundedness properties tell how many (and which) tokens a place may hold. The *best upper integer bounds* for a place specify the maximal number of tokens that can reside on each place in any reachable marking. For the place EQ, it

holds four EVENT '1' or two EVENT '0' tokens at most. The *best lower integer bounds* for a place specify the minimal number of tokens that can reside on each place in any reachable marking.

Following the boundedness properties are the *home properties*, which are about the reachable property of markings and transitions in the model. A *home marking* is a marking which can be reached from any reachable marking. The report of the example model shows one home marking exists whose index is 109. A *dead marking* is a marking which no binding elements are enabled. The current report shows the home marking is a dead marking.

A transition is *live* if from any reachable marking we can always find an occurrence sequence containing the transition. A transition is *dead* if there is no reachable marking in which it is enabled. The report shows that all transitions in the model are neither live nor dead. In other words, they can be reached from some initial markings but cannot from others.

The information given in the report can help users have a more specific and thorough understanding of their models so as to correct errors which cannot be easily found by simulation and improve the performance of the corresponding systems.

Because no dead transition exists in the model, it means all transitions can be enabled at least once. However the occurrence of some transition causes an abnormal marking which makes no more transitions can be enabled since then. Since the dead marking is a home marking, it means this abnormal marking will always happen no matter what occurrence sequences may happen. This analysis can help the designer finally find the error in the arc between Shutdown transition and the TQ place. When the error is removed from the model, the corresponding state space report is given in Table 2(all identical items with the report in Table 1 are omitted).

Table 2: State Space Report for a Correct CPN Model

Statistics	

Occurrence Graph	Scc Graph
Nodes: 177	Nodes: 1
Arcs: 471	Arcs: 0
Secs: 1	Secs: 0
Status: Full	
Boundedness Properties	

...	
Home Properties	

Home Markings: All	
Liveness Properties	

Dead Markings: None	
Dead Transitions Instances: None	
Live Transitions Instances: All	

When Error B in Figure 5 happens, the corresponding report is shown below (all identical items with the correct model report are omitted):

Table 3: State Space Report for CPN Model with Error B

Statistics

Occurrence Graph	Scc Graph
Nodes: 266	Nodes: 1
Arcs: 757	Arcs: 0
Secs: 0	Secs: 0
Status: Full	
Best Upper Multi-set Bounds	
TOP'NTask 1	1'1 (Identical to the correct report)
TOP'RQ 1	4'1

No dead marking in the model means the token flow can continue forever in the simulation and the designer cannot use the simulation termination method to find the error. When the Best Upper Multi-set Bounds are checked, it shows the RQ places can hold at most 4 tokens while only one token can be held in the NTask place. When a new task suggested by the '1' token is loaded in the IP core, the correct operation in the IP core should first do the new event preparation based on the completed task, and then try to load a new task. Therefore, the multi tokens should not happen in the RQ place. Based on this analysis, the designer can easily find the error in the RQ place.

3.1.5 The Extension of Top Level Model

In this section, we present a top level CPN model of a VSB (including a STEP and an IP core). Although abstract, this model clearly presents the basic architecture and execution flow in a VSB. The integration of A&F policy for power control is also specified in the model. Both simulation and state space function tools provided by CPN Tools are used to check the correctness of the model.

The abstract declaration of both EVENT and TASK colours make the top level model maintain robustness when events are specified by different concepts in various implementations. However, it also prevents representing the execution details in the model. For example, the top level model gives no information about how incoming events are handled in the EH and how scheduling is carried out in the TM. Refining to lower level models is needed to clarify the design. In these models EVENT and TASK colours are re-declared. In the following sections, four CPN models are designed, each of which focuses on one component of a STEP and works as the extension of the top level CPN model.

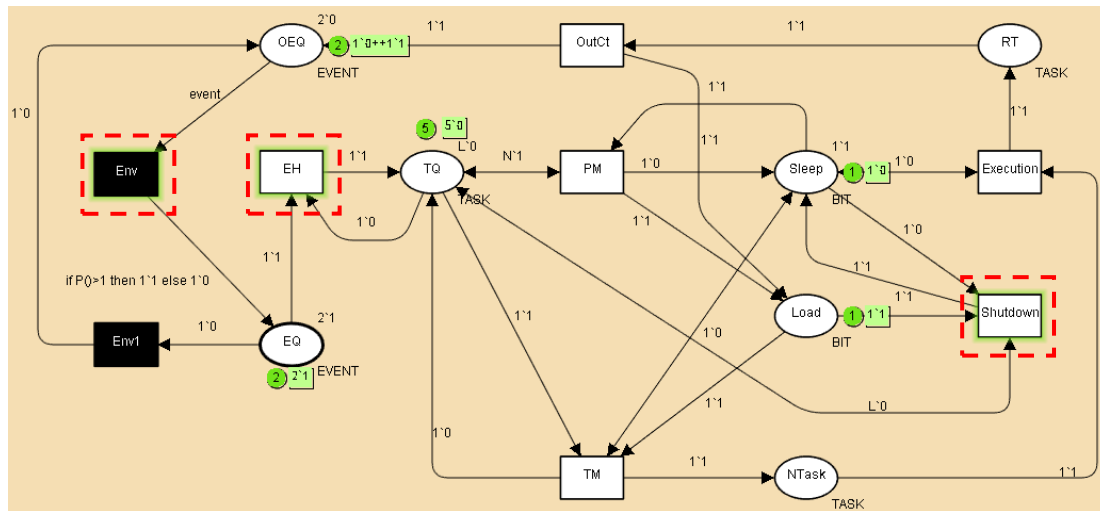


Figure 6: Possible Hazards Brought by Concurrent Executions

During simulation, the top level model indicates all possible concurrent executions between different parts of a VSB. Some concurrency can bring parallel processing so

as to reduce system latency. However, others may cause hazards which may affect a VSB's performance. For example, Figure 6 presents the case when both the Shutdown and EH transitions are concurrently enabled. It indicates the case when the EH component is handling incoming events while the IP core is shutting down. If the EH transition occurs first, one TASK '0' token in the TQ place changes to '1' and the Shutdown transition is disabled. This occurrence sequence indicates the case when a shutdown process is interrupted by a new incoming event. However, an interruption in a mode switching transition may cause data loss or more serious consequences in most IP core implementations. Therefore, a new component named as *Interface* will be designed to avoid interruptions when a shutdown process is ongoing. This component, together with the event handler (EH), the power manager (PM), the task manager (TM), the output controller (OutCt), constitutes the basic structure of a STEP.

3.2 The CPN Model of the Event Handler Component

In this section, we try to model and specify executions in the event handler component of a STEP. In the top level model, every occurrence of the EH transition can only consume one EVENT token in the EQ place, which means all incoming events from different communication Channels must wait in a queue to be responded by the STEP and therefore arbiter(s) become indispensable when events may arrive simultaneously. The direct use of arbiters will bring cost in both power dissipation and latency. A better solution should enable multiple events to be handled in parallel.

Moreover, the occurrence of the EH transition in the top level model will update the value of one task token from '0' to '1', which means every incoming event will make one corresponding task ready for execution. However, this is not true in the implementation of STEPs with multiple input Channels. Although events from the same Channel always indicate different tasks in an IP core (otherwise two events can be taken as one with double amount of information), events from different Channels are highly possible to indicate the execution of the same task (but with different information like data for operation). In this case, the consumption of one event token may not change the value of its corresponding task token if the latter's value has been updated by one previous event with the same task indication.

3.2.1 A Matrix Structure of Event Handler

When we take the two problems into consideration, a matrix structure used in the Butler coprocessor's design [9] is a good reference for the design of the event handler in the STEP (Figure 7).

Suppose in the current VSB, there are M tasks embedded in the IP core and S input Channels provided by the STEP, an $M \times S$ matrix is built and the unit U_{ij} ($i \leq M, j \leq S$) in the matrix responds to the event which comes from the j^{th} Channel and the processing in this unit will determine if task i is ready for execution in the IP core. With a matrix structure, several events coming from a different Channel can be responded to in parallel since the corresponding executions are carried out in different units.

If there is at least one U_{ij} in the i^{th} row of the matrix indicating the i^{th} task is ready for execution, a *ready* signal (which is written as Rdy for short in Figure 7) becomes valid. All ready tasks are called *candidates*. One and only one candidate can be scheduled out and loaded to the IP core for execution each time, and the ready signal

of the corresponding task will be withdrawn afterwards so that the task cannot be a candidate for next scheduling.

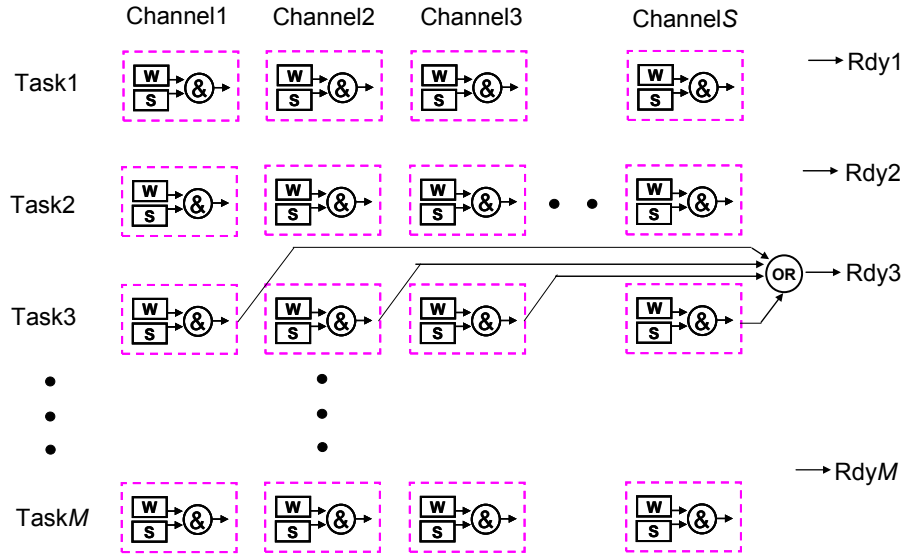


Figure 7: A Matrix Structure of Event Handler

The structure within every U_{ij} relies on the implementation of the VSB. When the VSB is used for data processing, the execution of a task needs the combination of both operation codes and the data for operation. An incoming event in this case indicates the corresponding data is available, and the operation codes which are embedded in the IP core will be ready for execution except when they are just under processing, or they are forbidden to be executed by other tasks in case of suspension, interruption or synchronization etc [9]. Therefore, two 1-bit variables *wait* and *stim* (which are written as W and S for short in Figure 7) are used in every unit of the matrix. The wait bit will be set when the operation codes of the corresponding task are ready for execution, and it will be reset otherwise. Similarly, the stim bit will be set when the event (mainly the corresponding data) is accessible and it will be reset otherwise. The ready signal for task i becomes valid (and the task becomes a scheduling candidate) only when at least one U_{ij} unit of the matrix has both stim and wait bits set.

The matrix structure will give high expandability to the STEP. When used in different environment or to cooperate with another IP core, the parameters of the matrix M and S may be changed accordingly. However, the Event Handler component can be easily adjusted by adding/deleting several units in the matrix while the entire structure keeps the same. As every unit in the matrix structure is identical, we only present the CPN model of a unit in this section (Figure 8).

3.2.2 Colour Set Description

When the implementation of the modelled VSB is specified as data processing, the colour of EVENT and TASK will be re-declared. In most cases, each task is given a unique *ID number* which will be used for the IP core to find the start address of the corresponding codes in its ROM memory if needed. Therefore, the colour TASK will be declared as:

color TASK = int with 0 .. Max

When data is transferred among different domains with different clock frequencies, an Asynchronous Communication Mechanism (ACM) can serve as an efficient and safe method used in many implementations and will be used in VSB design. Because the CPN model of an ACM has been designed in [10], an abstract *DATA* colour is declared as the colour string (as the set of all text strings) whose content shown as the DATA token value will be used to describe the property of the corresponding data.

Therefore the colour EVENT is re-declared as:

It means an EVENT token is composed by a TASK token and a DATA token. The TASK token indicates which operation will be used to process the data represented by the DATA token. The colour BIT keeps the same declaration in this model (as well as the rest models in this paper).

Figure 8: CPN Model of One Unit in the Event Handler

In Figure 8, the place *Channel* is used to hold EVENT tokens coming from one channel. A group of Channel places from all units of the Matrix is the extension of the EQ place in the top level model. Any EVENT token in this place will enable the *ACM* transition. This transition represents the data transfer carried out by the STEP when an ACM is used. The detailed description of this transition can be found in [10]. The occurrence of this transition will generate a TASK token to the *ID* place, which indicates the completion of the data preparation for the task suggested by the token value.

the upper left side of the transition *Sstim* (means Set stim bit). A guard is a Boolean expression and the corresponding transition is enabled only when the Boolean expression is true. Therefore, the *Sstim* transition is only enabled by a TASK token valued in '1' (ID1 is currently declared as 1). The occurrence of the *Sstim* transition will update the token value in the *Stim* place to '1' which means the data for the execution of task1 (task i is the short expression for the task whose ID number is i) is ready for execution. With an initial '1' token available in the *wait* place, the transition *Candidate* is enabled and the occurrence of this transition will update the token in the *Rdy* place to '1' which means task1 becomes a candidate for scheduling. A group of *Rdy* places from all units of the Matrix is the extension of the TQ place in the top level model.

The token value in the place *Ntask* indicates which task is chosen to be loaded to the IP core. Variable *ntask* is declared to represent the token value in the *Ntask* place. When the token value in this place becomes '1', the transition *selected* is enabled because task1 will be loaded to the IP core for execution. The occurrence of this transition will reset the value of tokens in both *stim* and *wait* places and the transition *Decand* (means disabled candidate) is enabled in sequence. The occurrence of the *Decand* transition will reset the token value in the *Rdy* place to '0', which means task1 will no longer be a candidate for scheduling and the corresponding ready signal becomes invalid.

3.2.4 Environmental Set Description

Similarly in the top level model in Figure 3, environmental places/transitions are highlighted by dark shade in the current model. The transition *Schedule* is used to represent the scheduling processing in the STEP. This transition is enabled only when the token in the *Rdy* place is '1' because the scheduling result will influence the current model only when task1 is a candidate task. No matter what scheduling policy may be implemented in the STEP, how quickly task1 can be chosen for loading after it becomes a candidate task is nondeterministic. Therefore, a CPN function *New()* is declared as follows:

fun New()=discrete(1,5)

This function will use the random integer number generator *discrete* provided by CPN Tools to generate a random integer number from 1 to 5. And the generated number indicates the ID number of the new selected task. A guard [$ntask \neq ID1$] (means *ntask* is not equal to ID1) is attached to the *Schedule* transition to make sure that the scheduling (as well as the execution of tasks in the IP core) is enabled until task1 is chosen (after that the scheduling result will not influence the current model until the token value in the *Rdy* place becomes '1' again).

The execution of the selected transition will also generate two tokens, one for the *new* place and the other for the *new2* place. The cooperation of place *new* with transitions *env* and *env1* are used to simulate the stochastic generation of another event corresponding to task1 from the same channel. The description of these places/transitions can be referred to places/transitions with the same names in the top level model. CPN function *P1()* (as well as *P2()* in the expression of arc directing from the *execution* transition to the *new2* place) shares the same form as the *P()* function in the top level with different rate λ . The occurrence of the transition *env1* represents the incoming of another event (as well as the data) corresponding to task1 in the current model.

Similarly, the cooperation of the place *new2* with the transition execution is used to simulate the execution of *task1* in the IP core. When a '1' token is generated in the *new2* place, it indicates the execution of *task1* is complete so that the wait bit will be set again accordingly by the occurrence of the *Swait* transition.

Because of the random token value given by functions *P1()* and *P2()*, either the transition *Sstim* or *Swait* can be first enabled (or they are concurrently enabled), which reflects the nondeterministic operations in the STEP. CPN simulation and the verification of the correctness of the model by state space checking is give in Appendix I.

3.3 The CPN Model of the Power Manager Component

In this section, we try to model and specify executions in the power manager component of a STEP where A&F policy is implemented. According to the previous section, a group of ready signals indicates the status of tasks embedded in the IP core. And the A&F policy can be realized by counting the number of valid ready signals so as to decide whether task accumulation is enough or not.

When tasks in an IP core are assumed to be independent from each other, there is no pattern that can be predicted when their corresponding ready signals become valid. The STEP must be alert to any change in ready signals so as not to miss any new valid ready signals. On the other side, a valid ready signal will only be withdrawn by the reset in some *stim* & wait bits in the Event Handler. Since the PM part in the STEP cannot disable any ready signals after accumulation counting, the PM needs to know which ready signals have been used in the accumulation and which are not.

Furthermore, the Matrix structure used in the Event Handler enables responding to events from different Channels in parallel, and therefore several ready signals can become valid simultaneously. These signals need to be arbitrated before they are counted and added to the accumulation result.

3.3.1 Model Description

Figure 9 presents the CPN model of the PM component in a STEP when only two example tasks are considered. Tokens' value '1' or '0' in the *Rdy1/Rdy2* places indicates whether the ready signal for *task1* or *task2* is valid or not. A BIT token in the *En1/En2* (means Enable) places is used to record whether the corresponding ready token has been used for accumulation. A '1' token in *En1/En2* place means the corresponding ready '1' token has not been used for accumulation and the *access1/access2* transition is enabled accordingly.

The occurrence of the *access1/access2* transition will update the token value in the *Irdy1/Irdy2* place to '1' respectively, indicating a new ready token can be counted. The occurrence of the *access1/access2* transition will also toggle the token value in *En1/En2* place to '0' so that the '1' token in *Rdy1/Rdy2* can only enable the *access1/access2* transition any more and duplicated counting is avoided in this model.

As demonstrated in the top level model, the PM only needs to work when the IP core is inactive. Therefore, transitions *access1/access2* can be enabled only when the token value in the *STEPSleep* place is '1'. This place is related but not the same as the *Sleep* place in the top level model, and their relationship will be explained in Section 3.4.2.

Because only one accumulation value is kept in this part, all ready signals can only be added to the accumulation value in sequence. Therefore arbiters are indispensable in

the current model. One easy solution is to build an arbitration array for all ready signals like Figure 10(a). If M is the number of tasks, this solution will use C_M^2 arbiters to build the sequence. Given a big number M , the number of arbiters and corresponding logic gates will increase dramatically.

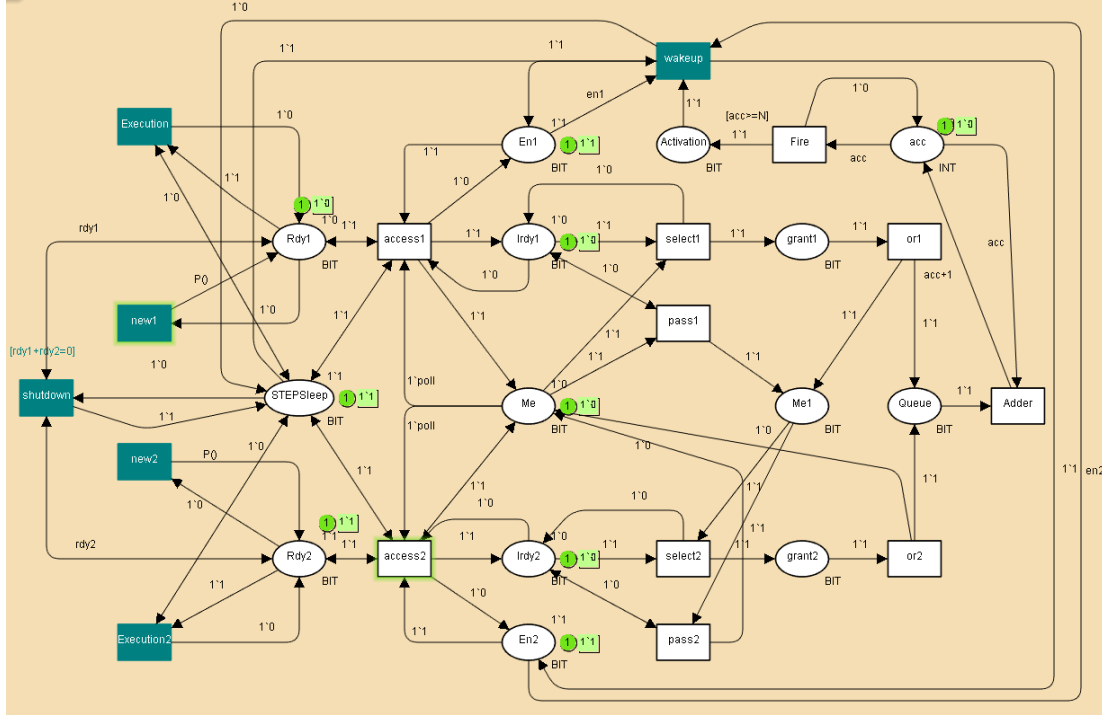


Figure 9: CPN Model of the Power Manager

Another improved solution (Figure 10 (b)) is inspired from the ring based arbiter introduced in the multi arbiter systems section in the book of [11]. In this case, a polling token will cycle in the arbitration system and any arbitration can only be carried out when it gets the token. Similarly in the A&F part, a valid ready signal can be added to the accumulation only when the polling token arrives. Therefore, no arbitration is needed to be given to different ready signals since they do not have any collision in polling token accessing. Although arbiters are still needed to solve the collision between the validation of one ready signal and the arrival of the polling token, the number of arbiters is reduced to M . Therefore this arbitration solution is chosen in the PM design.

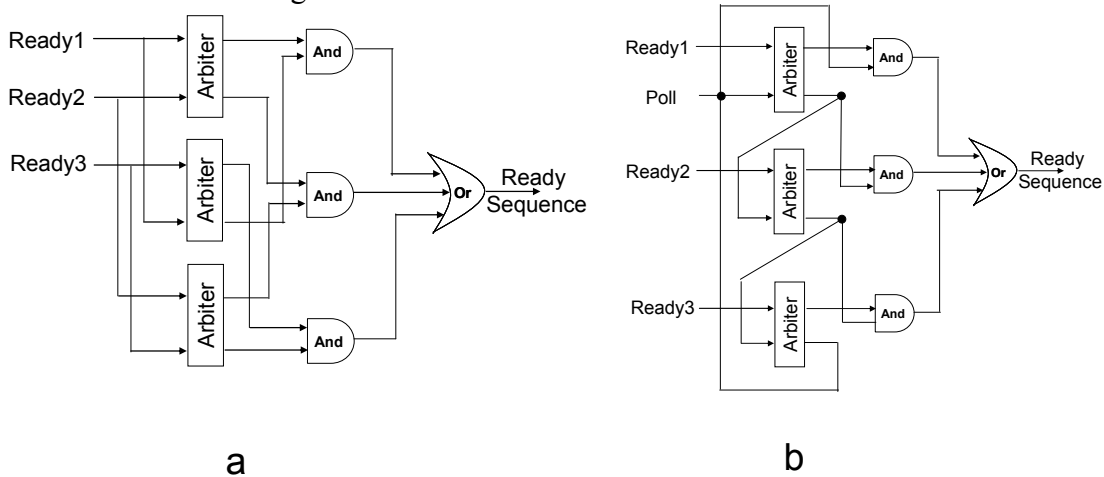


Figure 10: Two Arbitration Solutions in the PM Part

In the current CPN model, the polling token is held in the *Me/Me1* places and when at least one access transition occurs, the token in the *Me* place becomes '1' to enable the polling accumulation. The pair of *select1* and *pass1* transitions indicates the operation of polling accumulation of the ready signal for task1. If the token value in *Irdy1* is '1', the availability of the polling token in the *Me* place will enable the transition *select1*. The occurrence of the transition will first grant the ready token for accumulation, and then pass the polling token to the *Me1* place. If the token value in *Irdy2* is '0', the transition *pass1* will be enabled accordingly and pass the polling token directly to the *Me1* place. The occurrence of *select2/pass2* transition is carried out in the similar way and it will return the polling token to the *Me* place. For power saving reason, the polling will be ended after the occurrence of *select2/pass2* transition since the polling token value is reset to '0', and it will begin next time when at least one *access1/access2* transition occurs.

The arbitration between a ready signal and a polling signal is modelled by the competition of polling token in the *Me* place between the *access1/access2* transition and the *select1/pass1* transition. When two (or more) tokens in the *Rdyi* ($i=1,2$) places become '1', their corresponding *accessi* transitions will be enabled. After one *accessi* transition occurs, the polling token in the place *Me* becomes '1' and enables one of the pair *select1/pass1* transitions. Therefore, both the other *accessi* and one of the pair *select1/pass1* transitions are enabled concurrently. If the *select1/pass1* transition occurs first, no token is left in the *Me* place. The *accessi* transition is disabled until the end of one round polling accumulation. This occurrence sequence reflects the situation when the polling token is first granted by the arbiter and the valid ready signal will be added to the accumulation result next time when the polling token arrives. Otherwise, if the other *accessi* transition occurs first, the token polling will increase the accumulation by two. This occurrence sequence reflects the situation when the valid ready signal is first granted by the arbiter and one round of polling will realize the accumulation of several tasks.

The *or1/or2* transitions represent a logical OR gate, and the execution of one *or1/or2* transition will add one token to the *Queue* place and move the polling token to the *Me/Me1* place and let the token polling continue. The colour in the place *acc* is set to INT because the integer value of the token held in this place represents the accumulation result. As soon as one token is available in the *Queue* place, the *Adder* transition is enabled and the execution of this transition will increase the accumulation by 1. One guard [$acc \geq N$] is attached to the *Fire* transition to make sure one token will be added to the *Activation* place only when the token value in the *acc* place is greater than the accumulation limit N (N is set to 2 in the current model). The occurrence of the *Fire* transition will reset the token value in the *acc* place to '0' to prepare for the next accumulation procedure.

3.3.2 Environmental Set Description

The environmental transition *Wakeup* represents the wakeup processing in the IP core and its occurrence will set the token in the *STEPSleep* place to '0' and all transitions in the current model are disabled afterwards. The occurrence of this transition will also set the tokens in both *En1* and *En2* places to '1' so that new valid ready tokens can access the current model when the IP core becomes inactive again.

In the left side of Figure 9, environmental transitions *Execution1* and *Execution2* represent the executions of task1 and task2 in the IP core respectively. These two

transitions can be enabled concurrently and the random occurrence of these transitions represents the different scheduling result generated by the STEP. The occurrence of each Execution transition will reset the token value in the corresponding Rdy place. Assuming only two tasks are embedded in the IP core, the *shutdown* transition is enabled when both tokens in the Rdy place are '0'.

Environmental transitions *new1* and *new2* are used to change the token values in their corresponding Rdy1/Rdy2 places. The occurrences of these transitions reflect the generation of new events in the environment and function $P()$ (which is also used in the top level model) is used to make the generation of tokens in Rdy i place stochastically. All these environmental transitions/places will generate all possible combination of input tokens to and consume output tokens from the current system. The correctness of the current model has been verified by state space checking in Appendix II.

3.4 The CPN Model of the Task Manager Component

In this section, we try to model and specify executions in the task manager component of a STEP where task scheduling is provided.

3.4.1 Priority Based Round Robin Scheduling Priority

Although many different scheduling priorities have been used in various systems, we prefer to use a priority based round robin policy (Figure 11) in our task manager design.

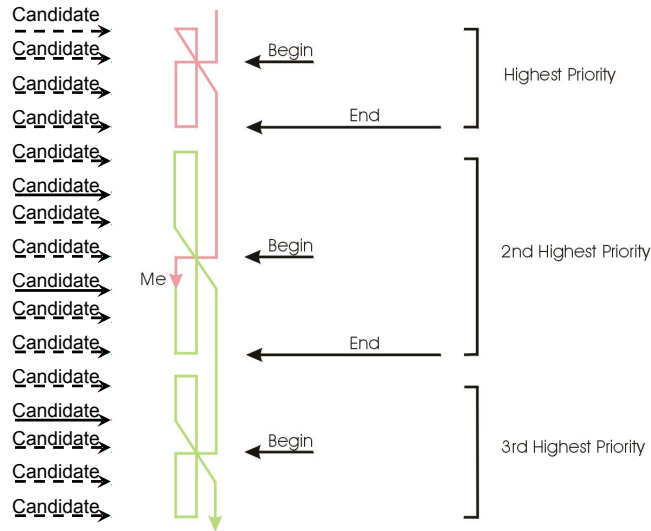


Figure 11: Priority Based Round Robin Policy

Arrows in the left of Figure 11 keep a list of all tasks in the IP core sorted by their priorities. A dotted arrow represents an invalid scheduling candidate (the corresponding task is not ready for execution) and a solid arrow indicates a valid candidate. A new scheduling will always start from the highest priority group and towards the lowest priority group. For tasks in the same priority group, the scheduler will use round robin policy to choose a new task so as to give all tasks in the same group fair opportunity to be executed in the IP core.

In each priority group, the task loaded to the IP core most recently is marked as a *last* task. In Figure 11, the last task in every priority group is pointed by the Begin arrow.

A new polling scheduling starts from the last task in the highest priority group and checks the validation of each task in turn. The scheduling ends when the first valid task is found. If no valid candidate can be found in this group, the scheduling point will jump to the last task in the second highest priority group to carry out the similar exploration. When no valid task can be found even in the lowest priority group, it means no task is ready for execution, and a particular ID number (for example 0 or 255) will be fetched to the IP core.

3.4.2 The CPN Model for Scheduling

A CPN model (Figure 12) is built to show the scheduling execution when four example tasks are involved. In Figure 12, tokens in places Rdy_i ($i=1,2,3,4$) represent the status of the corresponding ready signals. In this example, task1 and task2 have the same priority which is higher than that of task3 and task4. The initial tokens in the model indicate only task3 is a valid candidate. Task1 and task3 are set as the last task in each group and one '1' token is given to places $Last1$ and $Last3$ each while tokens in places $Last2$ and $Last4$ are '0'. Places Me , MeN and Me_i ($i=1,2,3,4$) are used to hold the polling token for the round robin scheduling in each group. A new scheduling is enabled by a '1' token in the Me place and simulation results in Figure 13 show the scheduling procedure (a dotted rectangle in each figure is used to indicate an enabled transition).

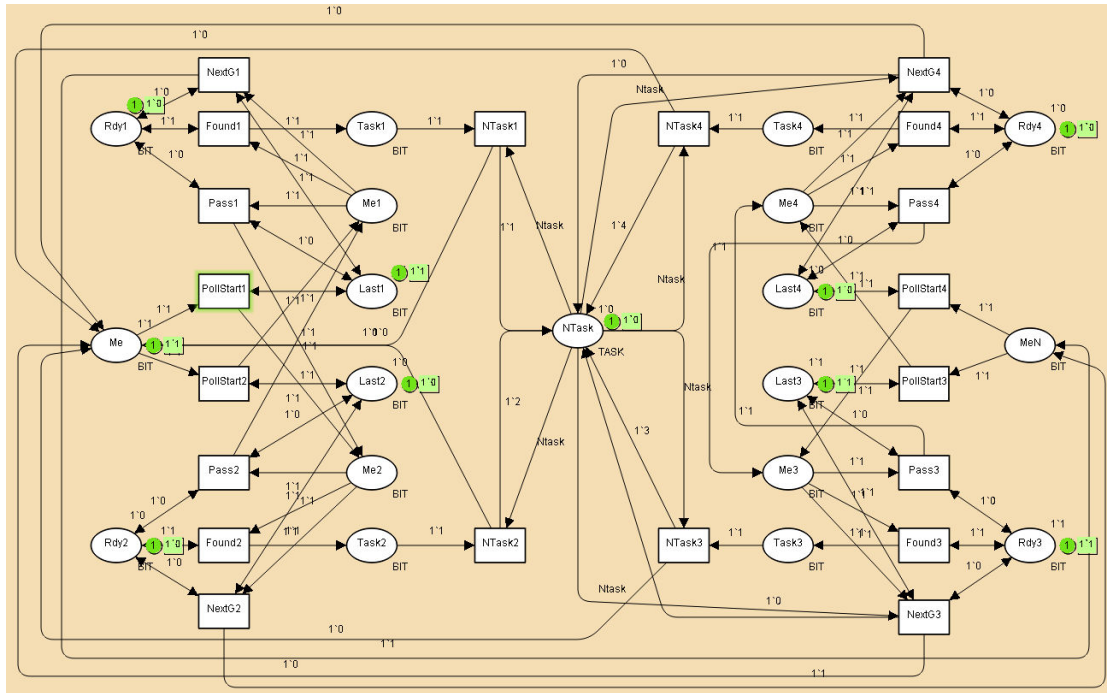


Figure 12: CPN Model for Scheduling

The token in the place Me will first enable the scheduling in the high priority group. When task1 serves as the last task in this group, the transition $PollStart1$ is enabled. The occurrence of this transition will add one token to the $Me2$ place because task2 is the first task for checking. With the initial token '0' in the $Rdy2$ place (indicating task2 is not a valid candidate), the $pass2$ transition is enabled (Figure 13(a)) and the occurrence of this transition will pass the polling token to the place $Me1$.

As the polling token finds the last task in the group (task1) is not a valid candidate, it means no valid candidate contains in this group and the scheduling will move on to

the next priority group. Therefore, the *NextG1* instead of *pass1* transition is enabled by the token in the Me1 place (Figure 13(b)).

The occurrence of the *NextG1* transition will generate one token to the MeN place which enables transition *PollStart3* because task3 is the last task in the group (Figure 13(c)). After the occurrence of the *PollStart3* transition, the polling token is moved to the Me4 place and task4 is under checking. With one '0' token in the Rdy4 place, the transition *pass4* is enabled (Figure 13(d)) and one token is added to the Me3 place after this transition's occurrence. Since the token value in the Rdy3 place is '1', the transition *Found3* is enabled which indicates one valid candidate task is found (Figure 13(e)).

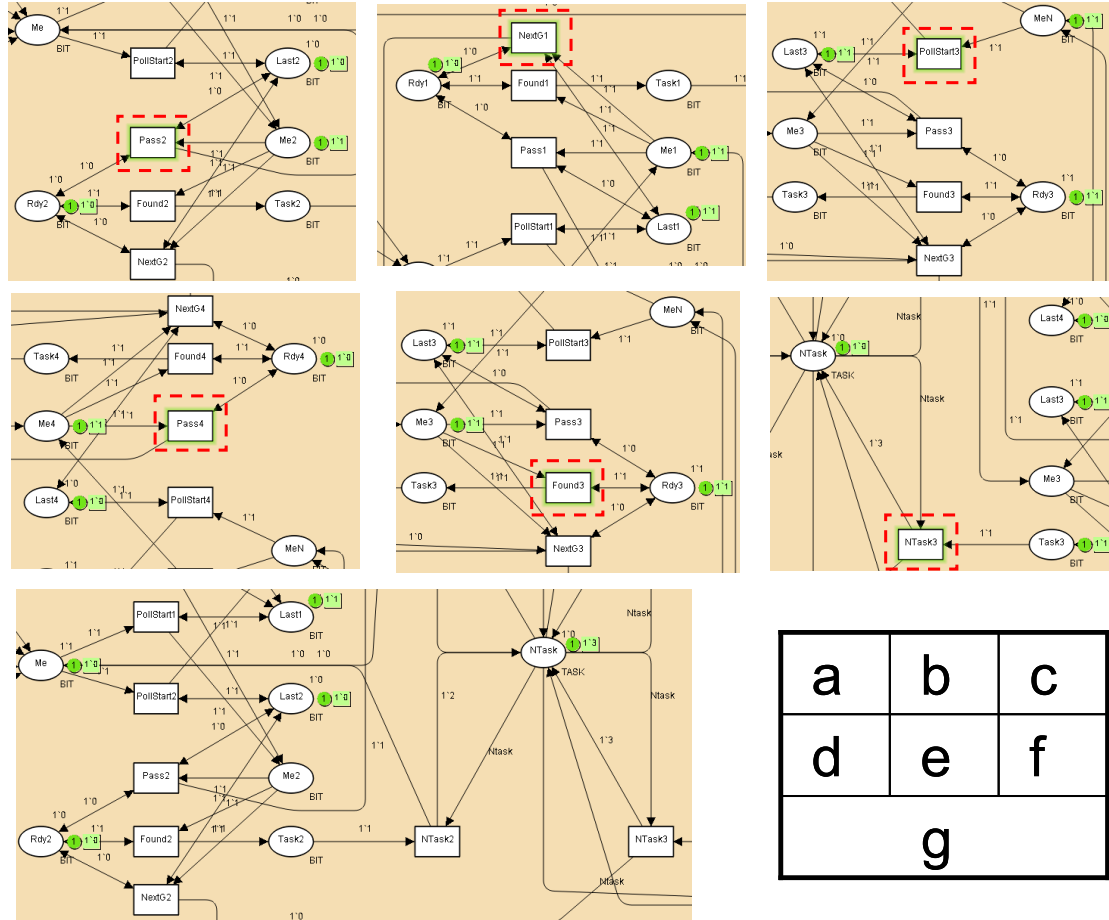


Figure 13: Simulation Result of the Scheduling

The token in the place *Ntask* is used to save the scheduling result. As soon as a token is generated in the *Task_i* ($i=1,2,3,4$) place, the corresponding *NTask_i* ($i=1,2,3,4$) transition is enabled (Figure 13(f)) and its occurrence will update the token value in the *NTask* place with the corresponding ID number of the valid candidate, and move a '0' token to the Me place which indicates the completion of scheduling (Figure 13(g)).

Given any combination of token values in the four *Rdy_i* places, the scheduling flow is similar. Next, we only discuss the scheduling flow when all tokens in the *Rdy_i* places are '0'. The first several steps are similar to the case introduced in Figure 13(a) to (d). Since the token '0' in *Rdy3* place indicates task3 is not a valid candidate task, the *NextG3* transition is enabled (the dotted rectangle in Figure 14) because no candidate task can be found in this priority group. Without any lower priority group available, it

means no task can be executed in the IP core. Therefore, the occurrence of NextG3 (or NextG4 when task4 is the last task in the group) will reset the token value in NTask place by '0' to indicate the IP core no more new task can be loaded.

In Figure 14, transitions and places with same index number (for example, transitions PollStart1, Found1 share the same index 1) can be seen as a basic unit of the model (the dotted cycle in Figure 14). Therefore, the current model can be easily extended to represent a scheduler when more tasks are involved, or when tasks are divided into more groups.

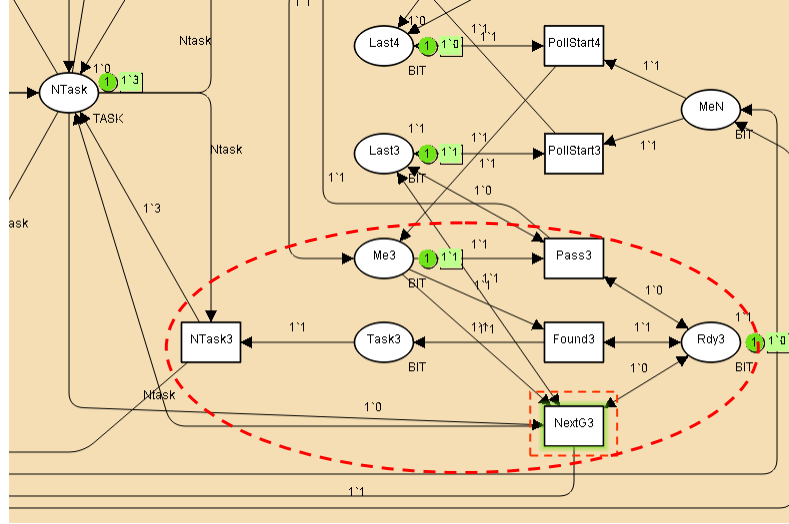


Figure 14: No Ready Task Found in Scheduling

3.4.3 The CPN Model of the Task Manager

Although transitions and places in Figure 13 can successfully carry out a priority based round robin scheduling, some more places and transitions are needed to guarantee the safety and correctness of scheduling every time. Figure 15 gives one example model of the Task Manager in the STEP and its test environment when only two tasks (and one priority group) is concerned in the scheduling.

The environmental place within the dotted circle is named *LoadEn* whose token '1' represents the task loading request from the IP core. A '1' token in the LoadEn place will enable the *Load* transition in the right side of the figure and the occurrence of the transition indicates the task loading execution in the IP core. One token whose value is the ID number of the new task will be added to the *Ltask* (means Loaded task) place in consequence.

Tokens in the Rdy1/Rdy2 places indicate whether task1/task2 is a valid candidate task or not. As external events may come to the current VSB at any time, the two tokens in the Rdy1 and Rdy2 places may become '1' simultaneously when the Load transition is enabled. In this case, new scheduling execution and task loading execution are carried out simultaneously. Suppose the new scheduling will update the token value in the Ntask place from '1' to '2', whether task1 or task2 will be loaded to the IP core depends on whether the scheduling transitions or the loading transition will occur first. The uncertainty in task loading will confuse the IP core and may cause serious consequence. A safer design will enable scheduling only when no load request is given. In Figure 15, transitions *Access1* and *Access2* can be enabled only when the token value in the LoadEn place is '0'. Therefore, when the token in the LoadEn place becomes '1', no further token change in Rdy1/Rdy2 place can influence the token

value in the LTask place. As task scheduling is of no use when the IP core is in its off mode, another enabling precondition of transitions access1 and access2 is the existence of '0' token in the *STEPSleep* place.

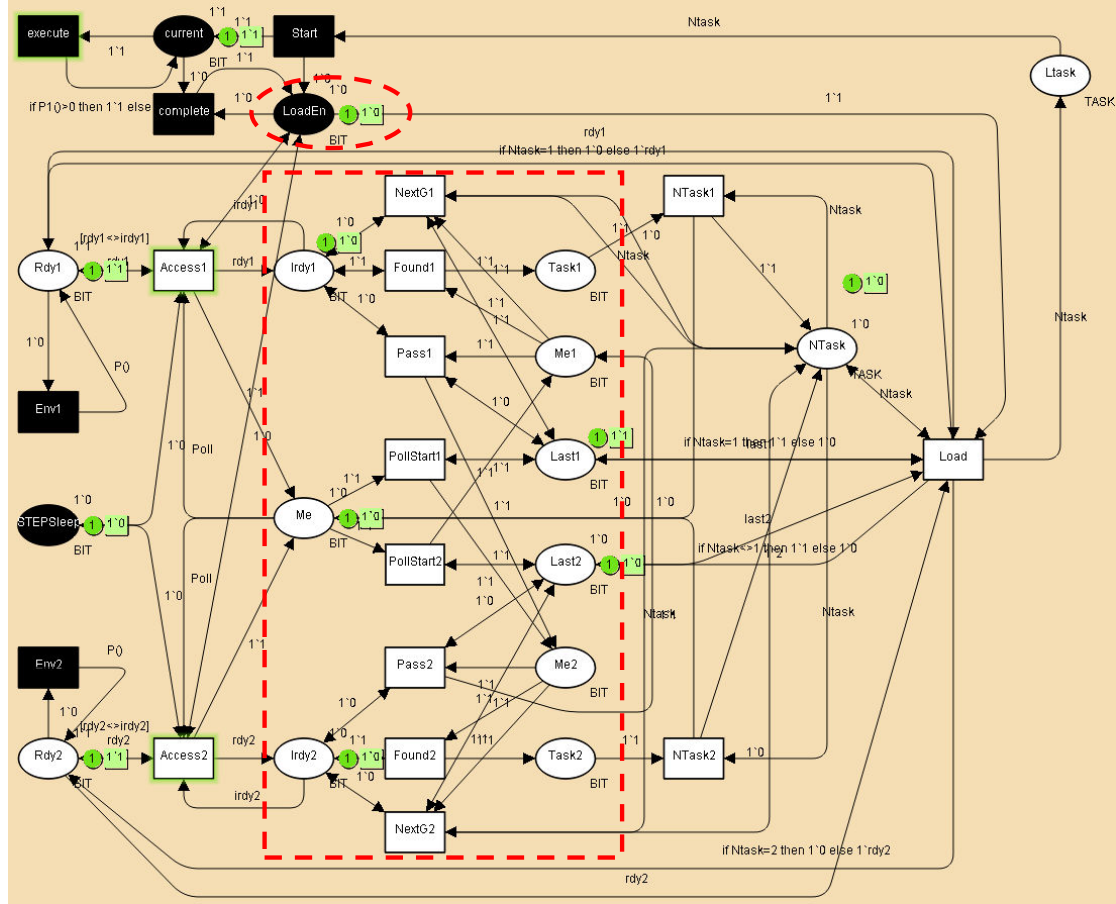


Figure 15: The Full CPN Model of the Task Manager

The tokens held in places *Irdy1* and *Irdy2* indicate the status of ready signals for the usage of scheduling. Variables *irdy1*, *irdy2*, *rdy1* and *rdy2* are used to indicate the token value in the place with the same name (But capital first character) respectively. With the guard $[irdy1 \triangleleft rdy1]$ and $[irdy2 \triangleleft rdy2]$ in the *access1/access2* transitions, scheduling will only begin when some changes happen to the ready signals. The occurrence of these transitions will update the token value in the *Me* place to '1'. The scheduling executions are modelled by places and transitions within the dotted rectangle which has been introduced in Section 3.4.2 in detail.

According to the model, when more than one ready token is toggled concurrently, one *Assessi* ($i=1,2$) transition and some scheduling transition within the dotted rectangle may be enabled concurrently. The different occurrence sequences of these transitions reflect the competition between the validation of a ready signal and the arrival of the round robin polling signal. However, given no valid *LoadEn* signal is generated from the IP core, different occurrence sequences of these transitions will achieve the same scheduling result.

The occurrence of the *Load* transition will not only load the ID number of the new task to the IP core, but also update the status in the STEP. After the task that is loaded to the IP core, the corresponding unit in the EH will be reset and the task will not serve as a candidate for scheduling any more. In Figure 15, the expression of the arc from the place *Rdy1* to the transition *Load* is written as “if $Ntask=1$ then 1'0 else

1'rdy1". Therefore, if the token value in the place Ntask is '1' which means when task1 is loaded to the IP core, the token value in the place Rdy1 will be reset to '0'. Otherwise, the token value stays the same as before. Furthermore, any token reset in the Rdy1/Rdy2 place will enable the Access1/Access2 transition when the IP core starts execution about the new task and the token value in the LoadEn place becomes '0'. Therefore, new scheduling will be carried out in parallel with the execution in the IP core and a new task can be prepared in the Ntask place in advance of the next load request from the IP core.

The occurrence of the Load transition will also reset the last task in every priority group if it changes. And if no task is found to be ready for execution, the occurrence of the Load transition will reset the last task to its default status (for example, in the current model, task2 is the default last task in its group).

3.4.4 Environmental Set Description

When one token is added to the Ltask place, the environment transition *Start* is enabled which indicates the IP core starts the execution of the new chosen task. Therefore one token '1' is given to the place *current*, which indicates that one task is under processing. The occurrence of the Start transition will give one '0' token to the LoadEn place, which means the task loading procedure is completed. One function *P1()* (which is the same as the *P1()* function in the EH unit model in Figure 8) is used in the arc expression from the *execution* transition to the current place. This function is used to simulate the stochastic processing behaviour in the IP core. When the token value in the current place becomes '0', the current task's execution is completed. And the token value in the LoadEn place will be updated to '1' and new tasks will be loaded afterwards. If the token value in the Ltask place is '0' which means no more valid task has been loaded to the IP core, the Start transition can be seen to indicate the shutdown operation in the IP core and the execute transition can be seen to indicate task accumulation procedure. Similarly, the complete transition indicates the activation of the IP core in this case.

The environmental transition *Env1/Env2* uses the function *P()* (which has been introduced in the top level model) to simulate the generation of new event which will in turn validate the corresponding ready signals again. All these environmental transitions/places will generate all possible combinations of input tokens to and consume output tokens from the current system. State space has been done to prove the correctness of the current model (Appendix III).

3.5 The CPN Model of the Output Controller and Interface Components

When the wakeup and shutdown executions in an IP core are not taken as instantaneous, it is highly possible that some events come during the same time. As indicated in Section 3.1.6, executions in the STEP about these events may interrupt the mode switching transitions in an IP core so as to bring serious consequences. The interface design in this section is to avoid possible hazard brought by STEP executions.

As indicated in the top level model, the output controller will generate a new event when the execution of the current task is completed. If the new task stimulated by the event generated from the output controller locates in the same VSB, the new task can become ready for execution much faster than the case when the new task is located in the other VSB. It is because both asynchronous/synchronous transform and data

transfer between two VSBs are omitted. The CPN Model in this section tries to specify this difference.

Figure 16 presents the CPN model of the Interface and Output Control part in the STEP where only two example tasks (task1 and task2) are concerned. The declarations of the four token colours involved in the figure, BIT, EVENT, TASK and DATA, are the same as those given in the EH CPN model in Section 3.2.

The relationship among *Fire* and *wakeup* transition as well as the *Activation* and *STEPSleep* places have been introduced in the PM in Section 3.3 (the accumulation limit is set to 1 to simplify the current model). The token in the *STEPSleep* place indicates the command given by the PM while that in the *Sleep* place indicates the mode of the IP core. Transitions *Waking* and *Shutting* represent the wakeup and shutdown executions in the IP core respectively. A '0' token in the *STEPSleep* place will enable the *Waking* transition and the latter's occurrence will update the token in the *Sleep* place to '0', which indicates the completion of the wakeup execution in the IP core. In Section 3.3 and 3.4, the token in the *STEPSleep* place instead of that in the *Sleep* place is used to enable/disable the executions in the PM and TM components. Therefore, the PM execution can be terminated and the TM execution can start as soon as the beginning of the wakeup execution, which can reduce system latency as well power dissipation.

Transitions ACM, as well as places Ch3, Rdy and ID, are used to represent the executions in the EH component. The token value in the Rdy place indicates the ID number of the ready task. Different from the model in Section 3.2, the occurrence of the ACM transition will also produce a DATA color token to the *DIN2* place, which indicates the data for the corresponding task's execution.

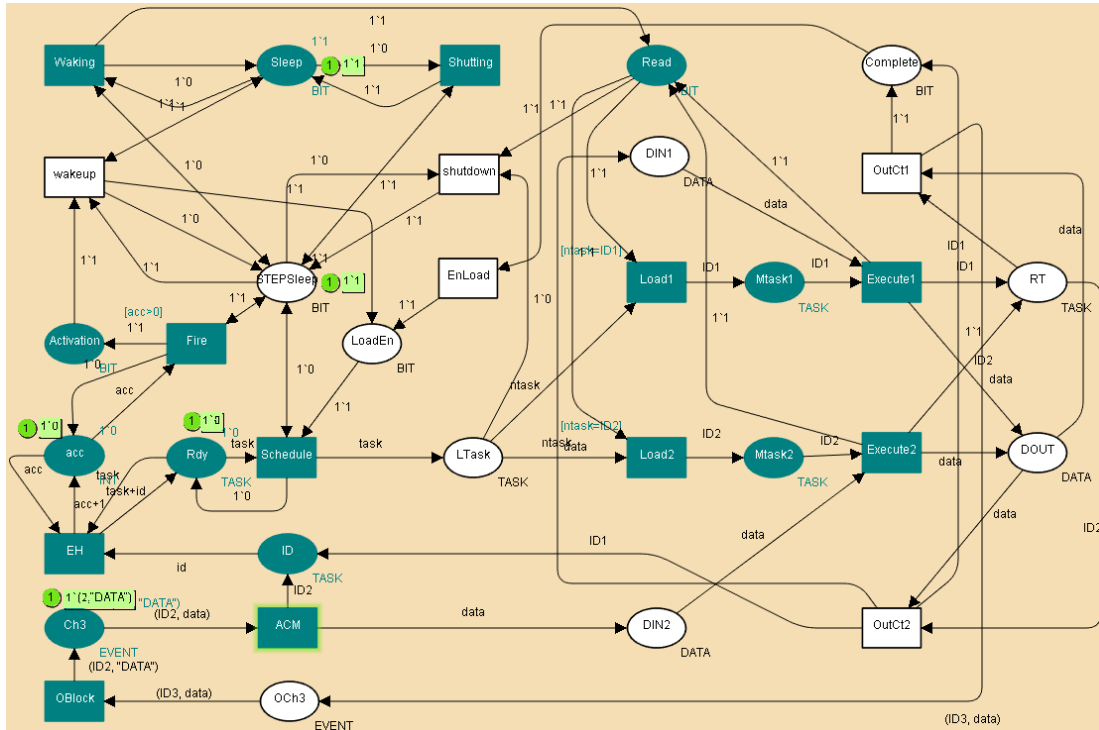


Figure 16: The CPN Model of the Output Controller and Interface Components

The occurrence of the wakeup transition will also generate one token to the *LoadEn* place so that the scheduling result can be loaded to the *LTask* place. A '1' token in the *Read* place indicates the read signal from the IP core, and it will enable the

Load1/Load2 transition based on the ID number of the new task. The occurrence of the *Execution1/Execution2* transition represents the execution of task1/task2 respectively in the IP core, and it will consume the DATA token in the *DIN1/DIN2* place indicating the data processing involved in the corresponding task execution. When the task execution is completed, a TASK token is generated to the *RT* place indicating the ID number of the completed task and a DATA token is put to the *DOUT* place indicating the result data generated by the task execution.

When the task execution is completed, the output controller starts event routing, which means to decide which task is supposed to use the result data and which VSB the task (called *target task* later) locates in. In the current model, we suppose the target task for task1 is task3 which is located in a different VSB and that for task2 is task1 which is in the current VSB. When task1 is the completed task, the transition *OutCt1* is enabled. The occurrence of this transition generates an EVENT token in the *OCh3* place, which represents the generation of a new event that will be sent to the other VSB with Output Channel 3. The environmental transition *OBlock* represents the event relay in the environment, and it will generate an EVENT token in the *Ch3* place which means a new event comes to the current VSB. The corresponding task can become ready only after the completion of data transfer/transform represented by the occurrence of the ACM transition.

On the other hand when task2 is the completed task, the transition *OutCt2* is enabled. Its occurrence simply move the DATA token to the *DIN1* place because it represents simple data transfer within the same time domain. Therefore task1, as the target task of task2, can become ready much faster than the previous case. The occurrence of *OutCt1/OutCt2* transition will also add one token to the *Complete* place, which indicates the completion of output control. A BIT token is added to the *LoadEn* place in sequence to require the new task from the TM component. If the value of the token moved to the *Ltask* place is '0', it indicates no active task can be found. In this case, the *shutdown* transition is enabled whose occurrence will toggle the token value in the *STEPSleep* to '1'. This token value change will enable the shutting transition and the token value in the *Sleep* place will be changed to '1' after the latter's occurrence.

In Section 3.1.6, we present the hazard brought by concurrent execution of the event handling in the STEP and the shutdown processing in the IP core. This hazard is avoided by the design in the current model. When the shutdown processing is carried out in the IP core, the token value in the *STEPSleep* place is '1' and that in the *Sleep* place is '0'. Suppose there are enough events come during this period and the *Fire* transition occurs to generate a '1' token in the *Activation* place. However, the *wakeup* transition cannot be enabled because it requires a '1' token in the *Sleep* place. When the *wakeup* transition is disabled, no token will be sent to the *LoadEn* place. Therefore, no new task will be loaded to the IP core so as to interrupt the shutdown processing.

4. CONCLUSION AND FUTURE WORK

This paper presents the modelling and design work of an asynchronous coprocessor named as STEP in a framework of a virtual self-timed block (VSB) under GALS architecture. This coprocessor is designed to provide effective power control and fast event handling for task executions in an IP core. Hierarchical CPN models have been used to develop the STEP design from basic functional specifications to detailed

signals and data processing. CPN simulation and state space checking have been used to testify the correctness of our design.

Although strong in modelling concurrent and nondeterministic executions in a real system, CPN has limited power for time based analysis, and can not provide concrete verification about the efficiency of a VSB in power saving for an SoC. In the future, another modelling tool will be used to simulate the execution of an SoC built by VSBs so as to verify the power efficiency of our design.

The current model of STEP can only give power on-off control to an IP core, the future work is to extend the model of the power manager component of a STEP to give dedicate mode switching control to an IP core with multiple operation modes.

REFERENCES

- [1] S.Moore, G.Taylor, R.Mullins, P.Robinson, "Point to Point GALS Interconnect", *ASYNC* 2002.
- [2] H.R.Simpson, "Four-slot fully asynchronous communication mechanism", *IEE Proceedings, Computers and Digital Techniques*, 1990
- [3] L.Benini, A.Bogliolo, G. De Micheli, "A survey of Design Techniques for System-level Dynamic Power Management" *IEEE Transactions on VLSI* June 2000.
- [4] Yuan.Chen, Fei.Xia, Alex.Yakovlev, "Virtual Self-timed Block for Systems-On-Chip", *ISCAS*, 2006
- [5] Yuan Chen, Fei Xia, Delong Shang and Alex Yakolev "Stochastic Modeling Of Dynamic Power Management Policies And Analysis Of Their Power-Latency Tradeoffs", 4th UKEF, Southampton, 2008
- [6] K.Jensen, L.M.Kristensen, L.Wells, "Coloured Petri Nets and CPN Tools for modeling and validation of concurrent systems", *International Journal on Software Tools for Technology Transfer*, 2007.
- [7] J. Cortadella, K. Gorgonio, F. Xia, A. Yakovlev, "Automating Synthesis of Asynchronous Communication Mechanisms". *ACSD* 2005
- [8] CPN Tools, <http://wiki.daimi.au.dk/cpntools/cpntools.wiki>
- [9] E.Campbell, H.Simpson, "Patent – Integrated circuits for multitasking support in single or multiple processor networks", *World Intellectual Property Organisation WO 97/22926*, June 1997
- [10] K. Gorgonio, F. Xia, "Modeling and Verifying Asynchronous Communication Mechanisms using Coloured Petri Nets", *Technical Report*, NCL-EECE-MSD-TR-2008-127, School of EECE, Newcastle University, March 2008
- [11] D.J.Kinniment, "Synchronization and Arbitration in Digital Systems", *John Wiley & Sons, Ltd*, 2007.

APPENDIX

Appendix I State Space Report for Event Handler Part of a STEP

This state space report is about the CPN model of Event Handler in Figure 8:

Statistics			
<hr/>			
Occurrence Graph		Scc Graph	
Nodes:	31	Nodes:	6
Arcs:	44	Arcs:	5
Secs:	0	Secs:	0
Status:	Full		
Boundedness Properties			
<hr/>			
Best Integers Bounds	Upper	Lower	
Matrix'Channel 1	1	0	
Matrix'NTask 1	1	1	
Matrix'Stim 1	1	1	
Matrix'Wait 1	1	1	
Matrix'new 1	1	0	
Matrix'new2 1	1	0	
Matrix'Rdy 1	1	1	
Best Upper Multi-set Bounds			
Matrix'Channel 1 1` (1,"DATA1")	Matrix'Wait 1	1`0++1`1	
Matrix'Stim 1 1`0++1`1	Matrix'new 1	1`0++1`1	
Matrix'new2 1 1`0++1`1	Matrix'Rdy 1	1`0++1`1	
Matrix'NTask 1 1`1++1`2++1`3++1`4++1`5			
Best Lower Multi-set Bounds			
Matrix'Channel 1	empty	Matrix'Wait 1	empty
Matrix'Stim 1	empty	Matrix'new 1	empty
Matrix'new2 1	empty	Matrix'Rdy 1	empty
Matrix'NTask 1	empty		
Home Properties			
<hr/>			
Home Markings: None			
Liveness Properties			
<hr/>			
Dead Markings: None			
Dead Transitions Instances: None			
Live Transitions Instances: Matrix'env1			

Appendix VI: State Space Report for A&F Part in a STEP

This state space report is about the CPN model of A&F part in Figure 9.

Statistics			
Occurrence Graph		Scc Graph	
Nodes:	58	Nodes:	29
Arcs:	104	Arcs:	49
Secs:	0	Secs:	0
Status:	Full		
Boundedness Properties			
Best Integers Bounds		Upper	Lower
AF'Cand 1		2	0
AF'En1 1		1	1
AF'En2 1		1	1
AF'Irdy1 1		1	1
AF'Irdy2 1		1	1
AF'Me 1		1	0
AF'Mel 1		1	0
AF'Rdy1 1		1	1
AF'Rdy2 1		1	1
AF'Sleep 1		1	1
AF'Wakeup 1		1	0
AF'acc 1		1	1
AF'grant1 1		1	0
AF'grant2 1		1	0
Best Upper Multi-set Bounds			
AF'Cand 1	2`1	AF'En1 1	1`0++1`1
AF'En2 1	1`0++1`1	AF'Irdy1 1	1`0++1`1
AF'Irdy2 1	1`0++1`1	AF'Me 1	1`0++1`1
AF'Mel 1	1`1	AF'Rdy1 1	1`0++1`1
AF'Rdy2 1	1`0++1`1	AF'Sleep 1	1`0++1`1
AF'Wakeup 1	1`1	AF'acc 1	1`0++1`1++1`2
AF'grant1 1	1`1	AF'grant2 1	1`1
Best Lower Multi-set Bounds			
AF'Cand 1	empty	AF'En1 1	empty
AF'En2 1	empty	AF'Irdy1 1	empty
AF'Irdy2 1	empty	AF'Me 1	empty
AF'Mel 1	empty	AF'Rdy1 1	empty
AF'Rdy2 1	empty	AF'Sleep 1	empty
AF'Wakeup 1	empty	AF'acc 1	empty
AF'grant1 1	empty	AF'grant2 1	empty
Home Properties			
Home Markings: None			
Liveness Properties			
Dead Markings: None			
Dead Transitions Instances: None			
Live Transitions Instances: AF'new1 AF'new2			

According to the Best Integers Bounds in the report, all places other than the Cand place contain no more than one token in any cases, which indicates the correct operation in this part without any confusion. The availability of multiple tokens in the Cand place happens when more than one ready signal become valid simultaneously. The upper bound of the token in this place is M ($M=2$ is the number of tasks in the model) means the accumulation of simultaneous validated ready signals will not be conflicted with that of later validated ready signals.

The Best Upper Multi-set Bound of the acc place indicates the token value in this place will be no more than N ($N=2$ is the accumulation limit) which means the activation signal will be generated without delay when the accumulation limit is achieved. The Best Upper Multi-set Bound of the Me place indicates only one polling accumulation is carried out each time because it holds at most '1' token. The availability of '0' token in the Me place indicates polling accumulation can have a rest when no more ready signal becomes valid.

Appendix VII: State Space Report for Task Manager Part in a STEP

This state space report is about the CPN model of Task Manager part in Figure 15:

Statistics			
Occurrence Graph		Scg Graph	
Nodes:	873	Nodes:	201
Arcs:	2521	Arcs:	468
Secs:	1	Secs:	0
Status:	Full		
Boundedness Properties			
Best Integers Bounds	Upper	Lower	
TM'Irdy1 1	1	1	
TM'Irdy2 1	1	1	
TM'Last1 1	1	1	
TM'Last2 1	1	1	
TM'LoadEn 1	1	0	
TM'Ltask 1	1	0	
TM'Me 1	1	0	
TM'Me1 1	1	0	
TM'Me2 1	1	0	
TM'NTask 1	1	1	
TM'Rdy1 1	1	1	
TM'Rdy2 1	1	1	
TM'Task1 1	1	0	
TM'Task2 1	1	0	
TM'current 1	1	0	
Best Upper Multi-set Bounds			
TM'Irdy1 1	1`0++1`1	TM'Irdy2 1	1`0++1`1
TM'Last1 1	1`0++1`1	TM'Last2 1	1`0++1`1
TM'LoadEn 1	1`0++1`1	TM'Ltask 1	1`0++1`1++1`2
TM'Me 1	1`0++1`1	TM'Me1 1	1`1
TM'Me2 1	1`1	TM'NTask 1	1`0++1`1++1`2
TM'Rdy1 1	1`0++1`1	TM'Rdy2 1	1`0++1`1
TM'Task1 1	1`1	TM'Task2 1	1`1
TM'current 1	1`0++1`1		
Best Lower Multi-set Bounds			
TM'Irdy1 1	empty	TM'Last1 1	empty
TM'Irdy2 1	empty	TM'Last2 1	empty
TM'LoadEn 1	empty	TM'Ltask 1	empty
TM'Me 1	empty	TM'Me1 1	empty
TM'Me2 1	empty	TM'NTask 1	empty
TM'Rdy1 1	empty	TM'Rdy2 1	empty
TM'Task1 1	empty	TM'Task2 1	empty
TM'current 1	empty		
Home Properties			
Home Markings: None			
Liveness Properties			
Dead Markings: None			
Dead Transitions Instances: None			

Live Transitions Instances: TM^{execute} 1
