High Level Modelling and Design of a Low Power Event Processor

Yuan Chen

Technical Report Series
NCL-EECE-MSD-TR-2009-141

January 2009

Contact: yuan.chen1@ncl.ac.uk

High Level Modelling and Design of a Low Power Event Processor

Yuan Chen

School of Electronic Engineering and Computer Engineering

Newcastle University

PhD Thesis

January 2009

To My Parents

Contents

List of Figures

List of Tables

List of Abbreviations

ACM	Asynchronous Communication Mechanism
A&F	Accumulation & Fire
APDV	Average Percentage of Deadline Violation
CPN	Colored Petri Nets
CU	Control Unit
DFS	Dynamic Frequency Scaling
DL	Dead Line
DSP	Digital Signal Processing
DPM	Dynamic Power Management
DVS	Dynamic Voltage Scaling
DVFS	Dynamic Voltage and Frequency Scaling
ECG	ElectroCardioGram
EQ	Event Queue
EH	Event Handler
FCFS	First Come First Service
GALS	Globally Asynchronous and Locally Synchronous
GUI	Graphical User Interface
IP	Intellectual Property
MATLAB	Matrix laboratory

PDF	Probability Distribution Function
PM	Power Manager
PMC	Power Manageable Component
PN	Petri Nets
QoS	Quality of Service
RTL	Register Transfer Level
SCC	Strongly connected Component
SOC	System On Chip
SP	Service Provider
SR	Service Requestor
STEP	Self Timed Event Processor
TBE	Break-Even Time
TOL	Tolerance of Latency
TM	Task Manager
TQ	Task Queue
VLSI	Very Large Scale Integration
VSB	Virtual Self-timed Block

List of Publications

Conference Publications:

1 Yuan Chen, Fei Xia, Alex Yakovlev, "Virtual Self-timed Block for Systems-On-Chip", ISCAS, 2006

2 Yuan Chen, Fei Xia, Delong Shang, Alex Yakovlev, "The Design of Virtual Self-timed Block for Activity Communication in SOC" ACSD 2007 pp.100-109

Workshop/Forum Publications:

1 Yuan Chen, Fei Xia, Alex Yakovlev, "The Design of STEP Processor", 17th UK Asynchronous Forum, Southampton, 2005

2 Yuan Chen, Fei Xia, Delong Shang, Alex Yakovlev, "Power Management with Accumulation and Fire Mechanism", 19th UK Asynchronous Forum, London, 2007

3 Yuan Chen, Fei Xia, Delong Shang and Alex Yakovlev "Fine Grain Stochastic Modeling and Analysis of Low Power Portable Devices with Dynamic Power Management", 24th UKPEW Workshop, London, 2008

4 Yuan Chen, Fei Xia, Delong Shang and Alex Yakovlev "Stochastic Modelling Of Dynamic Power Management Policies And Analysis Of Their Power-Latency

Tradeoffs", 4th UKEF, Southampton, 2008

Technical Reports:

1 Yuan Chen, Fei Xia, Alex Yakovlev, "Modelling Asynchronous Artificial Neural Networks for Energy Efficient Implementation", NCL-EECE-MSD-TR-2005-105, Microelectronic System Design Group, School of EECE, University of Newcastle upon Tyne, May 2005

2 Yuan Chen, Fei Xia, Alex Yakovlev, "Stochastic Modelling Of Dynamic Power Management Policies And Analysis Of Their Power-Latency Tradeoffs", NCL-EECE-MSD-TR-2007-123, Microelectronic System Design Group, School of EECE, Newcastle University, November 2007

3 Yuan Chen, Fei Xia, Delong Shang, Alex Yakovlev, "Virtual Self-timed Block Design using Coloured Petri Net", NCL-EECE-MSD-TR-2008-134, Microelectronic System Design Group, School of EECE, Newcastle University, August 2008

4 Yuan Chen, Fei Xia, Delong Shang, Alex Yakovlev, "Studying an SoC with Virtual Self-timed Blocks using MATLAB Simulink", NCL-EECE-MSD-TR-2008-135, Microelectronic System Design Group, School of EECE, Newcastle University, August 2008

5 Yuan Chen, Fei Xia, Delong Shang, Alex Yakovlev, Mohammad Rastegar Tohid, "Modelling and Design of a Low Power Event Processor", NCL-EECE-MSD-TR-2008-136, Microelectronic System Design Group, School of EECE, Newcastle University, September 2008

Acknowledgement

I would like to first express my gratitude to my supervisor, Prof. Alex Yakovlev, for introducing me to the area of asynchronous design. His guidance and valuable advice is essential to every achievement in my research. I would specially thank Dr. Fei Xia for his help in my entire PhD study in inspiring new ideas, in discussing about my research and in patient correcting errors in my paper drafts. I would also thank Dr. Delong Shang for all numerous technical discussions and instructions in my research.

I would like also to express my gratitude to my parents and family for their whole hearted support during the course even if they are thousands of miles away. This thesis is also to memory my grandfather, who brought me up since I was a kid and passed away last year.

I am thankful to my friends, Jincheng Zhu, Jun Zhou, Ping Wang, Fei Hao and Yu Zhou for their help in both research and life.

Finally, I would like to acknowledge that this work was partly sponsored by Overseas Research Students Awards Scheme (ORSAS), Newcastle University, School of EECE, and UK EPSRC through NEGUS Project (EP/C512812/1) and STEP Project (EP/E044662/1) respectively.

Abstract

With the fast development of semiconductor technology, more and more Intellectual Property (IP cores) can be integrated into one chip under the Globally Asynchronous and Locally Synchronous (GALS) architecture. Power becomes the main restriction of the System-on-Chip (SOC) performance especially when the chip is used in a portable device. Many low power technologies have been proposed and studied for IP core's design. However, there is a shortage of system level power management schemes (policies) for the GALS architecture. In particular, the area of using Dynamic Power Management (DPM) to optimize SOC power dissipation under latency restriction remains relatively unexplored.

Event driven programming is widely used in the design of embedded software. A task execution in an IP core is enabled only when the corresponding event, which represents the availability of resources, arrives. Therefore, the design of an efficient event handler which can quickly respond to incoming events in a highly nondeterministic and concurrent on-chip environment is essential to the improvement in system performance.

This thesis describes the work of modelling and design of an asynchronous event coprocessor to control the operations of an IP core in the GALS architecture. This coprocessor is called a Self-timed Event Processor (or STEP in short), and it provides event handling, power management as well as asynchronous communication for its cooperating IP core. The combination of one IP core and its STEP constitutes a Virtual Self-timed Block (or VSB in short).

In order to demonstrate the justification for such a scheme, stochastic models were used for power-latency analysis of a virtual self-timed block with different DPM policies. Both the event arrival and task execution of an IP core were modelled as continuous time Markov processes. The integration of mode switching transition states in the stochastic models provides accurate analysis in the research. One DPM policy named Accumulation & Fire (or A&F in short) was given a particular emphasis in this thesis not only because it has great advantage in trading latency for power, but also because it is easy in hardware realization.

A general architecture for STEP was developed from basic functional specifications. Coloured Petri Nets (CPN) was used to model the architecture of the resulting virtual self-timed block hierarchically. These CPN models focus on the concurrent processing between different components of a STEP as well as that between a STEP and its IP core, so as to improve system performance as well as avoid metastability. Functional performance of a Virtual Self-timed Block was demonstrated in simulation and verified by state space checking.

To better present SOCs composed of Virtual Self-timed Blocks, an example SOC with four virtual self-timed blocks was built in MATLAB Simulink, whose design follows the specification given in the previous CPN models. A “ball game” test bench application runs on this 4-VSB system in the MATLAB environment, showing important aspects of STEP operations such as the A&F.

Chapter 1

Introduction

1.1. IP Cores and GALS Architecture

Up to now, the evolution of digital microelectronics is characterized by the exponential growth of the number of transistors per chip which results in an exponential increase of computing power. All components of a computer system can now be integrated into a single chip, which is called System On Chip (SOC). In order to provide more functions to the on chip system and satisfy the fast manufacture and update requests of the market, chip designers prefer to integrate several predesigned and reusable hardware modules or blocks to make their new chip. These predesigned and reusable hardware modules are called IP cores or IP blocks because they are treated as intellectual properties and licensed to original equipment manufacturers (OEM). Companies that supply IP Cores, like ARM, become new highlight of the IT industry.

Although most IP Cores are still synchronously designed, more and more SOCs can not be treated as pure synchronous systems. It is not only because integrated IP Cores are designed to have different clock frequencies to optimize their performances, but also because deep transistor integration makes it hard to keep an accurate global clock system which can distribute an identical clock signal to every corner of the chip. In this case, different IP cores can keep their own clock systems and operate synchronously while communicate asynchronously with each other. These electronic systems are called **Globally Asynchronous Locally Synchronous** (GALS for short) systems [moor02]. The main concern of chip designers is to make sure signals and data among different clock islands or clock domains can be exchanged correctly and efficiently. SOC design becomes more communication centric rather than computation centric.

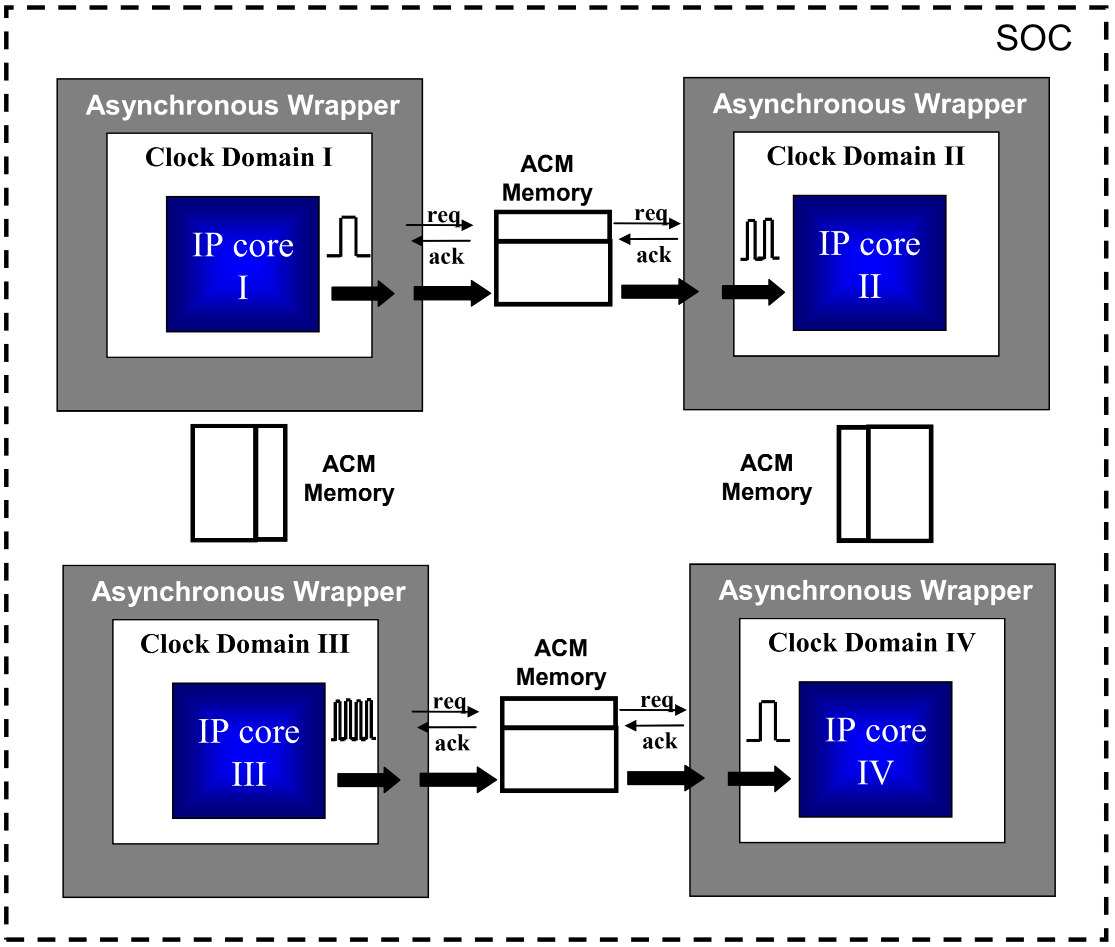


Figure 1-1: SOC with the GALS Architecture

In a GALS system, an asynchronous wrapper [zhua02] is usually added to every IP core. All synchronous signals and data generated by an IP core are first transformed into their asynchronous counterparts by the wrapper, and then be transferred to another IP core (Figure 1-1) using the Asynchronous Communication Mechanism (ACM), which was first researched by Hugo Simpson [simp90] in 1990. Most studies about GALS as well as asynchronous wrappers tried to use different sizes of buffer as well as different types of ACM to increase the throughput of asynchronous communication so as to reduce the latency of the entire GALS based SOC [dasg06].

1.2. Power Dissipation and Low Power Technologies

The large degree of transistor integration also made electronic equipments portable or wearable. Statistics show a 30% decrease in the device dimensions with each technology generation [bork99]. Portable devices such as laptops, digital cameras, mobile phones, and iPods, bring convenience and become indispensible in our everyday life. However, they must rely on batteries for power supply. Compared with the exponential integration of transistors, battery capacity has improved very slowly (a factor of two to four over the last 30 years) [simu01], which makes power the bottleneck to improve the performance of SOCs.

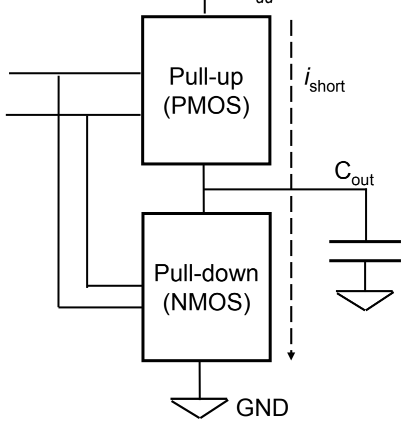


Figure 1-2: The Structure of CMOS Transistors

In the past decades, many researches have been done to reduce the power consumption of SOCs which are built by CMOS circuits. Figure 1-2 is the CMOS gate structure given in [beni98]. The pull-up network, which is generally composed of PMOS transistors, connects the output node Out to the power supply V_{dd}. The pull-down network, which is generally composed of NMOS transistors, connects Out to the ground node GND. When a transition on the inputs causes a change in the conductive state of the pull-up and the pull-down network, electric charge is transferred from the power supply to the output capacitance C_{out} or from C_{out} to ground. The power dissipation caused by transition(s) in pull-up and/or pull-down networks is called **dynamic power**. The measurement of dynamic power is given by Equation 1-1:

$$P_{dynamic} = C_{eff} V_{dd}^2 f \qquad \text{Equation 1-1}$$

In Equation 1-1, V_{dd} is the supply voltage, f is the operating frequency and C_{eff} is the effective switching capacitance of C_{out}.

Another kind of power dissipation in CMOS circuits is called **short-circuit power**. It is caused by the non-zero rise/fall time in input signal change. During this time, both the pull-down and pull-up networks are on for a short period of time and some current is drawn from the supply and flows directly to ground. This current is called short-circuit current and the short-circuit power is just the power consumed by the short circuit current.

The third contributor to the power dissipation in CMOS is the **leakage power**, which is mainly caused by two phenomena: 1) **diode leakage current** due to the reverse saturation currents in the diffusion regions of the PMOS and NMOS transistors; 2) **sub-threshold leakage current** of transistors. Therefore, the average power dissipation in a CMOS transistor can be expressed in Equation 1-2:

$$P_{ave} = P_{dynamic} + P_{short} + P_{leakage} \qquad \text{Equation 1-2}$$

In Equation 1-2, $P_{dynamic}$, P_{short} and $P_{leakage}$ represent the dynamic, short-circuit and leakage power consumptions in the transistor respectively.

Traditional low power design tries to reduce the dynamic power consumption since it gives the main contribution to the total power consumption in CMOS circuits. Technologies have been explored in different levels to change at least one of the three parameters in Equation 1-1 so as to reduce dynamic power. Transistor level low power design focuses on reducing V_{th}, the threshold voltage of CMOS, so that an SOC system can be operated by a lower supply voltage V_{dd}. Two main low power solutions have been given at the gate level. One is called **scaling**, which provides multiple V_{dd} to the same CMOS circuits in different cases. A high supply voltage $V_{dd,H}$ is connected only to CMOS circuits whose operations are critical to signal propagation, and a low supply voltage $V_{dd,\ L}$ is used to other non-critical CMOS circuits to achieve lower power consumption. When $V_{dd,\ L}$ is used in oscillator circuits, the clock frequency controlled by the oscillator circuits is reduced, and the dynamic power dissipation of the CMOS circuits controlled by the clock signal are reduced accordingly.

Another solution which tries to reduce the effective capacitance in switching is called **gating**, which stops the propagation of input signals to some part of CMOS circuits when the latter are not used. Since the clock signal is the most important control signal for synchronous circuits, **clock gating** is the most popular and widely used gating technology [jaco04]. **Data Compression** is the most popular low power technology used at the behaviour level, which tries to reduce the number of connecting wires among electronic components.

In the past decades, low power researchers have made lots of efforts to reduce the supply voltage V_{dd} (as well as V_{th}) since it contributes to the power by a square factor. However, the leakage power dissipation reduces slowly with V_{dd} decreasing, which makes it play a more important or even dominant role in the power dissipation in contemporary electric devices or chips. Furthermore, deep transistor integration brought by advanced semiconductor technology sees an increasing proportion of the leakage power in the total power consumption. For example, the sub-threshold leakage current is predicted in [aydi01] to increase from 0.01 *µA/µm* for the 130 *nm* technology to 3*µA/µm* for the 45 *nm* technology.

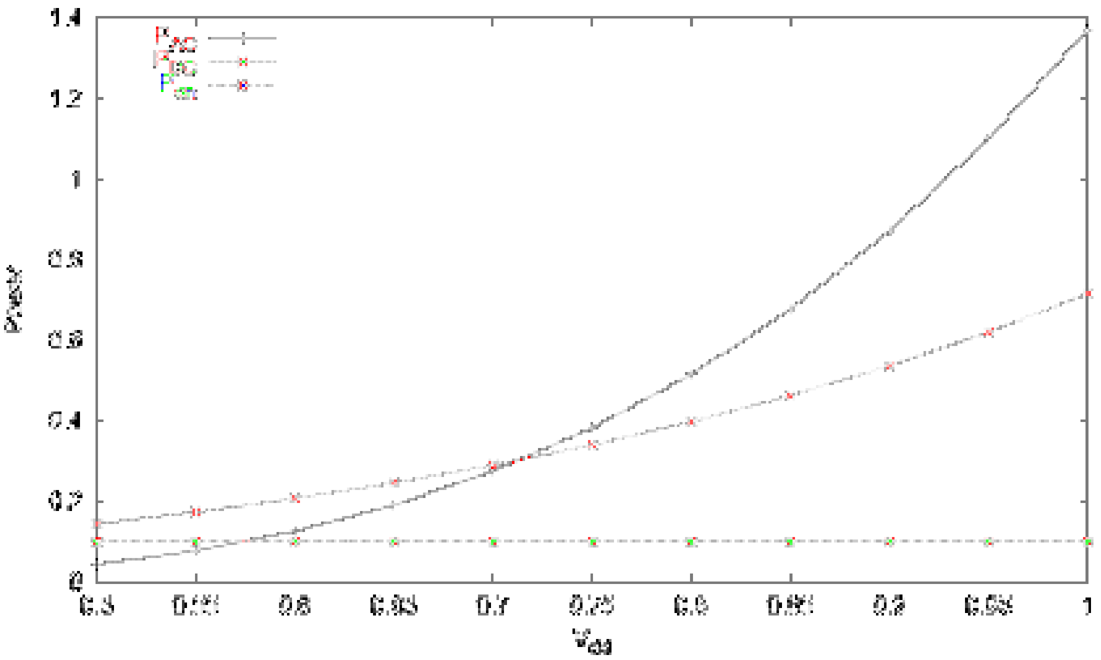


Figure 1-3: The Variation of Power Consumption with V_{dd}.

In [jiju04], the power consumption in a Transmeta Crusoe processor which is built by 0.07 *µm* technology is under test. Figure 1-3 shows the reduction in both dynamic power and leakage power with the decreasing of V_{dd} (P_{AC} in the legend is the dynamic power dissipation and P_{DC} is the leakage power dissipation. P_{on} is the power consumed by transistors that cannot be turned off.). It is clear that reducing leakage power is more important when V_{dd} becomes lower than 0.7V.

Because CMOS transistors with low a threshold voltage V_{th} have large leakage current, **multi threshold CMOS** (MTCMOS) [calh03] is a popular technique at the transistor level, which increases the V_{th} of CMOS transistors in a non-critical path so as to reduce its leakage power. At the gate level, signal gating such as clock gating can stop switching in CMOS transistors, but cannot avoid leakage current when the transistor capacitance has been charged. Therefore, **power supply gating**, which cuts down V_{dd} supply for the CMOS transistors [henz07] is used to reduce both dynamic and leakage power dissipation in the gated transistors.

All these low power technologies have been widely used in IP core design. Low power dissipation, as well as high throughput, becomes one basic performance requirement of IP cores as well as SOCs. [yseb07] describes how to use low power technologies at different levels to design a DSP microprocessor so as to satisfy the low power requirement. When several low power technologies have been integrated into the design of an IP core, operations in the IP core are changed accordingly so as to satisfy various throughputs as well as power requirements. Therefore an IP core can do its processing in various **operation modes**. For example, a sleep mode in an IP core always means all transistors in the core have been power gated. An idle mode in an IP core can be taken as the case when switches in transistors are stopped by clock gating. When different supply voltages are used to drive transistor switching, the corresponding IP core is said to use different work modes to provide service. The more low power technologies have been integrated, the more operation modes can be provided by an IP core. Multiple mode IP cores are widely adopted in both Hard Disk Drivers (HDDs) (like IBM Travelstar [ibmt97], FUJI MHF 2043AT [lu00]) as well as microprocessors (like SA1110 [sa11], Transmeta Crusoe [jiju04]). When some low power technology is enabled or disabled, the corresponding IP core is said to switch to another operation mode.

One thing needs to be highlighted here is that operation mode switching transitions bring **overheads** in both power and latency. For example, a microprocessor entering its sleep mode needs three steps: 1) flush to memory all system information that should be preserved throughout the sleep period; 2) reset all internal processor register and program wakeup events; and 3) shutdown the internal clock generator. Similarly, three steps are taken when the processor is switching back to its work mode: 1) turning on and stabilizing the power supply and the clock; 2) reinitializing the system; and 3) restoring the context. The possible high overheads in mode switching means frequent mode switching cannot benefit, and may even deteriorate an IP core in its power dissipation. Therefore when several IP cores are integrated into one SOC, some scheme or policy is needed at the system level to manage mode switching in all component IP cores so as to minimize not only the power dissipation in every power domain, but also that of the entire SOC. The group of circuits where this policy is implemented is often called a **power manager**. When only one power manager is used to control all IP cores, it is called a centre power manager. Otherwise, several distributed power managers are used to provide power control in different power domains.

System level low power technologies can be generally divided into two groups: one group tries to make an IP core to carry on its operation in its full power and then switch the latter to stay in one of its low power modes (such as sleep mode) as long as possible. This kind of technology is often called **Dynamic Power Management (DPM)**. The other group of technologies tries to make an IP core do its operation as slowly as possible. The IP core can only switch to some mode with faster operation speed when the corresponding latency cannot be tolerated. This kind of technology is often called **Dynamic Scaling**, which includes Dynamic Voltage Scaling (DVS), Dynamic Frequency Scaling (DFS), and Dynamic Voltage and Frequency Scaling (DVFS) depending upon which parameter(s) can be scaled. Delicate DPM/DVS design can reduce both dynamic and leakage power in an IP core [jeju03].

However, system level power management has not been considered by SOC with GALS architecture, although this architecture has great potential in power saving. Without a global clock system, an SOC built in the GALS architecture can easily power on/off an IP core or switch it to another mode without interfering with the clock propagating to other IP cores.

Furthermore when low power dissipation is concerned, asynchronous circuits show great advantage over synchronous ones. By eliminating the clock system, which always has the largest capacitance and switching frequency in the chip, asynchronous circuits can do the same operation as their synchronous counterparts with extremely low power. That is why asynchronous technology is claimed as a “revolutionary” low power technology in [beni98]. Therefore, an asynchronous power manager can provide power control to its IP core with extreme low power overheads.

1.3. Event Driven System and STEP

The increase in the degree of transistor integration brings changes not only to hardware design but also software design. On the one hand, executions in an IP core become multiple processing. More and more **tasks**, which represent operation threads, can be embedded into one IP core. On the other hand, 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 one 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 of **resources**, which represent limited battery energy, finite memory space or communication bandwidth, etc. Nondeterminism in task execution brings great challenge in the area of fast or real-time resource allocation.

In this case, event driven programming is preferred to 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 a request signal or data, or idleness of input/output ports, or enough energy in the battery, depending on different implementations. Therefore, event handling can be taken as resource allocation in an SOC. After some event is handled by an IP core, its corresponding task can be carried out.

In an event driven system, a task, when allowed to run, must return control when it completes or when it cannot be executed further. In other words, the task cannot perform an operation which would cause execution to suspend within that task. If the task was half way through an operation and was waiting on more resources such as data, it would need to remember where it was and return. When the resource that the task was waiting on arrived, the task would then continue from where it had previously stopped. A scheduler or dispatcher is used to allow other tasks to run when the execution of the current task is completed or stopped.

When on chip nondeterminism and concurrency are taken into consideration, the synchronous or software based event handler and scheduler designed by previous research cannot satisfy the requirement of SOC.

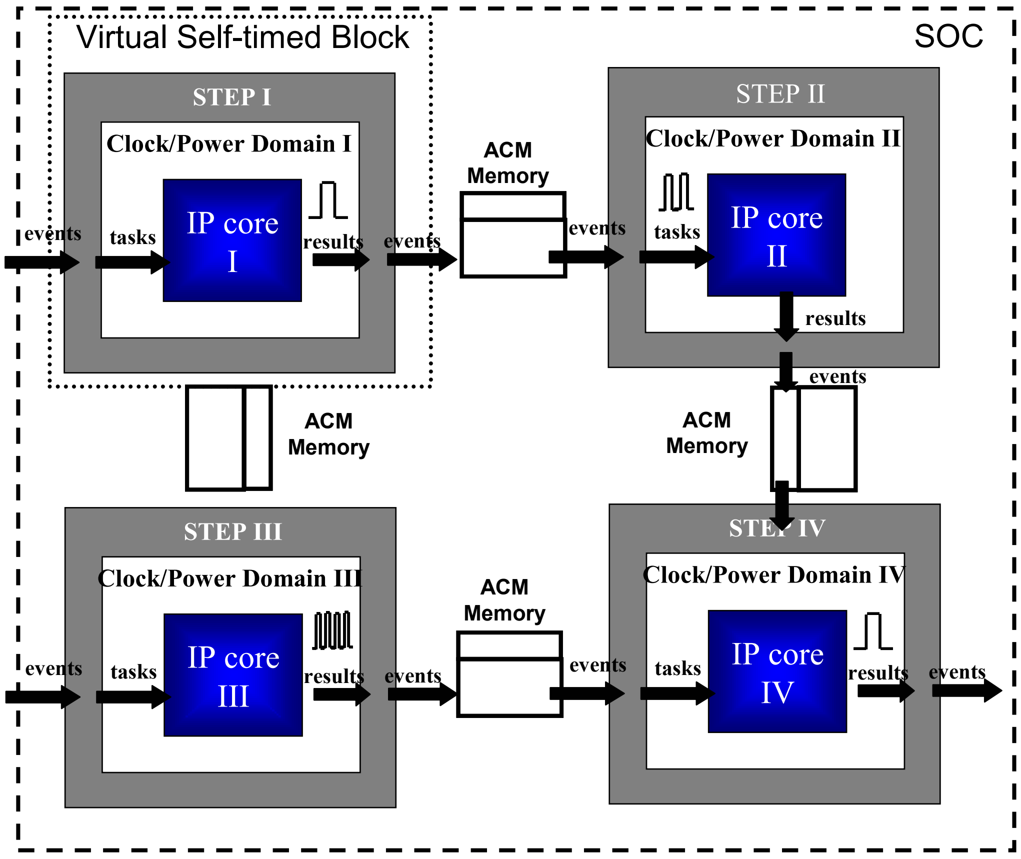


Figure 1-4: Self-timed Event Processor and Virtual Self-timed Block

First of all, the operation of synchronous circuits (as well as the software running in synchronous circuits) is controlled by clock signals. When several events come within one clock cycle, they are taken as simultaneous by synchronous circuits and can only be handled in the next clock cycle. If arbitration [kinn07a] involves due to the unnecessary accumulation of events, synchronous circuits may use hundreds of clock cycles to solve it and the corresponding latency may greatly deteriorate the performance of IP cores as well as the entire SOC. Secondly, the nondeterminism of event arrival means circuits for event handling cannot be powered off at any time. Although the power consumption in these circuits may be trivial, given a high enough frequency of the IP core and enough time, the total energy cost in these circuits cannot be ignored.

On the other hand, asynchronous circuits have a great advantage in event handler and scheduler design. Without clock control, an asynchronous event handler can respond to new incoming events without delay, and the chance of metastability should be greatly reduced. Similarly since no power is wasted in the handler (and scheduler) when no change happens in the system, an asynchronous handler can fulfil its job while keeps an energy hungry IP core in its sleep mode when no task is enabled.

Therefore, an asynchronous coprocessor rather than a simple asynchronous wrapper is necessary to be used in a GALS based SOC content. This coprocessor helps IP cores to work as event driven domains in a highly nondeterministic and concurrency environment with limited power. This coprocessor is called a **Self-Timed Event Processor** (or STEP in short) and the combination of one STEP with its processor works as a **"Virtual" Self-timed Block** (or VSB in short) in the GALS architecture (Figure 1-4). The main function modules of a STEP are as follows:

1) An asynchronous wrapper which can realize asynchronous/synchronous signal and data conveyance

2) A power manager where a system level DPM policy is used to reduce the power

consumption while not seriously deteriorating system throughput. Since low power technologies have been integrated into IP core design, the power manager in a STEP provides not low power circuit realization but low power commands to an IP core. In other words, the power manager dynamically adjusts the IP core to use a proper operation mode according to the environment situation.

3) An event handler which can quickly respond to the incoming events and enable their corresponding tasks. A memory about what tasks have been enabled by the corresponding events (if it has not been executed yet) and which tasks have been stopped due to lack of resources is kept in the handler.

4) A scheduler who chooses a task from all candidates for processing in an IP core when the execution of the current task is completed or terminated.

The main contributions of this thesis are as follows:

1) To present the architecture of an asynchronous designed Self-timed Event Processor where asynchronous communication, power control and event handling are taken into consideration.

2) To obtain analytical solutions for stochastic models of DPM systems which for the first time allow an infinite number of system states in mode switching transitions. The achievement of an analytical solution enables a more accurate estimation of the power/latency performance in an IP core with DPM control.

3) To present Fine Grain models for DPM systems for the first time which does not take the cost in power manager circuits as cost free.

4) To present a thorough analysis of the implementation of Accumulation & Fire policy with different kinds of IP cores. Both power efficiency and applicability of this policy have been explored so as to prove its great potential in power saving.

5) To model the structure of a Virtual Self-timed Block with the modelling tool of Coloured Petri Net (CPN) where all nondeterministic and concurrent processing in a VSB has been modelled and proved by simulation and state space checking.

6) To present the construction of a SOC with VSBs in MATLAB Simulink. A test bench named as ball game was designed for the analysis of a VSB performance in a real implementation.

1.4. Thesis Outline

The rest of the thesis is organized as follows:

Chapter 2 first categorises previous studies about DPM policies into two classes: prediction policies and stochastic policies. When the overheads in mode switching transitions of a processor are highlighted, a new policy named Accumulation & Fire becomes promising to increase the power efficiency of a processor. This chapter also introduces different modelling languages (tools) that are used in subsequent chapters of this thesis.

Chapter 3 is about stochastic models for power-latency analysis of a VSB when different DPM policies are used. In these models, both events incoming and task executions of an IP core are modelled as continuous time Markov processes. The integration of mode switching transition states in the stochastic models increases the accuracy in our analysis. Three kinds of DPM systems, named as On-off DPM systems, DPM systems with multiple inactive modes and DPM systems with multiple active modes, have been modelled and analyzed. One DPM policy named as Accumulation & Fire (or A&F in short) is highlighted in this chapter not only because it has great advantage in trading latency for power for all DPM systems, but also because its easy hardware realization.

Chapter 4 presents the modelling work of a VSB modelled in Coloured Petri Nets (CPN). These CPN models focus on the concurrent processing between different components of a STEP as well as that between a STEP and its IP core, so as to improve system performance as well as avoid metastability. State space checking is used to verify the correctness of CPN models.

Chapter 5 describes the implementation of an example SOC system with four virtual self-timed blocks in MATLAB Simulink. Simulation results are provided when a test bench named ball game is running in the model system.

Chapter 6 concludes this thesis and suggests some ideas for further studies.

Chapter 2

Background

2.1.Research in System-Level Dynamic Power Management

With rapid progress in semiconductor technology, portable devices are enabled with sophisticated processing capability and can provide services that were only available in desktop computers decades ago. However, the high frequency and chip density in new designs not only bring fast execution performance, but also make battery-based systems more energy hungry. Therefore, low-power design which tries to reduce the power dissipation while still satisfying the latency requirement becomes a hot research topic in the electronics field.

System level energy-saving technologies focus on increasing the energy efficiency in portable devices. **Dynamic Power Management (DPM)** [beni98], for example, provides power on-off control to a portable device whose computation units are event-driven for reactive processing. These units are activated and can access the battery power only when they are triggered by some external events to carry out the corresponding tasks, and a so-called power management (PM) unit is added to the system where some scheme (policy) is implemented to decide when and how to shutdown or wakeup certain units. These computation units are called **power manageable components** (**PMCs**) [beni00].

In system level, PMCs are modelled as black boxes. As an atomic block in power management execution, the detail of task executions in a PMC is ignored by DPM control. Instead, the (multiple) **modes of operation** that can be controlled for power-latency trade-off are the fundamental characteristics of a PMC. A **mode switching transition** command is issued by the PM when the current mode in the PMC cannot minimize the power dissipation under certain latency constraints (or when the PMC cannot minimize the latency under tolerable power dissipation). If a PMC only has two operation modes: **on** and **off**, the corresponding mode switching transitions are called **shutdown** (from mode on to off) and **wakeup** (from mode off to on) respectively.

2.1.1. Cost of PMC Mode Transitions

As introduced in Section 1.2, mode switching transitions in PMCs have costs in both power and latency. In most cases, the lower the power dissipation one mode can provide, the longer latency and higher power dissipation are paid in the switching transitions from/to the mode. Therefore, a switching transition to a mode with lower power dissipation should only be carried out when the energy saved (or latency improved) by the mode can compensate for that consumed in the corresponding mode switching transition.

The concept of **Break-Even Time** (T_{BE}) is defined in [beni00] and [lu00] to measure the energy cost caused by a mode switching transition. The Break-Even Time for switching from mode i to mode j in a PMC is defined as the minimum time spent in mode *j* to compensate for the cost of entering this mode. Therefore, a switching transition from mode *i* to mode *j* is carried out only when the PMC can spend at least T_{BE} in mode *j*.

If T_{in} and T_{out} are defined as the time cost for the switching from mode *i* to mode *j* and visa versa respectively, T_{TR} as the **transition time** is the sum of T_{in} and T_{out}.

$$T_{\mathrm{TR}} = T_{\mathrm{in}} + T_{\mathrm{out}} \qquad \text{Equation 2-1}$$

The average **transition power dissipation** P_{TR} can be expressed as

$$P_{TR} = \frac{T_{in}P_{in} + T_{out}P_{out}}{T_{TR}} \qquad \text{Equation 2-2}$$

P_{in} and P_{out} in Equation 2-2 are the power cost spent in the transition to and from Mode *j* respectively. If P_i and P_j are defined as the power dissipation of mode *i* and *j* (P_i>P_j), T_{BE} can be expressed as:

$$T_{BE} = T_{TR} + T_{TR}\frac{P_{TR} - P_i}{P_i - P_j} \qquad \text{if } P_{TR} > P_i \qquad \text{Equation 2-3}$$

$$T_{BE} = T_{TR} \qquad \text{if } P_{TR} \le P_i$$

2.1.2. Predictive DPM Policies

The T_{BE} values of a PMC are of great importance when some predictive DPM policy is used. Predictive DPM policies turn a PMC to one mode with lower power dissipation if the PMC is predicted to stay in the mode long enough. These policies use “the correlation between the past history of the workload and its near future in order to make reliable predictions about future events” [beni00].

The simplest prediction policy is called **timeout** policy, which does a mode switching transition (for example, shuts down the PMC) after a fixed **idle** time (represented by the parameter τ) during which no executions are carried out in a PMC. The PMC is switched back to its previous mode when a new event arrives. This policy relies on the assumption that a PMC is highly likely to remain idle if it has been idle for some time. Although simple, this policy is widely used for many laptops and other portable devices [lu00].

Therefore, if T_{idle} is the total time span of the idle period in a PMC, the timeout policy shuts down the PMC when $T_{\mathrm{idle}}>\tau$ and can only save the power of the PMC if $T_{\mathrm{idle}}>\tau+T_{\mathrm{BE}}$. The choice of τ value in different PMCs relies on the conditional probability of $Q(T_{idle,pred} >\tau+T_{\mathrm{BE}} \mid T_{\mathrm{idle}}>\tau)$. In order to differentiate power from probability, alphabet $\boldsymbol{P}$ is only used for power and $\boldsymbol{Q}$ is used to represent probability or probability distribution in this thesis. $T_{idle,pred}$ here means the predicted length of T_{idle}. If $T_{idle,pred} > \tau+T_{\mathrm{BE}}$ while $T_{\mathrm{idle}}<\tau+T_{\mathrm{BE}}$, the timeout policy increases rather than reduce the power dissipation of a PMC. If a predicted idle period is longer (shorter) than the actual one, it is called over-prediction (under-prediction). An over-prediction means the corresponding DPM control worsens rather than improve the power efficiency in a PMC, and an under-prediction means the energy saving that can be brought by the corresponding DPM control is not fully used. In [beni00], two measurements are used to reflect the quality of a PM when some predictive policy is used. The concept **safety** is defined as the complement of the risk of making over-predictions, and **efficiency** is defined as the complement of the risk to make under-predictions. In this thesis, we redefine the two concepts to give them wider description. The concept safety is defined as the percentage of the risk of making energy worse, and efficiency is defined as the percentage of power saving compared with the original power dissipation.

The timeout policy has two main advantages: it is general enough to be implemented in almost all portable devices, and its prediction safety can be improved simply by increasing the τ value. However, its limitation is obvious as well. It trades safety against efficiency, and the power is wasted during the τ period. Therefore, advanced prediction policies try to improve the prediction efficiency without much loss of safety.

Some policies try to make a mode switching decision as soon as the idle period begins, so that no energy is wasted in the τ period (since τ=0). These polices are called **predictive shutdown** polices in [beni00]. If an active period is defined as the period when a PMC is doing processing, it happens alternately with an idle period. In the end of the (*n*-1)th active period, the PM makes a prediction about the length of *n*th idle period based on the history data:

$$T_{idle\_pred}^{n} = F(T_{active}^{n}, T_{idle}^{n-1}, T_{active}^{n-1}, \ldots, T_{idle}^{n-k}, T_{active}^{n-k-1}) \qquad \text{Equation 2-4}$$

Different functions *F*() are used by different policies to make the prediction safer or more efficient. For example, the nonlinear regression equation in [sriv96], α adaptation in [hwan97] and Artificial Neural Network in [lu06] are used as the *F*() in their own implementations. A mode switching (shutdown) transition command is issued when $T_{idle,pred}^{n}$ >T_{BE}.

Besides the prediction function *F*(), the prediction safety and/or efficiency also relies on the amount of history data used for prediction (the parameter *k* in Equation 2-4). The use of a large amount of history data makes the PM circuits' complex, and somehow counteracts the power saved by DPM control. A threshold method is given by [sriv96] which shuts down a PMC when the ending active period is shorter than some threshold ($T_{active}^{n-1} < Threshold$). This policy can only be used in PMCs whose active periods are in L-shaped, i.e., a short active period in a PMC is often followed by a long idle period.

Other prediction policies, which focus on wakeup transitions, are always called **predictive wakeup** policies. These policies also predict the length of T_{idle} in Equation 2-4, and a PMC is woken up as soon as $T_{idle,pred}$ is reached. Compared with predictive shutdown policies, these wakeup policies focus more on latency rather than power dissipation in a PMC.

All these policies try to improve the power efficiency of a PMC without much deterioration of the safety. However, there is no easy trade-off between efficiency and safety in these policies, like the timeout policy. The prediction efficiency highly depends on the correlation between continuous idle and active periods. Therefore, their efficiency varies greatly in different implementations and simple variations in parameters (like the k in Equation 2-4) do no guarantee prediction efficiency and/or safety improvement. Therefore comparative simulation is indispensible in deciding whether and which prediction policy can be implemented in one particular PMC.

A timer is indispensible in the PM for all kinds of prediction policies. Although the timer is only used casually to record the time τ when the timeout policy is used, this counter is always used to record the length of T_{active} and T_{idle} for the predictive wakeup policies. In these cases, the timer works as a clock in the PM, which is energy hungry especially when an accurate record of T_{active} and T_{idle} is needed. Besides, the PM can hardly be taken as cost free, as claimed by most research about prediction policies.

2.1.3. Stochastic DPM Policies

Although highlighting the uncertainty (as well as the correlation) of the workloads, predictive policies always assume deterministic response and transition times for a PMC. However, the abstraction of a PMC as a black box makes this assumption doubtful, and the other group of DPM policies, named as stochastic DPM policies, prefers to model the execution in a PMC as a stochastic process as well. Rather than trying to eliminate uncertainty by prediction, these policies try to take the DPM control as an optimization problem under uncertainty. Therefore, these polices have wider implementations in different PMCs to satisfy the power-latency trade-off required by the environment or the users.

In most of these policies, both the workload and the PMC execution are taken as Markov Processes and Markov Based Model is used to describe the processing in a PMC under DPM control. The fitness of Markov Processes modelling portable devices have been demonstrated in [simu00] where three experiments, with a hard disk for a laptop, a personal communication interactive device and a WLAN card, are carried out respectively.

2.1.3.1 Stochastic Model of DPM Systems

A **stochastic process** is a family of random variables $\{X(t), t \geq 0\}$ where t is the time parameter. The values assumed by the process are called the **states**, and the set of possible values is called the **state space**. A stochastic process $X(t)$ is called a **Markov Process** if for any set of time $t_0 < t_1 < \ldots < t_n < t$, its conditional distribution has the property:

$$Q[X(t) \leq x \mid X(t_n) = x_n, \ldots, X(t_0) = x_0] = Q[X(t) \leq x \mid X(t_n) = x_n] \qquad \text{Equation 2-5}$$

Where $t_0, t_1, \ldots, t_n, t \in \boldsymbol{T}$ and $x_0, x_1, \ldots, x_n \in \boldsymbol{S}$. $\boldsymbol{T}$ and $\boldsymbol{S}$ are called the parameter space and state space of the Markov process respectively. When both $\boldsymbol{T}$ and $\boldsymbol{S}$ belong to discrete space, the Markov process is called the **discrete-time Markov process**. When $\boldsymbol{T}$ is a continuous space and $\boldsymbol{S}$ is a discrete space, the Markov process is called the **continuous-time Markov process**. A Markov model of a DPM system consists of three parts (Figure 2-1):

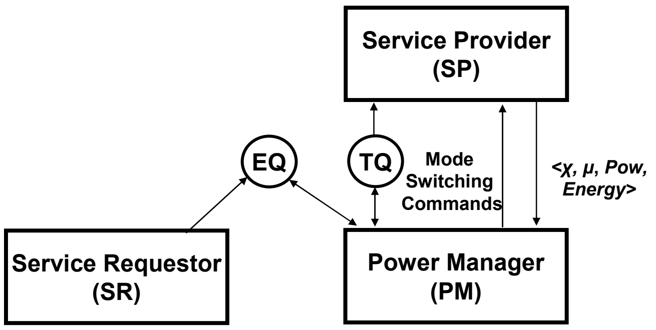


Figure 2-1: The Architecture of a DPM system

A **service requester** (SR) is a Markov process with state set $\boldsymbol{R}$ which models the arrival of service requests in the system (i.e., the workload of events).

A **service provider** (SP) is a Markov process which models a PMC where r operation modes are provided. Transitions among these states are stochastic, which are controlled by commands issued by the power manager. An SP responds to the incoming events from the SR by execution of their corresponding tasks. All tasks waiting to be executed are saved in a task queue (TQ) and the SP fetches new task from the TQ by FCFS (First Come First Serve).

In [qiu99], the description of the SP has been specified as a set group $<\boldsymbol{\chi}, \boldsymbol{\mu}, \boldsymbol{Pow}, \boldsymbol{Energy}>$. If $\boldsymbol{M}=\{M_i \mid i=0,1,2,\ldots,r-1\}$ is the operation mode set in an SP, we have:

$\boldsymbol{X}$ is an $r \times r$ matrix called the transition rate matrix of the SP. The $\chi_{i,j}$ $(i<r, j<r)$

component in the matrix represents the transition rate from M_i to M_j (the transition is written as $M_{i,j}$ later). Since the switching from M_i to itself is instantaneous, $\chi_{i,i}$ is set to ∞. In case a mode M_j cannot be switched directly from another mode M_i, the corresponding $\chi_{i,j}$ is set to 0.

M is an r vector and μ_i stands for the mean service rates of the SP when it is in mode M_i.

Pow is an r vector and Pow_i is the power consumption in the SP in mode M_i.

Energy is an $r \times r$ matrix and $Energy_{i,j}$ indicates the energy cost of $M_{i,j}$. $Energy_{i,i}$ is set to 0 because no extra energy is needed for the SP to keep in the same mode.

A **power manager** (PM), which issues mode transition commands (***Cmd***) to the SP following the function f: $\boldsymbol{M}\times\boldsymbol{R}\rightarrow\boldsymbol{Cmd}$. This represents a decision process: the PM observes the mode in the SP as well as the workload, makes a decision and issues a command to the SP so as to control the future status of the system. The execution in the PM is generally taken as cost free in both power and latency.

Given L is the maximum length of the TQ, vector $\boldsymbol{Len}=\{0,1,\ldots L\}$ represents the variable length of the TQ. When $\boldsymbol{T_r}$ represents the set of mode switching transitions $\boldsymbol{T_r}=\{M_{i,j}\ \ i\neq j\}$, the full state space of a DPM system can be represented by $\boldsymbol{Sys}=\boldsymbol{R}\times(\boldsymbol{M}+\boldsymbol{T_r})\times\boldsymbol{Len}$. A DPM policy $\boldsymbol{\pi}$ is the set of commands that is issued by the PM when the system stays in any of its states $\boldsymbol{Sys}\times\boldsymbol{Cmd}\rightarrow\boldsymbol{\pi}$.

In a simple example, we assume that only one requesting mode exists in the SR (The average interval time of requests generated by SR follows the exponential distribution with mean value $1/\lambda$), and the SP has two operation modes: M_1 for mode on and M_0 for mode off. L=1 is the maximum length of TQ, the full state space of the corresponding DPM system has $\boldsymbol{R}\times(\boldsymbol{M}+\boldsymbol{T_r})\times \boldsymbol{Len}$ = $1\times(2+2)\times2$ = 8 states. We use $<M_i, n>$ to represent the state when the operation mode in SP is M_i and the TQ length is n. Suppose two commands can be given by the PM: {wu, sd} (wu stands for wakeup and sd stands for shutdown). A DPM policy can be expressed as follows:

Sys	$(M_0,0)$	$(M_0,1)$	$(M_{0,1},0)$	$(M_{0,1},1)$	$(M_1,0)$	$(M_1,1)$	$(M_{1,0},0)$	$(M_{1,0},1)$
wu	0	1	1	1	0	0	0	0
sd	0	0	0	0	1	0	1	1

2.1.3.2 The Generator Matrix and the Probability Distribution

A system generator matrix $\boldsymbol{G}$=$\boldsymbol{Sys}\times\boldsymbol{Sys}$ is kept to record state transitions according to the new arrival of events, the execution carried in the SP as well as commands given by the policy. If $Q_{i\Rightarrow j}(t)$ is the transition probability from state i to state j during time 0 to t and $Q^{'}_{i\Rightarrow j}(t)$ is its derivative, a generator matrix $\boldsymbol{G}$ is shown like below:

$$G = \begin{pmatrix} -\sigma_{0,0} & \sigma_{0,1} & \sigma_{0,2} & \cdots \\ \sigma_{1,0} & -\sigma_{1,1} & \sigma_{1,2} & \cdots \\ \sigma_{2,0} & \sigma_{2,1} & -\sigma_{2,2} & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix}$$

A unit $\sigma_{i,j}$ in $\boldsymbol{G}$ is called the **transition rate** from state i to state j which is calculated by Equation 2-6 or Equation 2-7:

$$\sigma_{i,i} = \lim_{t\to0}\frac{1-Q_{i\Rightarrow i}(t)}{t} = -Q^{'}_{i\Rightarrow i}(0),\ i=1,2,\ldots,Sys \qquad \text{Equation 2-6}$$

$$\sigma_{i,j} = \lim_{t\to0}\frac{Q_{i\Rightarrow j}(t)}{t} = Q^{'}_{i\Rightarrow j}(0),\ i,j=1,2,\ \ldots,\ Sys;\ i\neq j \qquad \text{Equation 2-7}$$

According to queuing theory [klei75], for a continuous Markov process, a state i is said to be **recurrent** if and only if, starting from i, eventual return to this state is certain. A recurrent state is said to be **positive recurrent** if and only if the mean time to return to this state is finite. A state *i* is said to be **transient** if and only if, starting from *i*, there is a positive probability that the process may not eventually return to this state.

State *j* is said to be accessible from state *i* if *j* can be reached from *i* within finite time, which is denoted as $i \rightarrow j$. If $i \rightarrow j$ and $j \rightarrow i$, they are **communicate**, which is denoted as $i \leftrightarrow j$. The set of all states of a Markov process that communicate with each other forms a **communicating** class. If the set of all states of a stochastic process X form a single communicating class, then X is **irreducible**.

If the Markov process is irreducible, the limiting distribution $\lim_{t\to\infty} Q_i(t) = Q_i, i \in Sys$, exists and is independent of the initial conditions of the process. The probability distribution among all states ***Sys*** is given by the unique solution of the equation: $\boldsymbol{Q}\mathbf{G} = 0$ and $\sum_{i\in Sys} Q_i = 1$ where $\boldsymbol{Q} = (Q_0, Q_1, \ldots)$.

Therefore, if we use $\boldsymbol{G}^{\pi}$ to indicate the generator matrix of the current DPM system when a certain policy $\boldsymbol{\pi}$ is implemented, this matrix can be used to derive the probability distribution among all states. In this case, the power cost can also be defined as:

$$C^{\pi}_{pow} = \sum_{i\le r, n\le L} Q_{M_i,n} Pow_i + \sum_{i,j\le r, n\le L, i\ne j} Q_{M_{i,j},n} Energy_{i,j} \chi_{i,j} \qquad \text{Equation 2-8}$$

In Equation 2-8, $Q_{M_i,n}$ and $Q_{M_{i,j},n}$ represent the probability of state $<M_i,n>$ and $<M_{i,j},n>$ respectively. In many works like [beni99, qiu99, beni00, ren05], the average length of TQ has been used as the measurement of the system latency, therefore, the latency cost can be defined as:

$$C_{latency}^{\pi} = \sum_{n=0}^{L} nQ_{M_i,n} + \sum_{n=0}^{L} nQ_{M_{i,j},n} \qquad \text{Equation 2-9}$$

If **Π** is the set of all possible policies *π,* an optimized policy π_{opt} is a policy that can minimize C_{pow} within certain latency restriction *D*:

$$C_{pow}^{\pi_{opt}} \leq C_{pow}^{\pi}, \pi, \pi_{opt} \in \Pi \qquad \text{Equation 2-10}$$

$$C_{latency}^{\pi_{opt}} \leq D \qquad \text{Equation 2-11}$$

This policy optimization has been described in detail when the system is modelled as discrete Markov processes [beni99] or continuous Markov processes [qiu99] respectively.

In all these papers, authors modelled mode switching transitions as Markov processes in their models. It is first because these transitions are task dependent. For example, the shutdown transition needs to save all system information to the memory before gating the power supply to go into sleep mode. Therefore, the time and power cost in this transition highly depends on the amount of system information involved, which varies from task to task. This can be supported by Table 2 in [mihi04], which gives different time costs for the shutdown and wakeup transitions (called A-S-A time in the paper). Furthermore, the nondeterministic characteristic of an on board battery is another important factor for the stochastic execution of mode switching transitions. Recent papers [chia01, luci08] have disclosed that the energy consumption in a battery is not linear, and the energy may be partially recovered after some idle period. All these factors give a more nondeterministic character to the mode switching transitions.

One thing that must be emphasized here is that the optimized policy chosen by this method is highly parameter sensitive. In other words, whether the policy can achieve the lowest power consumption highly depends on whether the workload parameter (λ) as well as the SP (like μ, χ) used in the system level model is accurate or not.

Some research has been done to increase the robustness of the optimized policy. For example, Hidden Markov Process in [tan08] and Partially Observable Markov Process [qiu07] are used to give a better description of workloads whose variability may be hidden or partially observable. Others [chun99] keep several optimized policies in the PM. Each policy responds to a different group of possible parameters, and the PM can dynamically change to a new policy when the environment varies. However, considering circuit design and real system implementation, the robust optimized policy achieved by these methods still faces problems in the implementation, as shown below:

First of all, the safety of DPM policy is totally unexplored. These optimized policies are trying to increase the power efficiency. However they do not measure the possible unsafeness. If some unsafeness happens because of environmental changes, there is no easy way to trade power efficiency for safety.

Secondly, it is difficult to adjust trade-off between power and latency. It is clear that all power and latency requirements are implementation oriented and can be changed from time to time. Therefore, the user prefers a DPM policy that can easily realize different power latency tradeoffs without much change in the PM circuits. However, every optimized policy is only for one particular latency (or power) restriction, and it is not easy to find an optimized policy to fit the requirement of a new latency (or power) restriction.

Thirdly, hardware circuits for the robust optimized policy may be very complex. The more advanced Markov Processes are used to model the behaviour of a DPM system, the more complex the corresponding circuits may be. If designers want to implement several optimized polices for different parameter sets in their PM, the corresponding circuits may increase in size several times since no evidence is given in these papers that any two optimized policies have similarities.

Finally, it may involve a lot of possible redesign work for different implementations. Since every optimized policy is effective only for one parameter set, the PM needs to be redesigned every time the optimized policy is changed due to parameter changes. No evidence shows this redesign work can be easily carried out.

All these reasons give us enough motivation to look for a sub optimized but easily implemented policy for the real implementation. This policy should take both policy efficiency and safety into consideration. It can realize an easy (online or offline) adaptation for a new power latency trade-off. Besides, its realization in the PM should be simple to keep the power overhead of the PM as low as possible.

2.1.4. Accumulation & Fire Policy

Generally speaking, one and only one command can be issued by the PM when the SP is carrying out some mode transition (For example, only command wu can be issued when the SP is carrying mode transition $M_{0,1}$). Therefore, the main difference between various policies π lies in how many tasks are in the TQ when the mode transition commands are issued. If no less than one task is still left in the TQ when the shutdown command is issued, it is called **pre-shutdown** in this thesis, and if more than one task is available in the TQ before the wakeup command is issued, it is called **accumulated wakeup**.

Although it may help an SP to reduce its power consumption, the pre-shutdown method is not considered in this thesis because it may cause great latency for some task. For example, if the PM decides to shut down its SP when the latter has no more than two tasks in the TQ, the execution of the last two tasks must wait until the SP is woken up again which may cause serious latency to these two tasks. Although FCFS is considered in high level DPM model, many real systems prefer to use a priority based policy for scheduling. In these cases, tasks with the lowest priority are scheduled to be executed as the last one, even when they may come earlier than some high priority task. If the pre-shutdown method is used, the low priority task may NEVER get the chance to be executed in the SP because the SP is always shut down before they are chosen to be executed.

Our main focus is on the accumulation wakeup method because it is similar to the **integrate and fire** mechanism found in biological neural systems [buzs04]. One biological neuron may generate its own stimulation pulse when it receives enough stimulation pulses from other neurons. This mechanism helps many biological creatures to complete some complex tasks with extremely low power compared with electronic computers or chips. If this method uniquely used as a DPM policy, it is called **Accumulation and Fire policy** or **A&F policy** in short in this thesis. This policy is also referred as ***N*-policy**, which was first studied by Yadin and Noar [yadi63].

Because of the simplicity of the A&F policy, analytical solutions for probability distributions can be achieved even when an infinite number of states is involved. Although the powerful calculation capability of modern computers makes the numerical calculation involved in the $\boldsymbol{QG}$=0 much easier than before, the analytical solution of a Markov model (if the latter has) has a great advantage over its numerical counterpart. First of all, the analytical solution can directly reflect the influence of parameters on the probability distribution while the numerical solution cannot. When multiple parameters can be changed to achieve better performance, the analytical solution can indicate to the designer which parameter adjustment is the most effective. Secondly, the accuracy of probability distribution calculated by the numerical method depends on the number of states involved in the calculation. When only a small number of states is used for calculation, the inaccuracy in the solution may be too high to reflect the basic properties of the corresponding process. However, one can hardly know in advance how many states are enough to reflect the properties of the process. On the other hand, the accuracy of analytical solution is independent of the number of states. Finally, if the numerical solution use as many as possible states for calculation, the calculation complexity increases by n^2 while that for the analytical only increases by n where n is the number of states.

Since the A&F policy has great potential in power saving and can have analytical solutions for more accurate analysis, many investigations have been carried out about this policy. On the one hand, many queuing theory models have been used by mathematical studies. To be more specific, if the SP (the server in queuing theory) can be turned off from service providing sometimes, it is called “*N*-policy with single removable server”. Hersh and Brosh [hers80] used the M/M/1 model, Teghen [tegh87] used the M/G/1 model, and Wang and Huang [wang95, pear04] used the M/E_k/1 model to analyze the probability distribution in *N*-policy. The latest research about the *N*-policy can be seen in [chou04, thom08]. Some optimization in *N*-policy is also provided accordingly. However, these policies all take the server turn-on/off transition as instantaneous and cost free, which makes their analysis of *N*-policy not suitable for DPM implementations. Furthermore, all these studies limit themselves to deal with simple SPs with only on/off modes. (Queuing theory uses *T*/*S*/*N* to describe different queueing systems. *T* indicates the type of stochastic process for incoming customers, S indicates the type of stochastic process for service providing and *N* represents the number of servers in the system. For example, M/M/1 describes a queuing system when both incoming customers and service in the server follows Markov Processes and only one server exists in the system. For the meaning of other abbreviations, one can use [klei75] for reference.)

On the other hand, electronic engineers also make attempts to implement this policy. For example, in the ultra low power DSP processor designed for electrocardiogram (ECG) applications [yseb07], 50 ECG samples are accumulated before activating the DSP for processing. However, no mathematical analysis is given in the paper and the accumulation limit of 50 is purely based on simulation results. Some models have been built for the A&F analysis, while the models they have used seem not accurate enough to reflect the DPM system behaviour. In the following sections, we use the model given by [ren05] and [wang95] to demonstrate the previous research about the A&F policy in both electronics and mathematics studies.

2.1.4.1 State Transition Diagram

When one and only one task is accumulated before the activation of an SP, the A&F policy in this case is called the **greedy** [ren05] or **eager** policy [beni99], which serves as the simplest stochastic policy of DPM. The analytical estimation of the average power consumption for the greedy policy is given in [ren05], which is based on the M/M/1 model [klei75] in queuing theory.

According to Section 2.1.3.2, a Markov process can be expressed by a generator matrix. It can also be described by a **state-transition diagram**. Each state is denoted by a circle (or ellipse) in the diagram and transitions among these states are represented by arcs connecting the corresponding circles. The transition rate is marked as an expression on the arc. An M/M/1 model describes the behaviour of a system when there is only one server which provides service to incoming customers. Both the execution rate (μ) in the server and the arrival rate (λ) of the customers follow the exponential distribution.

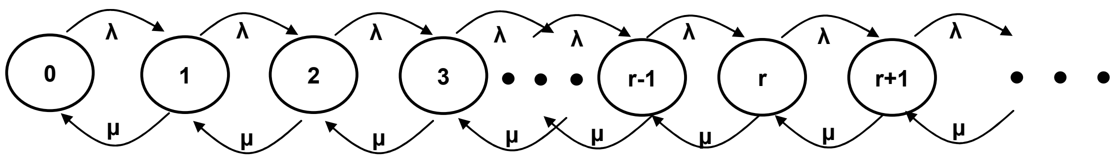


Figure 2-2: The State-transition-diagram of the M/M/1 Model

Figure 2-2 is about the corresponding state-transition diagram, and the generator matrix of the M/M/1 model is given below:

$$\begin{array}{c|ccccc|}
 & 0 & 1 & 2 & 3 & \cdots \\
\hline
0 & -\lambda & \lambda & 0 & 0 & \cdots \\
1 & \mu & -(\lambda+\mu) & \lambda & 0 & \cdots \\
2 & 0 & \mu & -(\lambda+\mu) & \lambda & \cdots \\
3 & 0 & 0 & \mu & -(\lambda+\mu) & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\hline
\end{array}$$

As an irreducible Markov process with infinite states, the numerical solution for the probability distribution can be achieved from the generator matrix when only n states are involved in the calculation and the probability of rest states are thought to be 0. On the other hand, an analytical solution can be found for a Markov process with infinite states if the process satisfies two pre-requests:

1) The process has at least one **delegate state**. A delegate state is a state whose probability can represent the probability of all others according to the equation $\boldsymbol{QG}$=0.

2) If Q_d is the probability of the delegate state, the probability distribution of all states can be expressed as a convergent serial of Q_d.

Take the M/M/1 model in Figure 2-2 for example, the equation $\boldsymbol{QG}$=0 is written as:

$$-\lambda Q_0 + \mu Q_1 = 0 \qquad \text{Equation 2-12}$$

$$\lambda Q_0 - (\lambda+\mu)Q_1 + \mu Q_2 = 0 \qquad \text{Equation 2-13}$$

$$\lambda Q_1 - (\lambda+\mu)Q_2 + \mu Q_3 = 0 \qquad \text{Equation 2-14}$$

$$\dots$$

$$\lambda Q_n - (\lambda+\mu)Q_{n+1} + \mu Q_{n+2} = 0 \qquad \text{Equation 2-15}$$

In Equation 2-12, Q_0 can be used to represent Q_1 as:

$$Q_1 = \frac{\lambda}{\mu} Q_0 \qquad \text{Equation 2-16}$$

When Equation 2-16 is integrated into Equation 2-13, Q_2 can also be represented by Q_0. The remaining probabilities can also be expressed through Q_0 in Equation 2-17.

$$Q_n = (\frac{\lambda}{\mu})^n Q_0 \qquad \text{Equation 2-17}$$

Therefore, the state 0 in the M/M/1 model is a delegate state. The probability distribution of $\{Q_0, (\frac{\lambda}{\mu})Q_0, (\frac{\lambda}{\mu})^2 Q_0, (\frac{\lambda}{\mu})^3 Q_0, \dots\}$ becomes a convergent series of Q_0 if and only if $\lambda<\mu$. The probability distribution can be solved by introducing the restriction equation $\sum_{i\in S} Q_i = 1$. For the M/M/1 model, we have

$$\sum_{i=0}^{\infty} Q_i = [1+\frac{\lambda}{\mu}+(\frac{\lambda}{\mu})^2+...]Q_0 = \frac{1}{1-\frac{\lambda}{\mu}} Q_0 = 1$$ Equation 2-18

Therefore, the analytical solution for the M/M/1 model is given in Equation 2-19.

$$Q_n = (\frac{\lambda}{\mu})^n (1-\frac{\lambda}{\mu})$$ Equation 2-19

Therefore, the probabilities of the 0 state and non-0 states in the M/M/1 model indicate the proportion of the idle period (T_{idle}) with the active period (T_{active}) of an SP.

2.1.4.2 Analytical Solutions for the Greedy Policy based on the M/M/1 Model

In [ren05], T_{wu} and T_{sd} represent the average latency for wakeup and shutdown transitions respectively ($T_{\text{sd}}=1/\chi_{\text{on,off}}$ and $T_{\text{wu}}=1/\chi_{\text{off,on}}$). An idle period $T_{\text{idle}}>0$ happens if and only if no more tasks are added to the TQ during the shutdown transition (otherwise, a wakeup transition follows the shutdown transition immediately). When the workload follows the Poisson distribution, the probability of non-zero idle period can be calculated as follows:

$Q(T_{\text{idle}}>0)=Q(\text{no tasks arrive during } T_{\text{sd}})=Q_{T_{sd}}(0)=e^{-\lambda T_{sd}}$ Equation 2-20

According to the greedy policy, the SP is woken up (and the idle period is terminated) when a new workload arrives. Following the Poisson distribution, the average interval between two continuous workloads is $1/\lambda$ and the average length of the idle period is calculated below:

$$\overline{T_{idle}} = \frac{1}{\lambda} e^{-\lambda T_{sd}}$$ Equation 2-21

With the assumption that the proportion between the average length of the non active period ($\overline{T_{idle}}+T_{\mathrm{sd}}+T_{\mathrm{wu}}$) and $\overline{T_{active}}$ is the same as that for $\overline{T_{idle}}$ with $\overline{T_{active}}$ in the M/M/1 model, the average length of the active period is calculated below:

$$\overline{T_{active}}=\frac{\frac{\lambda}{\mu}}{1-\frac{\lambda}{\mu}}(\overline{T_{idle}}+T_{sd}+T_{wu})=\frac{\lambda}{\mu-\lambda}(\frac{1}{\lambda}e^{-\lambda T_{sd}}+T_{sd}+T_{wu}) \qquad \text{Equation 2-22}$$

Finally the corresponding average power dissipation $\overline{P}$ is derived in Equation 2-23.

$$\begin{aligned}\overline{P}&=\frac{P_{off}\overline{T_{idle}}+P_{on}\overline{T_{active}}+P_{sd}T_{sd}+P_{wu}T_{wu}}{\overline{T_{idle}}+\overline{T_{active}}+T_{sd}+T_{wu}} \qquad \text{Equation 2-23}\\ &=P_{on}-\frac{(\mu-\lambda)[\lambda T_{TR}(P_{on}-P_{TR})+e^{-\lambda T_{sd}}(P_{on}-P_{off})]}{\lambda\mu T_{TR}+\mu e^{-\lambda T_{sd}}}\end{aligned}$$

In Equation 2-23, P_{TR} and T_{TR} are defined in Equation 2-2 and Equation 2-1 respectively. The $\overline{P}$ in Equation 2-23 is still not accurate enough since the assumption $\frac{\overline{T_{active}}}{\overline{T_{idle}}+T_{sd}+T_{wu}}=\frac{\lambda/\mu}{1-\lambda/\mu}$ can only be satisfied when both T_{sd} and T_{wu} are very small. The inaccuracy lies in the absence of the wakeup and shutdown transitions in the Markov model. Furthermore, this derivation of $\overline{P}$ can hardly be used for DPM systems whose SP has multiple operation modes.

2.1.4.3 Optimal *N*-policy Based on the M/E$_k$/1 Queuing Model

In [wang95], Wang and Huang used the M/E$_k$/1 model to find the probability distribution of *N*-policy, and optimized the parameter *N* accordingly. In their research, the execution in the server is modelled as an **Erlang** type *k* process which means the execution in the server can be divided into k independent stages and each of them follows the exponential distribution with mean $1/k\mu$. A customer goes into the first stage of the service (say stage *k*), then progresses through the remaining stages and must complete the last stage (say stage 1) before the next customer enters the first stage. The representations of state probabilities are defined as follows (In order to keep the same as [wang95], P stands for probability only in this section):

P_{00}^{0}	The probability that there are no customers in the system and zero stages of service when the server is turned off.
$P_{n,k}^{0}$	The probability that there are *n* customers in the system and the customer in service is in stage *k* when the server is turned off.
$P_{n,i}^{1}$	The probability that there are *n* customers in the system and the customer in service is in stage *i* when the server is turned on and in operation.

The probability distribution among all these states is given in the following equations:

$$\lambda P_{1k}^{0} = \lambda P_{00}^{0} \qquad \text{Equation 2-24}$$

$$\lambda P_{nk}^{0} = \lambda P_{n-1,k}^{0} \ (2 \le n \le N\text{-}1) \qquad \text{Equation 2-25}$$

$$\lambda P_{00}^{0} = k\mu P_{11}^{1} \qquad \text{Equation 2-26}$$

$$(\lambda + k\mu) P_{1i}^{1} = k\mu P_{1,i+1}^{1} \qquad (1 \le i \le k\text{-}1) \qquad \text{Equation 2-27}$$

…

The first and second equations describe the accumulation of *N* customers when the server is sleeping. The service of every customer at this time is pending in stage *k*. In the third equation, P_{11}^{1} indicates there is only one customer in its last service stage (stage 1) when the server is active. If the service is completed, there is no customer waiting for service and the server is turned off accordingly and the system moves to the state P_{00}^{0}. Therefore, this equation indicates the shutdown transition in the server. When no extra states are used to describe the behaviour when the server is under turn-off transition, this research considered the shutdown (as well as wakeup) transition as instantaneous. This assumption of instantaneous shutdown/wakeup transition derives on even probability distribution among all inactive states, i.e., $P_{00}^0 = P_{1k}^0 = P_{2k}^0 = ... = P_{n-1,k}^0$. With the probability distribution, the service in the system can be characterized by the following parameters:

L_N is the expected number of customers in the system.

I represents the **Idle Period**, which is the length of time when the server is turned off per cycle.

B represents the **Busy Period**, which is the length of time when the server is turned on and in operation and customers are being served per cycle.

C represents the **Busy Cycle**, which is the length of time from the beginning of the last idle period to the beginning of the next idle period.

The expected length of the idle period, the busy period and the busy cycle, are denoted by E[***I***], E[***B***] and E[***C***]. All these characteristics of the service can be derived from the probability distribution.

In order to optimize the parameter *N*, some cost variables are defined as follows:

C_h is the holding cost per unit time per customer present in the system.

C_o is the cost incurred per unit time for keeping the server on.

C_f is the cost incurred per unit time for keeping the server off.

C_s is the start-up cost for turning the server on.

C_d is the shut-down cost for turning the server off.

With these cost definitions, the total expected cost per unit time, *F*(*N*) is given by

$$F(N) = C_h L_N + C_o \frac{E[B]}{E[C]} + C_f \frac{E[I]}{E[C]} + (C_s + C_d)\frac{1}{E[C]} \qquad \text{Equation 2-28}$$

The optimized value of *N** can be derived by minimizing the corresponding cost function *F*(*N**), which satisfy:

$$F(N^*+1) \geq F(N^*) \leq F(N^*-1) \qquad \text{Equation 2-29}$$

Although C_o and C_f can be easily used to represent the power consumption for the server (in our case, the SP) when the latter is on and off respectively, the cost of the wakeup/shutdown transition cannot be simply represented by constants C_s and C_d because both transitions are not carried out instantaneously. When different number of tasks is added to the TQ during the shutdown or wakeup transitions, the power performance of an SP varies accordingly. This variation may have great influence about the choice of the optimized parameter *N*.

In conclusion, although many attempts have been made by electronics and mathematics studies, their results are still too sketchy for the description of the implementation of the A&F policy in a DPM system. Furthermore, these studies only considered a SP (or server) with only two operation modes. New models need to be built to describe the usage of the A&F policy in a DPM system whose SP has multiple operation modes.

2.2. Coloured Petri Nets

Coloured Petri Nets (CPN), as one kind of high-level net for system modelling, is now in widespread use for various practical purposes. This kind of high-level net model is developed from low-level Petri Nets (PN) [pete81] for representing complex information.

2.2.1 Petri Nets

Petri Nets, as a modelling tool for system behaviour, have been developed and implemented for real world practice for decades. Traditionally, a PN is defined as a tuple $\sum= (P, T, A, N, M_0)$ [alex98] where *P* is a finite set of **places**, which indicate states of the modelled system by means of ellipses (or circles). Each place may contain a dynamically varying number of small black dots, which are called **tokens**. M stands for a **marking**, which is an arbitrary distribution of tokens on places. M_0 represents the initial distribution of tokens on the places which is called the **initial marking**. T is a finite set of **transitions**, which indicate operations in the system by means of rectangles. The places and transitions of a PN are collectively referred to as **nodes (*N*)**. Nodes in a PN are connected by a set of directed arrows, which are called **arcs (*A*)**. Each arc connects a place with a transition or a transition with a place –but never two nodes of the same kind. Some positive integer is attached to each arc which is called the **arc expression**. If an arc points from node x to node y, node x is called an input node of y and y is the output node of x.

If and only if each input place of one transition contains at least the number of tokens prescribed by the expression of the corresponding input arc, the transition is **enabled**. Otherwise, the transition is **disabled** because some of its input places lack enough tokens. When a transition is enabled, the corresponding move may take place, which is called the **occurrence** of the transition. Tokens from the input places are removed from the input places and added to the output places after the execution of an occurrence. With the occurrence of different transitions, tokens are moved among different places and system processing can be modelled as a "token game". If every transition occurrence in the PN is called a **Step**, system marking is changed from M_0 to M_1 M_2, …M_i and so on, where i is the step number of the system. If one and only one transition is enabled in any step of the model, the corresponding system is a **sequential** system. When more than one transition is enabled in some step, the corresponding system is called a **concurrent** system. When several transitions are enabled by M_i, the occurrence sequence of these transitions is nondeterministic, and it may generate several different consequent markings M_{i+1}.

With all tokens having the same abstract meaning in Petri Nets, the system representation of PN is limited to integer meaning by the number of tokens contained in one place (0 means no token in the place). Therefore, two different nets have to be used by the PN to represent two systems with many similarities. This presents no problem in a small system, but it shows PN has limited power to describe a large real-world system which has many similar but not identical parts. Using Petri Nets, these parts must be represented by disjoint subnets. This not only means that the total Petri Nets model becomes very large, but also presents difficulties to show the similarities (and difference) between the individual subnets corresponding to similar parts.

2.2.2 Updating PN to CPN

In CPN, a more compact representation has been achieved by equipping each token with an attached data value — called the **token colour**. The data value may be of arbitrarily complex type. For a given place, all tokens must have token colours that belong to a specified type. This type is called the **colour set** of the place. Therefore, both the number and colour of a token are used to represent the marking in the CPN.

Attaching a colour to each token and a colour set to each place allows a CPN to use fewer places than would be needed in a PN. Tokens' movement in a CPN becomes more complex since enabling a transition depends not only upon token numbers from each input place, but also upon token colours. It also means that the colours of input tokens may determine the colours of the output tokens produced by ways of transition occurrence. Therefore, more elaborate arc expressions are used in CPN to specify a collection of tokens with a defined token colour.

With a colour set, a CPN place can represent a state of the modelled system which must be represented by several places in a traditional Petri Net. Similarly, a CPN transition can also represent a set of similar operations in the modelled system when **variables** are used in the arc expressions surrounding a given transition. These variables can be bound to different token colours (or values) so that arc expressions evaluate to different values. A transition in a CPN model is enabled if and only if each of its input places contains at least one set of tokens to which the corresponding arc expression evaluates.

Besides arc expressions, the CPN also uses the **guard** of a transition to evaluate input token values. The guard is a Boolean expression and may have variables in exactly the same way that arc expressions have. The guard defines an additional constraint which must be satisfied before the transition is enabled.

Similar to Petri Nets, CPN has its own mathematical definition of its syntax and semantics. A CPN net can be defined as a set of (∑, *P, T, A, N, C, G, E, I*) satisfying the requirement below [jens97]:

(i)	∑ is a finite set of non-empty types, called colour sets.
(ii)	*P* is a finite set of places.
(iii)	*T* is a finite set of transitions.
(iv)	*A* is a finite set of arcs such that : $P \cap T = P \cap A = T \cap A = \phi$
(v)	*N* is a node function. It is defined from *A* into $P \times T \cup T \times P$.
(vi)	*C* is a colour function. It is defined from *P* into ∑.
(vii)	*G* is a guard function. It is defined from *T* into expressions such that: $\forall t \in T : [Type(G(t)) = Boolean \wedge Type(Var(G(t))) \subseteq \Sigma]$
(viii)	*E* is an arc expression function.
(ix)	*I* is an initialization function.

With colours attached to tokens, and the extension in the expression of transitions and arc expressions, CPN can represent a system with a more delicate and compact model, or represent a system which cannot or hardly be represented by traditional PN. By simulation of the CPN model, it is possible to investigate different scenarios and explore the behaviours of a system.

2.2.3 CPN Tools and Example Implementation

CPN Tools [cpnt08] is the computer aid software for CPN modelling and analysis. Users of CPN Tools work directly on the graphical representation of CPN models. This software provides easy editing, simulation, state space analysis, and performance analysis of CPN models.

The interface of CPN Tools can be seen in Figure 2-3 where an example CPN model is given as well. This model is used to introduce the description and analysis functions provided by CPN Tools.

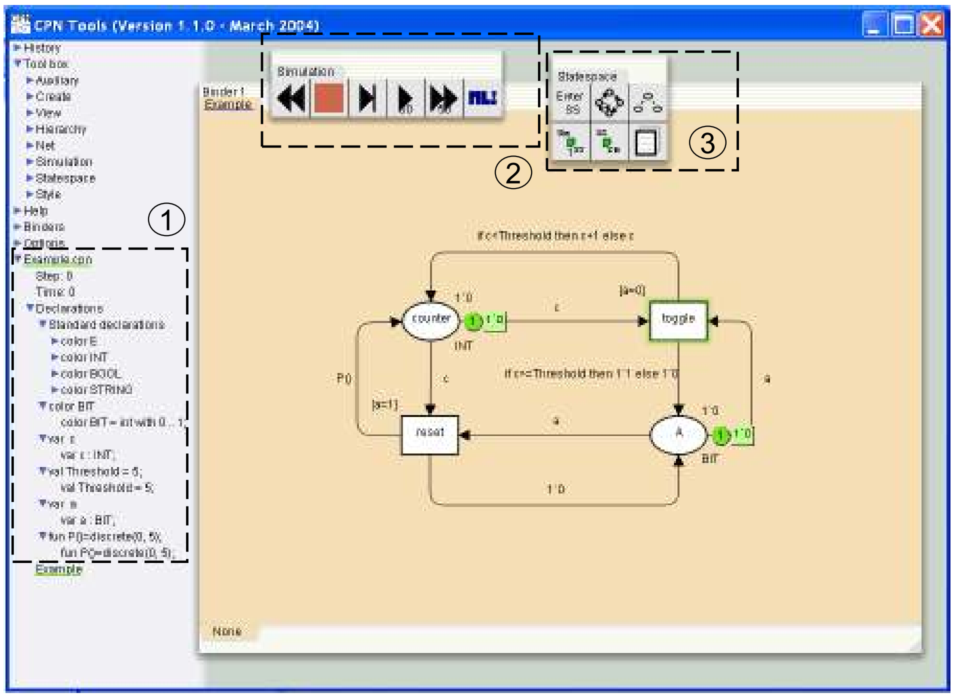


Figure 2-3: The User Interface of CPN Tools

2.2.3.1 Color Set Description

The declaration of the example model is given in the dotted rectangle (1) in the left side of the figure. The language used for declaration is called **CPN ML**. Colour sets are declared first. Four basic colour types in Standard ML (SML) [miln90] have been provided by the CPN Tools, which are declared in the "Standard declarations": colour **INT** (as the set of all integers), colour **STRING** (as the set of all text strings), colour **BOOL** (as the set of Boolean values, true or false), colour **E** (as the set of only one colour). Users can use any of these standard colours in their design, or declare their implementation oriented colours. Colours below the "Standard declarations" are colours declared by users themselves. A user defined colour can be a subset of the three standard colours provided by the CPN Tools (INT, STRING or BOOL). For example, the colour **BIT** is declared as an integer with only two values '0' or '1' (therefore a BIT token represents one bit of information). Users can also declare their colours from some already declared colour sets by means of a built-in colour set constructor 'product'.

Variables used in the system are declared with the key word '**var**'. Each variable declaration can introduce one or more variables with a type that has been declared before. In the example, variable *c* is declared as an INT variable and *a* is declared as a BIT variable.

Similar to high level languages in computer programming, a **constant** can also be used in CPN models if it has been defined before. Constant declaration is similar to variable declaration with the change of key word 'var' to '**val**'. For example, the constant Threshold is declared as 5 in Figure 2-3.

Functions can also be declared in CPN Tools. Each function declaration introduces a function. The function takes a number of arguments and returns a result. The arguments and the result have a type which is either a declared colour set, the set of all multi-sets over a declared colour set. In the example model, function **P()** uses the random number generator **discrete** provided by CPN Tools to generate a random number from 0 to 5.

2.2.3.2 Model Description

With the colour declaration, places can be put into the model. The name of a place is written inside the corresponding ellipse. The colour set of the place is given in the lower right side of the place and the initial tokens are given in the upper right side of the place. If the upper right side of a place is empty, no initial tokens are available in the place. 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, 1`1++1`0 describes two tokens with colours (values) of ‘1’ and ‘0’ respectively (In this thesis, a pair of quotations ‘’ is used to quote a colour value when it may be confused with the token number).

Similarly, the name of a transition is inside the rectangle and the Boolean expression of the transition guard is given in the upper left side of the transition (sometimes the guard is dragged to other side of the transition) within a pair of braces []. In the example model, the transition **toggle** has one guard [a=0] and the transition **reset** has one guard [a=1]. The expression of an arc can be found in the upper or lower side of the arc. A simple arc expression has the format *N*`*C* (where *N* is the token number and

C is the token colour or a variable). When the number of tokens used in the arc expression is 1, the 1` prefix can be omitted. Therefore, the expression for the arc from the place **A** to the transition reset is written as a, which has the same meaning as 1`a. More complex logic expressions can also be used in arc expressions, always in the form of "if … then… else…". For example, the expression for the arc from the toggle transition to the place A is written as "if c>Threshold then 1`1 else 1`0". It means that if the token value held in the place counter is bigger than 5 (Threshold), one '1' token is added to the A place, otherwise, one '0' is added instead.

With all declarations of colours, variables, constant(s) and function(s), the model given in the example describes a system where a counter (the place **counter**) is used to update the count number with a certain frequency (the transition toggle) until some threshold (the constant Threshold) is reached. A signal (the place A) is sent to reset (the transition reset) the number in the counter by a random number (the function P()) and the counter starts again.

2.2.3.3 Simulation and State Space

A CPN model is built for analysis and performance testing. The most straightforward kind of analysis is **simulation**, which in many respects is similar to the debugging and execution of a program. A simulation tool palette (the dotted rectangle (2)) is used in the CPN Tool to control the simulation. The user can choose to run the simulation by single step or automatic multiple steps.

Figure 2-4 gives the marking of the example model in several steps of the simulation (enabled transitions are marked in a dotted rectangle). The number of steps taken in the simulation so far is shown in the left side of the model, just under the model name.

Tokens in the model change their values and places according to the model construction until all transitions are disabled. In this way, the user can know whether the occurrence of transitions in the model can correctly reflect the processing in the modelled system, or whether the processing of the modelled system reflected by the occurrence of transitions is correct or not. Therefore simulation result can help users to update or correct their models.

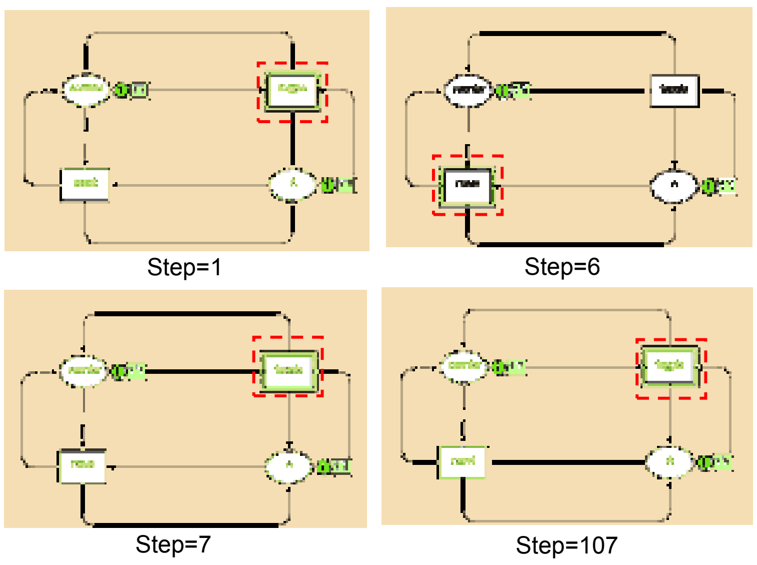


Figure 2-4: Several Steps in the Example Model's Simulation

However, simulation result cannot obtain a complete proof of the properties of CPN (Unless the nets or the properties are trivial) since the result achieved from simulation cannot be guaranteed to cover all possible executions. The property verification is given when full **state spaces** representing all possible executions of the model is analysed. A state space tool palette (the dotted rectangle (3) of Figure 2-3) is also provided by CPN Tools which does the state calculation, and the result is given in a standard report as below:

```
Statistics
----------------------------------------------------------------------
 Occurrence Graph                     Scc Graph
     Nodes:  7                          Nodes:  1
     Arcs:   7                          Arcs:   0
     Secs:   0                          Secs:   0
     Status: Full
 Boundedness Properties
---------------------------------------------------------------------
  Best Integers Bounds    Upper      Lower
  Example'A 1             1          1
  Example'counter 1       1          1

  Best Upper Multi-set Bounds
  Example'A 1         1`0++1`1
  Example'counter 1   1`0++1`1++1`2++1`3++1`4++1`5

  Best Lower Multi-set Bounds
 Example'A 1         empty         Example'counter 1   empty

 Home Properties
---------------------------------------------------------------------
  Home Markings:  All

 Liveness Properties
---------------------------------------------------------------------
  Dead Markings:  None
  Dead Transitions Instances: None
  Live Transitions Instances: All
```

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 some **state space statistics** telling how large the state space is. For example, the report indicates the directed **Occurrence Graph** uses 7 nodes and 7 arcs to show its full status. The generation of the full state spaces is in most cases followed by the generation of the **Strongly Connected Component Graph** (SCC-graph) which is derived from the graph structure of the state space.

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 report clearly shows one and only one token resides in the A and counter place of the example model respectively. 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 A, only one token can reside in the place (as also shown in the Best Integers Bounds) which has the value either ‘0’ or

'1'. For the token in the counter place, it may have five possible values from '0' to '5'. 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 is the **home properties**, which is about the reachable property of markings and transitions in the model. A **home marking** M_{home} is a marking which can be reached from any reachable marking. The report of the example model shows all markings in the model are home markings although a random function P() is used. A **dead marking** M_{dead} is marking which no binding elements are enabled. For the example model, no marking is dead because all markings can be repeated given enough simulation steps. 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 proves both transitions in the example model are live.

All this 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.

2.3. MATLAB Introduction

MATLAB is a highly versatile language for technical computing. The name stands for Matrix Laboratory. It integrates computation, visualisation, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. It is an interactive system whose basic data element is an array that does not require dimensioning. This allows the user to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or FORTRAN.

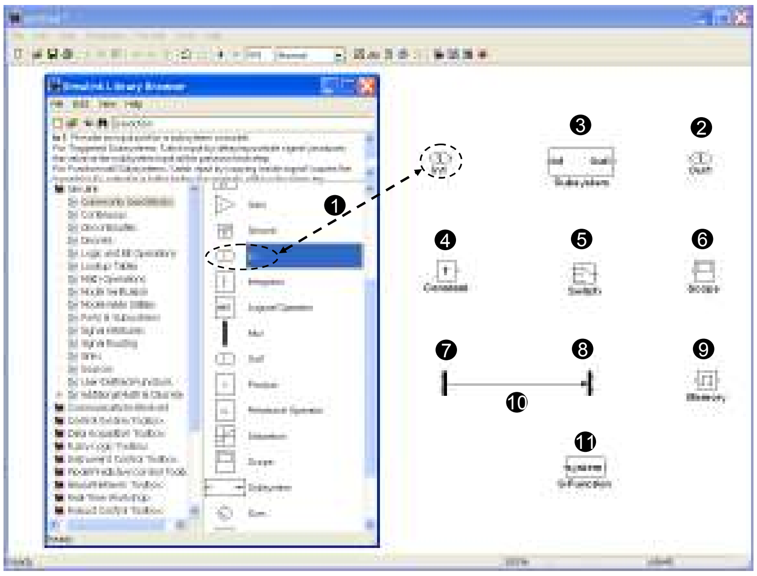


Figure 2-5: The GUI of MATLAB Simulink

2.3.1 Simulink

Simulink, as one important package of MATLAB, is used for modelling, simulating, and analysis of dynamic systems. It supports linear and nonlinear systems, modelled in continuous time, sampled time, or a hybrid of the two. Simulink provides a graphical user interface (GUI) for building models as block diagrams (Figure 2-5). Simulink includes a comprehensive library of sinks, sources, linear and nonlinear components, and connectors. Such components can be easily added to the Simulink model by click-and-drag mouse operations ((1) in Figure 2-5). Several components which are used to build our model in Chapter 5 are listed in the right side of Figure 2-5:

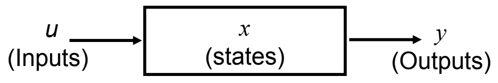
Inport components ((1) in Figure 2-5) and **Outport** components ((2) in Figure 2-5), stand for the input/output port for all kinds of model systems and subsystems built by MATLAB Simulink. When hierarchical design is implemented to build a complicated Simulink model, modules in different level can be represented by a group of nesting used **subsystem** components (3). When double clicked a subsystem component, Simulink opens another window to give the detail of the subsystem.

A **constant** component (4) represents a constant signal or value that may be used in the model design. A **switch** component (5) passes through the first (top) input or the third (bottom) input based on the value of the second (middle) input. Some adjustable threshold is set for each switch component. For example, when the threshold is 0, the switch component passes on the first input if the second input is higher than 0 and passes on the third input otherwise. A **scope** component (6) is used to observe the variation of the input signal (the number of inputs is adjustable) in time (sample) sequence.

A **mux** component (7) is used to merge all its inputs (the number of inputs is adjustable) into one integrated output if the designer want to make the model more concise. A **demux** component (8) is used to decompose an input to several outputs.

A **memory** component (9) outputs its input from the previous time step, applying a one integration step sample-and-hold to its input signal. The memory component is indispensible in the representation of feedback signals of a Simulink model. It is because Simulink takes a signal propagating in any connection line (like (10) in Figure 2-5) as instantaneous. The one sample step delay brought by every memory component can prevent ambiguous execution order in a loop. Therefore, in our design in Chapter 5, all memory components used for this aim are named as **delay** to differentiate other subsystem components that are used for storing signals or data.

2.3.2 Mathematical Expression of Simulink Execution



$$x = x_0 \quad \text{(Initialization)}$$

$$y = f_0(t, x, u) \quad \text{(Outputs)}$$

$$\dot{x}_c = f_d(t, x, u) \quad \text{(Derivation)}$$

$$x_{d_{k+1}} = f_u(t, x_c, x_{d_k}, u) \quad \text{(Update)}$$

where x=[x_c; x_d]

Figure 2-6: Mathematical Expression of Simulink Execution

Figure 2-6 presents the mathematical expression of a Simulink component. Vectors *u* and *y* represent the inputs to and outputs from a Simulink component respectively. Vector *x* represents the states of the component. x_c and x_d are used to represent the continuous and discrete states in *x* respectively. If x_0 represents the initial status of the component, it is initialized to *x* during the initialization phase of Simulation model execution. When the initialization completes, Simulation executes all components in sequence according to the model's structure/connection. For the execution of one particular component, Simulink calculates the new output of the component based on the current input *u* and state *x*. It calculates the component's new state by derivation (for a continuous system) and/or update (for a discrete system). This continues until the simulation is complete.

Therefore, although the model built in Simulink cannot represent and simulate truly asynchronous behaviours, because MATLAB is a synchronous platform, its simulation result can be very close to that generated in a real asynchronous system when the components used in the model are atomic and the sample intervals are short enough.

2.3.3 Simulink S-function

When some design can not be represented by components provided by the Simulink Library, the **S-function** component ((11) in Figure 2-5) can be used to describe their design in program codes and integrate the codes with the other components in Simulink. An S-function (system-function) is a computer language description of a Simulink block. S-function can be written in MATLAB, C, C++, Ada, or Fortran. S-function uses a special calling syntax that enables one to interact with Simulink equation solvers. The form of an S-function is very general and can accommodate continuous, discrete, and hybrid systems.

An S-function template in MATLAB language is given below (In MATLAB programming, codes after % mark are comments). It is composed of three main functions. Function **mdlInitializeSizes** is used to specify how many inputs, outputs as well as discrete and continuous states are used in this S-function, and it also gives the initial value of all states. The function **mdlUpdate** is used to realize the $y=f_o(t,x,u)$ function introduced in Figure 2-6 to calculate the output value of y. It also specifies the state derivation and update functions introduced in Figure 2-6. The function **mdlOutputs** is used to specify which variable is used for output.

```
function [sys,x0,str,ts] = sfundsc1(t,x,u,flag)

switch flag,

 %%%%%%%%%%
 % Initialization %
 %%%%%%%%%%
 case 0,
  [sys,x0,str,ts]=mdlInitializeSizes;

 %%%%%%%
 % Update %
 %%%%%%%
 case 2,
  sys = mdlUpdate(t,x,u);

 %%%%%%%
 % Output %
 %%%%%%%
 case 3,
  sys = mdlOutputs(t,x,u);

 %%%%%%%%
 % Terminate%
 %%%%%%%%
 case 9,
  sys = [];

 otherwise
  error(['unhandled flag = ',num2str(flag)]);
end

%end sfundsc1

%==============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%==============================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes

sizes = simsizes;

sizes.NumContStates    = 0;
sizes.NumDiscStates    = 1;
sizes.NumOutputs       = 1;
sizes.NumInputs        = 1;
sizes.DirFeedthrough   = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

x0  = 0;

str = [];
ts  = [0, 0];
% end mdlInitializeSizes

%
%==============================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step requirements.
```

```
%==========================================================
%
%system status

function sys = mdlUpdate(t,x,u)

sys = [x(1)];

%end mdlUpdate

%
%=========================================================
% mdlOutputs
% Return the output vector for the S-function
%=========================================================
%
function sys = mdlOutputs(t,x,u)

sys = [x(1)];

%end mdlOutputs
```

Chapter 3

Markov Models for Different DPM Systems

3.1. Power and Latency Analysis

3.1.1. Power Analysis

According to the introduction in Section 2.1.3, a stochastic model of a DPM system is composed of three components: A service requestor (SR), a service provider (SP) and a power manager (PM). An SP in a DPM system responds to the incoming events by processing their corresponding tasks with one of its *r* operation modes in the discipline of First-Come-First-Serve (FCFS). Some of the modes in the SP focus on low power dissipation and do not provide a task service. These modes, like sleep, idle or standby modes provided by many HDDs (Hard Disk Drives) and processors, are generally called **inactive** modes. Other modes in the SP, which provide task execution (but with different rates), are called **active** modes. If *I* and *A* are used to represent the sets of inactive and active modes respectively, we have $M = I \cup A$ where $M=\{M_i \mid i=0,1,2,\ldots,r-1\}$ is defined as the set of all operation modes in Section 2.1.3. Tasks for their corresponding incoming events can only be executed in the SP when the latter is in one of its active modes. Previous models in [beni99, qiu99, ren05] use the length of the task queue (TQ) to reflect the status of a DPM system. One and only one state 0 is provided in these models to present the status of a DPM system if the SP is inactive. This representation may neither reflect the different inactive modes that the SP is in, nor reflect transitions among these modes. When they are used for power and/or latency estimation, they may cause great inaccuracy. For more accurate description, the mode of an SP serves as the auxiliary index of the state in our Markov models.

In this chapter, dual indexing (n, M_i) is used to represent the states in the Markov models. The first index n is the length of TQ and the second index is the current operation mode chosen by the SP from its mode set M. We use $M_{i,j}$ to represent the mode switch transition from M_i to M_j, and the corresponding state index becomes (n, $M_{i,j}$).

With dual indexes, we can describe the behaviour of a DPM system with more accuracy. For example, when M_0 and M_1 are the two operation modes in the SP, state (n, M_0), (n, M_1) and (n, $M_{0,1}$) can represent different status of the DPM system although the length of the TQ is the same for all these cases.

When the SP is switching among inactive modes or from an inactive mode to an active mode or vice versa, no service is provided in the SP and the length of the TQ increases monotonically if new events come during these transitions (suppose the corresponding memory is unlimited). Additional states must be provided to the Markov models to represent the change in the TQ in these cases. Otherwise the execution of tasks which correspond to the incoming events during these transitions can not be reflected in the model. It makes the power estimation far lower than the real case.

Things are different for switching among active modes. DVS/DFS policies can be regarded as implementations of active mode switching management systems within a general DPM framework. According to [yuan05], DFS/DVS systems can be divided into **optimistic** or **pessimistic feasible** systems. The former allows continuous task execution during mode switching while the latter does not. If the DVS is pessimistic feasible, new events incoming during ongoing DVS operations only cause an increasing in the TQ, because the execution in the SP ceases before the complete of the DFS/DVS transition. However, in a system with optimistic feasible DVS, all tasks, whether they are enabled before or during the transition, are executed seamlessly during the mode switching transitions. Therefore, these transitions can be regarded as costing no extra energy and time [yuan05, pill01, beig08] (i.e. $Energy_{i,i}$=0 for $i, j \in A$) and do not need to be modelled as explicit states. Optimistic feasible DVS is supported in many advanced SOCs such as PowerPC 405LP [brok03] and is currently seeing an increased representation in new research [beig08, mats08]. In this thesis we concentrate on optimistic DVS/DFS. Not representing active to active mode switching as explicit states makes it easy to construct models with closed-form analytical solutions. For models of pessimistic DVS systems, additional states are needed to be integrated in the stochastic models so as to reflect the DVS/DFS transitions and one can seek [karg05] for detail.

In Section 2.1.3, an SP is described by a set group <χ, μ, *Pow, Energy*>, and two assumptions should be emphasized here. One is that all mode switching transitions in this chapter are assumed to be atomic, which means a mode switching transition $M_{i,j}$ cannot be replaced by two continuous mode transition like $M_{i,k}$ and $M_{k,j}$. The other is one and only one mode switching transition exists from one mode to the other. These assumptions are satisfied by most implementations.

In this case, low power design at the gate-level or Register Transfer Level (RTL) tries to lower the power consumption in the SP when it is processing and carrying out mode transitions (i.e. reduce Pow_i and $Energy_{i,j}$ for mode M_i). High level DPM policies, on the other hand, try to reduce the *average* power consumption ($\overline{P}$) in the SP by optimizing the latter's distribution among all possible operation modes. The unified expression for $\overline{P}$ is given in Equation 3-1.

$$\overline{P} = \sum_{i \in M} Pow_i \sum_{n=0}^{L_T} Q_{n,M_i} + \sum_{\substack{i,j \in I, or \\ i \in I, j \in A, or \\ i \in A, j \in I}} Energy_{i,j} \chi_{i,j} \sum_{n=0}^{L_T} Q_{n,M_{i,j}} \qquad \text{Equation 3-1}$$

In the equation above, L_T is the maximum length of TQ and $\sum_{n=0}^{L_T} Q_{n,M_i}$ represents the sum of probabilities of all states when the SP is in state M_i. When multiplied with Pow_i, the unit component $Pow_i \sum_{n=0}^{L_T} Q_{n,M_i}$ represents the contribution to $\overline{P}$ when the SP is in mode M_i. Therefore the first component of Equation 3-1 reflects the average power consumed in the SP when it stays in any of its active or inactive modes. Similarly, $\sum_{n=0}^{L_T} Q_{n,M_{i,j}}$ is the sum of probabilities of all states for mode transition $M_{i,j}$. The product of $Energy_{i,j}$ and $\chi_{i,j}$ is the average power consumption of this mode transition. Therefore the second component of Equation 3-1 represents the contribution of none 'active to active' mode transitions to $\overline{P}$. This unified equation is used in the following sections to show the different power performance brought by different policies.

3.1.2. Latency Analysis

Whatever policies may be implemented, the reduction in power of a DPM system may always be at the expense of longer latency. System engineers need to balance both the gain in power and the cost in latency before applying the proper DPM policy to the

PM in their systems. Previous studies [beni99, ren05, qiu99] always use the average length of TQ as the latency measurement since it is directly reflected by the stochastic models. However, the average length of TQ, which was used by all these models as the measure of the system latency, is of very limited practical use. This measure has no direct relationship to the Quality of Service (QoS) of an SP. For example for the same SP, an increase in task queue may cause faster drop of QoS for multimedia tasks than pure text tasks because the former tasks tend to take longer time to process. It is much more important to know how long a task is likely to wait, than how many other tasks are likely to sit before it in the TQ.

In system design, **soft deadlines** for task execution have been a popular measure for real-world latency performance, and are widely used in many implementations like DVS analysis [karg05]. Different from hard real-time deadlines, soft deadlines are not compulsory. They serve more as guidelines for execution scheduling to optimize system performance. Violation of such a deadline is not considered a catastrophe. The quality of service in a processor can be measured by the probability of task deadline violation. A high probability of deadline violation indicates bad QoS. The **Average Probability of Deadline Violation** (*APDV*) for all tasks embedded in an SP is used here as a much better measurement for the latency of different policies. This measurement is more practically expressive to directly reflect the system latency performance than the average length of TQ. If the *APDV* value for every state (n, M_i) or (n, $M_{i,j}$) in a DPM system is known, the *APDV* value for the system is just the sum of these *APDV* values weighted by their corresponding state's probability.

For a clear representation in the *APDV* derivation, we can generally divide all states in a DPM system into several groups (Table 3-1) according to the similarity in their latency performance.

Table 3-1: The Representation of Different State Groups

Index	Representation
(n, M_i)	The state when the length of TQ is n and the SP is in active mode M_i.
(n^*, M_i)	The state when the length of TQ is n and the SP is in inactive mode M_i.
$(n^*, M_{i,j})$	The state when the length of TQ is n and the SP is in transition from inactive mode M_i to active mode M_j.
$(n', M_{i,j})$	The state when the length of TQ is n and the SP is in transition from active mode M_i to inactive mode M_j.
$(n^{**}, M_{i,j})$	The state when the length of TQ is n and the SP is in transition from inactive mode M_i to inactive mode M_j.

3.1.2.1 *APDV* for state (n, M_i)

Suppose an SP is in the state (n, M_i) when a new event arrives, the corresponding task is added to TQ (the corresponding event handling is taken as instantaneously) and there are n+1 tasks so far waiting to be executed. The task corresponding to the new event is executed only after the completion of the previous n tasks, assuming FCFS without losing generality (Non-FCFS execution sequences have no influence on the *average* probability of deadline violation). The latency in this case is the execution time of n+1 tasks in the SP. Suppose the n+1 tasks are executed in the same mode M_i in the SP and the execution of each task following Poisson distribution, the execution of n+1 tasks follows the **Erlang** distribution [klei75] with parameters n+1 and μ_i (μ_i is the execution rate of mode M_i). A deadline violation happens when no more than n tasks can be completed during *DL*, and the *APDV(DL, n, M_i)* in this case can be expressed as Equation 3-2:

$$APDV(DL,n,M_i)=\sum_{k=0}^{n}E_r(DL;k,\mu_i)$$ Equation 3-2

where *DL* is the mean deadline requirement of task execution.

When the SP use a serial of active modes M_c, M_{c+1},…, M_i ($\chi_c \leq \chi_{c+1} \leq \ldots \leq \chi_i$) to execute the *n*+1 tasks according to the length of the queue, we can first divide the *n*+1 tasks to (*i*-*c*+1) groups G_c, G_{c+1}, …, G_i ($G_c+G_{c+1}+\ldots+G_i=n+1$) according to the operation modes that are used in their execution. Suppose m_j ($c \leq j \leq i$) is the time slice given by the system to complete the (*j*-*c*+1)th groups of tasks (G_c *to* G_j), a deadline violation happens when any task group cannot be completed within its given m_j time slot. Therefore we have:

$$APDV(DL,n,M_i)=(1-\sum_{n_i=G_i}^{\infty}E_r(m_i;n_i,\mu_i)\sum_{n_{i-1}=G_{i-1}}^{\infty}E_r(m_{i-1};n_{i-1},\mu_{i-1})\ldots\sum_{n_1=Gc}^{\infty}E_r(m_c;n_1,\mu_c))$$

Equation 3-3

When only optimistic featured DVS/DFS technologies are considered, the SP can carry out task executions during the ‘active to active’ transitions. Therefore, no extra latency is caused by the ‘active to active’ transitions.

3.1.2.2 *APDV* for state (*n**, $M_{i,j}$)

Similarly, if the SP is within the duration of one ‘inactive to active’ transition $M_{i,j}$ when a new event arrives, the execution of the corresponding task must first wait for the completion of the transition $M_{i,j}$, and then the execution of the *n* tasks accumulated previously. Suppose *m* is the time slice given to complete the $M_{i,j}$ transition, the corresponding *APDV* calculation is shown in Equation 3-4.

$$APDV(m, M_{i,j}) = e^{-\chi_{i,j} m} \qquad \text{Equation 3-4}$$

The rest *DL-m* time slice is given by the SP to execute the *n*+1 tasks using mode M_j (The new incoming events in the transition $M_{i,j}$ or the execution in M_j does no contribution to *APDV* value because FCFS). A deadline violation happens when the transition $M_{i,j}$ or the M_j execution cannot be completed in the given time. Therefore, we have the corresponding *APDV* for state (n^*, $M_{i,j}$) when *m* is integrated from 0 to *DL* (Equation 3-5).

$$APDV(DL, n^*, M_{i,j}) = 1 - \int_0^{DL} (1 - APDV(DL - m, n, M_j))(1 - APDV(m, M_{i,j}))dm$$

Equation 3-5

The *APDV*(*DL-m*, *n*, M_j) and *APDV*(*m*, $M_{i,j}$) components in Equation 3-5 come from Equation 3-3 and Equation 3-4 respectively.

3.1.2.3 *APDV* for state (n^*, M_i)

If an SP is in one inactive state (n^*, M_i), the time spent by a task before its execution completion can be divided into three parts.

First of all, it is the time spent before the PM issues a command to activate the SP. For example, DPM systems with the A&F policy only activate the SP when *N* tasks are accumulated in the TQ. If $n<N$-1, the PM only activates the SP after the other (*N*-*n*-1) events have come. If *m* is the time spent in waiting for the other (*N*-*n*-1) events coming, the corresponding *APDV* calculation is shown in Equation 3-6.

$$APDV(m, M_i) = \int_m^{\infty} E_r(t; N - n - 1, \lambda)dt \qquad \text{Equation 3-6}$$

After an activation decision is made, the SP starts one ‘inactive to active’ mode transition. Finally, the execution of the example task is carried out in the SP after the first *n* tasks have been executed. The corresponding *APDV* for the ‘inactive to active’ transition and the execution is given in Equation 3-5. Therefore, the *APDV* for the new event cost in the three parts can be derived by the following equation.

$$APDV(DL,n^*,M_i) = (1-\int_0^{DL}(1-APDV(DL-m,n^*,M_{i,j}))(1-APDV(m,M_i))dm$$

Equation 3-7

The $APDV(DL\text{-}m, n^*, M_{i,j})$ and $APDV(m, M_i)$ components in Equation 3-7 come from Equation 3-5 and Equation 3-6 respectively.

3.1.2.4 *APDV* for state (*n'*, $M_{i,j}$)

If the SP is in an 'active to inactive' transition when the new event arrives, the first part of its latency comes from the completion of the $M_{i,j}$ transition. Given that *m* is the time slot set for the completion of the $M_{i,j}$ transition, the corresponding *APDV* is given in Equation 3-4 where $\chi_{i,j}$ is the transition rate of $M_{i,j}$. When the transition is complete, the next action of the DPM system varies according to different state (n, $M_{i,j}$). If the PM decides to switch the SP to active mode M_k as soon as the $M_{i,j}$ transition is complete (for example, the current TQ length is longer than *N* in the A&F policy), the corresponding *APDV* is given in Equation 3-8.

$$APDV(DL,n',M_{i,j}) = (1-\int_0^{DL}(1-APDV(DL-m,n^*,M_{j,k}))(1-APDV(m,M_{i,j}))dm$$

($n \geq N$-1) Equation 3-8

The two *APDV* expressions in Equation 3-8 come from Equation 3-5 and Equation 3-4 respectively.

On the other hand, if the PM prefers to keep the SP in the inactive mode M_j after the $M_{i,j}$ transition (for example, the current TQ length is less than *N* in the A&F policy), the corresponding *APDV* becomes:

$$APDV(DL,n',M_{i,j}) = (1-\int_0^{DL}(1-APDV(DL-m,n^*,M_j))(1-APDV(m,M_{i,j}))dm$$

(n<N-1) Equation 3-9

The two *APDV* expressions in Equation 3-9 come from Equation 3-5 and Equation 3-7 respectively.

3.1.2.5 *APDV* for state (n**, $M_{i,j}$)

The actions in an SP when an 'inactive to inactive' transition takes place are similar to that in the SP when an 'active to inactive' transition is carried out. Therefore, we can use Equation 3-8 or Equation 3-9 to calculate the $APDV(DL, n^{**}, M_{i,j})$ in this case.

3.1.2.6 *APDV* for an entire DPM system

With the *APDV* values for all state groups available, the average deadline violation for a DPM system is the sum of these *APDV* values weighted by their corresponding states' probabilities.

$$\begin{aligned}APDV(DL) = &\sum_{i\in A}\sum_{n=0}^{L_T} Q_{n,M_i} APDV(DL,n,M_i) + \sum_{\substack{i\in I,\ n=0\\ j\in A}}\sum^{L_T} Q_{n^*,M_{i,j}} APDV(DL,n^*,M_{i,j}) \\ &+\sum_{i\in I}\sum_{n=0}^{L_T} Q_{n,M_i} APDV(DL,n^*,M_i) + \sum_{\substack{i\in A,\ n=0\\ j\in I}}\sum^{L_T} Q_{n',M_{i,j}} APDV(DL,n',M_{i,j}) \\ &+\sum_{\substack{i\in I,\ n=0\\ j\in I}}\sum^{L_T} Q_{n,M_{i,j}} APDV(DL,n^{**},M_{i,j})\end{aligned}$$

Equation 3-10

3.1.3 Balance of both power and latency

In previous sections, we introduced the equations for $\overline{P}$ (the average power consumption) and *APDV* (the Average Percentage of Deadline Violation). The two variables are used to represent the power and latency features of a DPM system respectively. System engineers may have different emphasis on power/latency of one DPM system when it is used in different implementations. Therefore, we provide one unified equation to evaluate the system performance in both power and latency so as to help system engineers to choose correct DPM policy in their implementations.

$$Balance(\lambda, DL) = (1 - TOL) \times \frac{\overline{P}}{P_{Max}} + TOL \times APDV(DL) \qquad \text{Equation 3-11}$$

Tolerance of Latency (*TOL*, 0< *TOL*<1) in Equation 3-11 reflects how much relative weight has been given to latency and this parameter can be adjusted by system engineers according to their requirements. Therefore, (1-*TOL*) is the concern given to the power dissipation. P_{Max} in the equation above, as the maximum *Pow* value for all SP modes, is used to normalize $\overline{P}$ and unify the dimension of the equation. Given the value of *TOL*, a DPM policy which can minimize Equation 3-11 is the optimal policy for both latency and power performance.

Furthermore, because all DPM policies can be regarded as better than others under certain circumstances, recent research work focuses on a hierarchical architecture of DPM design which can dynamically adapt different DPM policies to portable systems [ren05]. A unified cost function like Equation 3-11 can serve as a standard assessment framework of policies to allow hierarchical DPM to adjust different policies according to the variation of environment parameters such as λ and *DL*.

With the measure in both power and latency, we try to model and analyze the behaviour of different DPM systems in the following sections when different stochastic policies have been implemented.

3.2. On-off DPM Systems

If only two modes are used in the SP ($M=\{M_i \mid i=0,1\}$, $I=\{M_0\}$, $A=\{M_1\}$), we have an on-off DPM system. Normally M_0, M_1, $M_{0,1}$, $M_{1,0}$ are called the *sleep* mode, *work* mode, *wakeup* transition and *shutdown* transition respectively. The dual indexes introduced in section 3.1 can be specified and simplified according to Table 3-2.

Table 3-2: Alias Index Used in On-off DPM Systems

Standard	(n^*, M_0)	(n, M_1)	$(n^*, M_{0,1})$	$(n', M_{1,0})$
Alias	n^*	N	wu(n)	sd(n)

In the following sections, two models are given about an on-off DPM System when the greedy and the A&F policies are implemented respectively.

3.2.1. The Greedy Policy

3.2.1.1 The Description of the Markov Model

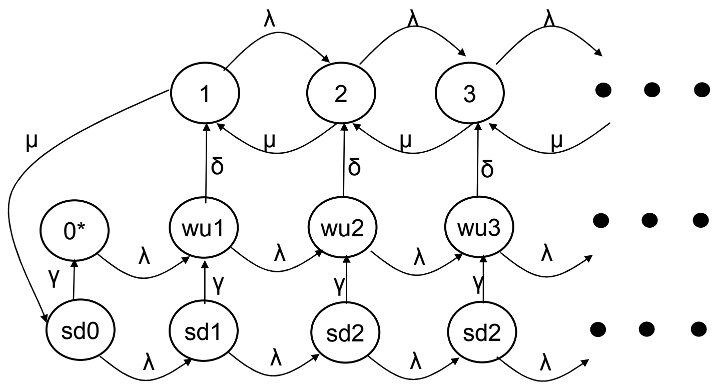


Figure 3-1: The Markov Model for the Greedy Policy

The transition-state-flow diagram of the Markov model for the greedy policy can be seen in Figure 3-1. Similar to previous models [qiu99, ren05], λ and μ are used as the arrival rate of external events from the SR and execution rate of tasks in the SP respectively. State 0* indicates the SP is in its sleep mode (M_0). When an event arrives, the SP starts the wakeup transition ($M_{0,1}$) which is represented by wakeup states wu1, wu2 and so on. As explained in Section 2.1.3, both wakeup and shutdown transitions are taken as Markov processes whose rates are represented by δ and γ respectively ($\chi_{0,1}=\delta$, $\chi_{1,0}=\gamma$) in Figure 3-1. During the wakeup transition, the system starts in state wu1, and may then move to state 1 if the transition is completed without any other events coming. Otherwise, the system may move from wu1 to wu2 and even further if one or more event comes during the transition. In order to make our model more general, we set L_T=∞ and use an infinite number of wakeup states in Figure 3-1. After the wakeup transition is completed (e.g. at the completion of state wu(*n*)), the SP is activated and starts its execution of all *n* tasks in the TQ. The system is now in (active) state *n* and the completion of one task enables the system to move one state to the left side.

When the execution of the last task in the TQ is completed (system leaving state 1), the SP starts a shutdown transition, described by shutdown states sd0, sd1 etc. If one or more event arrives during the shutdown transition, the SP is activated immediately on completion of the shutdown transition. Otherwise, the SP starts sleeping and moves back to state 0*.

3.2.1.2 The Derivation of the Analytical Solution

According to the method introduced in section 2.1.3, the probability distribution in Figure 3-1 can be solved analytically. In the derivation, (Active) state 1 is chosen as the delegate state, and its probability (Q_1) is used to represent the probabilities of the rest states. The derivation detail is given in Appendix I, and Table 3-3 lists the group of analytical solutions.

Table 3-3: The Analytical Solutions for the Greedy Policy

$Q_{sd(n)} = \frac{\mu}{\lambda+\gamma}(\frac{\lambda}{\lambda+\gamma})^n Q_1$	Equation 3-12
$Q_{0*} = \frac{\gamma}{\lambda}\frac{\mu}{\lambda+\gamma} Q_1$	Equation 3-13
$Q_{wu(n)} = \frac{\gamma Q_{sd1}}{\lambda+\delta}\sum_{k=0}^{n-1}[(\frac{\lambda}{\lambda+\gamma})^k(\frac{\lambda}{\lambda+\delta})^{n-1-k}]+(\frac{\lambda}{\lambda+\delta})^n Q_{0*}$	Equation 3-14

$Q_n = \sum_{k=1}^{n}(\frac{\lambda}{\mu})^{k-1}Q_1 - \frac{\delta}{\mu}[\sum_{k=1}^{n-1}\sum_{s=k}^{n-1}(\frac{\lambda}{\mu})^{n-s}Q_{wu(k)}]$	Equation 3-15
$S_{sd} = \frac{\mu}{\gamma}Q_1$	Equation 3-16
$S_i = Q_{0*}$	Equation 3-17
$S_{wu} = \frac{\mu}{\delta}Q_1$	Equation 3-18
$S_a = \frac{\mu}{\mu-\lambda}[1+\frac{\lambda}{\delta}+\frac{\lambda^2}{(\lambda+\gamma)\gamma}]Q_1$	Equation 3-19
$Q_1 = \frac{1}{\frac{\mu}{\gamma}+\frac{\mu\gamma}{\lambda(\lambda+\gamma)}+\frac{\mu}{\delta}+\frac{\mu}{\mu-\lambda}[1+\frac{\lambda}{\delta}+\frac{\lambda^2}{(\lambda+\gamma)\gamma}]}$	Equation 3-20

3.2.1.3 The Performance Analysis

In this chapter, we apply the greedy model in Figure 3-1 on some real example systems. A FUJI MHF 2043AT HDD which was used in both [beni00] and [lu00] serves as our first example whose parameters are given in Table 3-4.

Table 3-4: Parameters for a FUJI MHF 2043AT

P_s(W)	P_w(W)	T_{wu}(s)	T_{sd}(s)	P_{wu}(W)	P_{sd}(W)
0.13	0.95	1.61	0.67	2.85	0.54

In Table 3-4, P_s, P_w, P_{wu} and P_{sd} are the power consumption of the SP in its sleep mode, work mode, wakeup transition and shutdown transition respectively. For better presentation and comparison of the performance of different DPM systems, the execution speed μ in the SP is normalized to 1 (for an SP with multi active modes in the coming sections, μ_{max}, as the fastest execution rate provided by the SP, is normalized to 1), and the arrival rate λ and the transition rates in the matrix of χ are normalized accordingly. Therefore, the reciprocal values of T_{wu} and T_{sd} in Table 3-4 are used as δ and γ respectively. Because the overhead caused in mode switching transitions is our main concern, we first examine the variation of mode switching frequency.

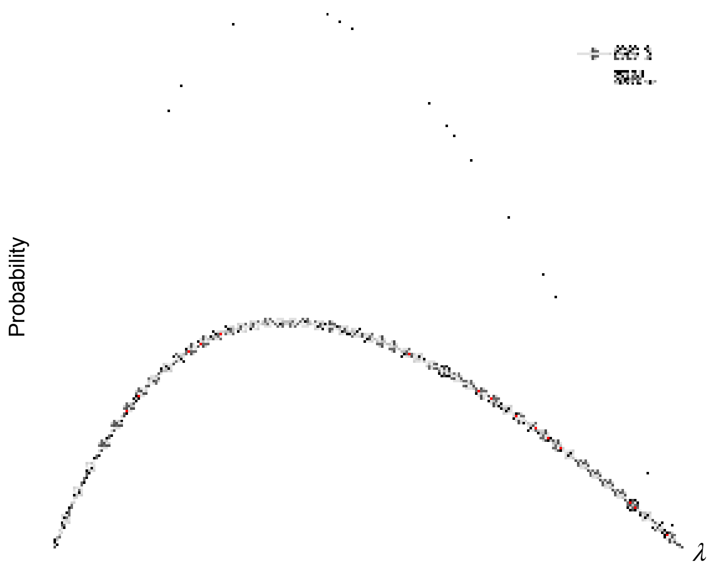


Figure 3-2: The Frequency of Mode Switching Transitions

In Figure 3-2, we choose S_{sd} and S_{wu}, which are the sum of probabilities of shutdown and wakeup states respectively, as the measure of the mode switching frequency (The calculation of S_{sd} and S_{wu} are given in Equation 3-16 and Equation 3-18 respectively). When the arrival of external events is sparse, the SP has a great chance to finish the execution of all tasks in the TQ and be turned to sleep before one new event arrives. Therefore, the mode switching frequency rises with λ increasing. However, when the arrival of external events is dense, new task is added to the TQ with much faster speed and the TQ is seldom to be empty. In this case, the mode switching frequency drops when λ increases. Therefore, both curves in Figure 3-2 are convex with λ and the variation in the mode switching frequency influences the power performance in the DPM system (Figure 3-3).

When external events come sparsely ($\lambda \to 0$), the SP is seldom woken up and spends most of its time in sleep. Therefore, the power curve above starts from P_s when $\lambda \to 0$. With the increase of λ from the start point, $\overline{P}$ increases quickly because more and more executions are needed in the SP.

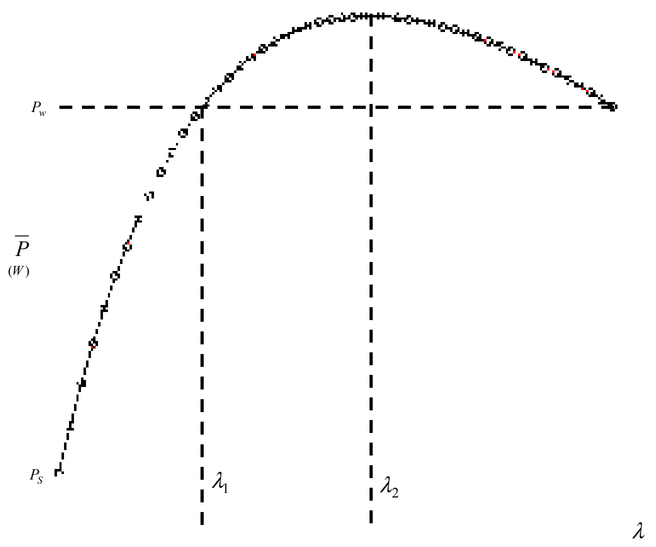


Figure 3-3: $\overline{P}$ in the FUJI HDD for the Greedy Policy

According to Table 3-4, the average power consumption cost in mode switching is even higher than that consumed by the SP when it is in its active mode ($\frac{P_{wu}+P_{sd}}{2} > P_w$), which means too frequent on-off mode switching transitions may cause more power consumption in the system and an SP like this HDD is called a **high transition cost SP** accordingly. In the figure, when the arrival rate is denser than λ_1, $\overline{P}$ becomes even higher than P_w. From then on, the power overhead brought by the power control becomes higher than the achieved power saving in the SP (λ_1 is called the **effective boundary** later). This situation reaches its **worst case** when $\lambda=\lambda_2$ and decreases after that because the mode switching frequency drops. When $\lambda \rightarrow \mu$ (μ has been normalized to 1), the SP is kept busy doing executions and hardly shutdown. Therefore, the power curve above ends in P_w.

In Section 2.1.3.2, we mentioned that the optimized policy given by previous study only focuses on the energy efficiency, while it ignores the energy safety. Parameters effective boundary and worst case are used to describe the energy safety of the greedy policy (as well as the A&F policy in Section 3.2.2 later). These parameters indicate the users how safe the corresponding DPM policy for their particular implementation can be, and what is the worst situation caused by the policy.

In Section 2.1.4, we have introduced the equation for the average power consumption of the greedy policy given by [ren05]. The power curves calculated from our equation (Equation 3-1) and Ren's equation (Equation 2-23) are compared in Figure 3-4 ('Old Pave' in the legend stands for the power estimation made by Ren's model and 'New Pave' is the power estimation made by our model). Because Ren's calculation uses the proportion between the working period and the sleeping period (as well as the wakeup and shutdown transitions) achieved from the M/M/1 model, its estimation of power is higher than our estimation value and the greatest difference happens in the middle where mode switching transitions happen frequently.

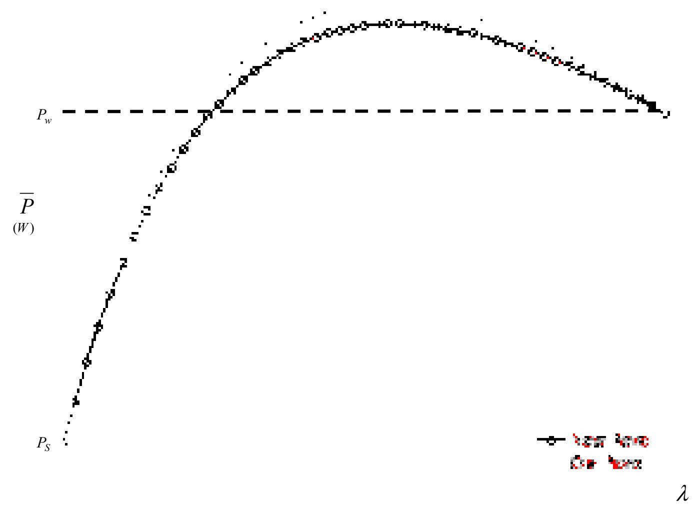


Figure 3-4: Power Curves in Different Models

Another study is carried out with one IBM HDD [iran03] whose parameters are given in Table 3-5. The IBM HDD can provide three inactive modes and one active mode in its operation. Here, only its sleep and work modes are studied and the other modes will be added to the investigation in the following sections. For easy calculation, we assume both the χ and Energy matrices are symmetric for this SP, i.e. $\chi_{i,j} = \chi_{j,i}$ and

$Energy_{i,j}$= $Energy_{j,i}$, which means the energy and latency cost in shutdown transition are 4.75J and 5s respectively. We can calculate P_{wu}=P_{sd}=0.95W.

Table 3-5: Parameters for IBM HDD

Mode	Power (W)	Start-up Energy(J)	Transition Time to Active
Sleep	0	4.75	5s
Standby	0.2	1.575	1.5s
Idle	0.9	0.56	40ms
Work	1.9	0	0

Different from the parameters in Table 3-4, the average power consumption in the mode switching transitions of IBM HDD is less than P_w (1.9W) ($\frac{P_{wu}+P_{sd}}{2} < P_w$) (an SP like this HDD is called a **low transition cost SP** accordingly).

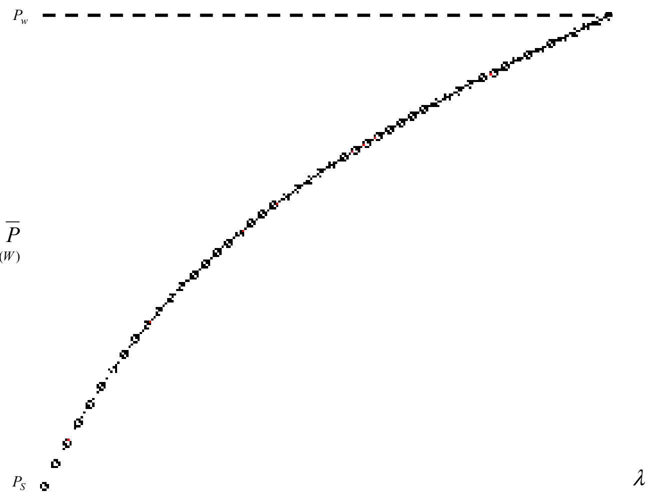


Figure 3-5: $\overline{P}$ for the IBM HDD

After we normalize the wakeup and shutdown transition rates (δ=3/T_{wu} γ=4/T_{sd}), Figure 3-5 indicates the power curve does not surpass P_w in the entire variation of λ and the greedy policy can help the SP to save power no matter how dense the external events arrival rate may be (0<λ<μ).

In the next figure, we give the *APDV* value of the greedy policy in the IBM HDD when we set the average deadline request to ten times of the average execution period (*DL*=10/μ)(Figure 3-6). The lower curve shows the *APDV* value when there is no power control in the system (the SP is always on). It can be seen that the *APDV* value is not always 0 and may become very high when *λ*→*μ*. It proves that deadline violation cannot be totally avoided even when an SP is never shut down.

The above curve in Figure 3-6 is the *APDV* curve for the greedy policy. The distance between the two curves in Figure 3-6 suggests the additional latency brought by the greedy DPM control. Figure 3-5 and Figure 3-6 clearly shows how the greedy policy trades extension in latency for reduction in power.

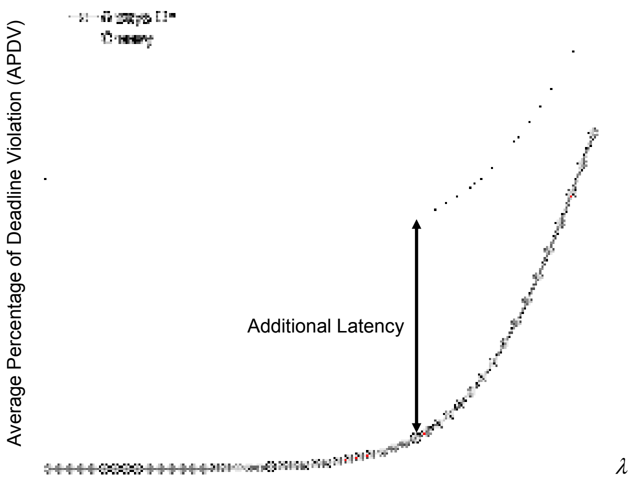


Figure 3-6: The *APDV* Value for the Greedy Policy (*DL*=10/*μ*)

Although the greedy policy is ‘to activate the SP as soon as a new event arrives’, the *APDV* value for the greedy policy is not 0 when *λ*→0 and the SP is in the sleep mode with almost probability 1. It is because the SP must carry on the wake up transition before providing execution service. With the increase of *λ*, the SP spends more time in the active mode, and incoming events when the SP is active do not have any wakeup time cost in their latency. This explains the slight drop in the middle of the greedy *APDV* curve. When events incoming becomes even denser, the time spent waiting for the other tasks serve as the main reason for the extension of deadline violation and it indicates the sharp increase in both *APDV* curves when *λ*→*μ*.

The *APDV* value depends not only on the policy used in the DPM system, but also on the capacity of the SP. Given the same incoming events, their corresponding tasks can be executed much faster in a SP with high capability than in a SP with low capability. In this case, the deadline request of incoming events is relatively looser in a DPM system with high capability SP than that in a DPM system with low capability SP. This trend is clearly shown in Figure 3-7.

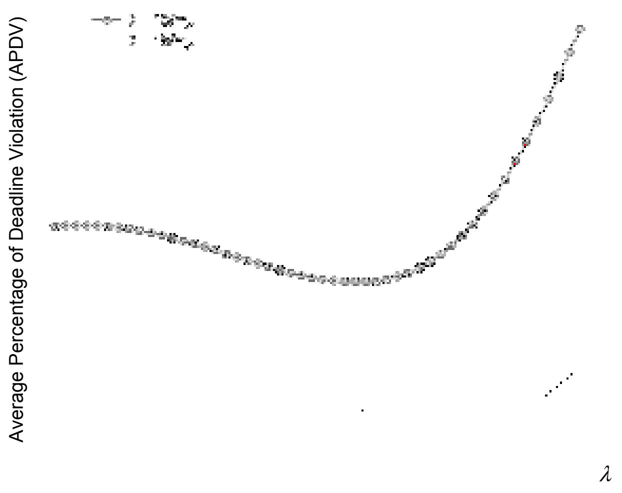


Figure 3-7: Different *APDV* Values in Different Deadline Requirements

3.2.1.4 Conclusions

The greedy policy, as the classic protocol of other advanced DPM policies, was thoroughly studied in this section. An infinite number of states is used in the Markov model in Figure 3-1 to represent the status of a DPM system when mode switching transitions are carried out. This helps to make our model avoid inconsistent representations in other models used in previous researches. Analytical solutions of the probability distribution in the Markov model are derived, which makes the power and latency estimation more accurate than before.

With two example SPs, the performance of the greedy policy is studied in the above section. The property of the greedy policy to trade latency extension for power reduction is clearly shown by both $\overline{P}$ and *APDV* curves. However, the reduction in power consumption cannot be achieved in the full range of λ if a high transition cost SP is used in the DPM system. The power reduced by the greedy policy is limited especially when the deadline request is loose. All these shortcomings serve as motivation to analyze more delicate policies which can better balance system performance in power and latency, or trade more latency for power when needed.

3.2.2. The A&F policy

Considering the energy overhead of mode switching in an SP, a natural improvement is to reduce the mode switching frequency. Therefore, we investigate the practice of accumulating tasks before activating an SP for batch processing, which is the Accumulation & Fire (A&F) policy introduced in Section 2.1.4.

In the A&F policy, an SP is not activated immediately when a new event arrives (and its corresponding task becomes ready for processing). Instead, the SP remains inactive while tasks accumulate in the TQ. This task accumulation continues until a certain limit N is reached. The SP is then activated to batch process all accumulated tasks. The moment of activation is called the **fire** moment in the A&F policy. The greedy policy can be regarded as a basic A&F policy with N=1.

3.2.2.1 Markov model description

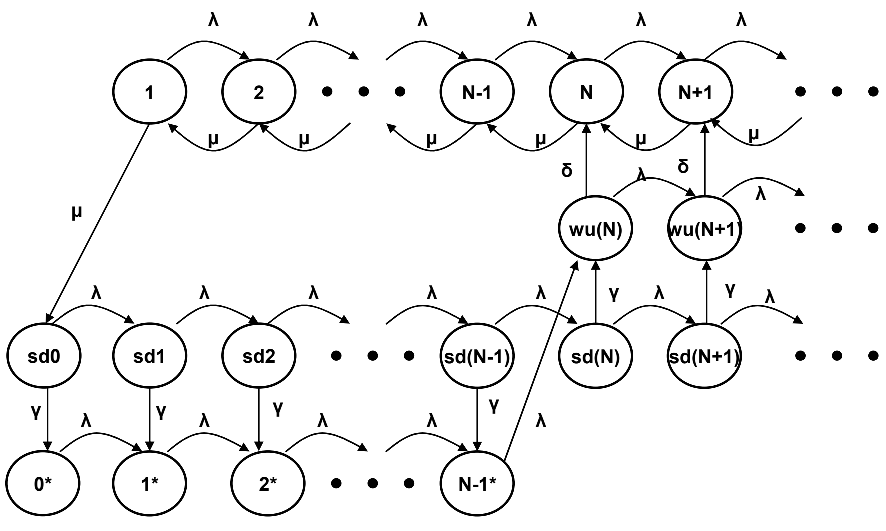


Figure 3-8: The Markov Model for the A&F policy

Figure 3-8 gives the transition-state-flow diagram of the Markov model for the A&F policy. According to the introduction before, a sleeping SP is activated by a new incoming event only when *N*-1 tasks are available in the TQ. Therefore, the TQ may contain 0 to *N*-1 tasks before the SP is activated and the inactive state(s) extend from only one state 0* in Figure 3-1 to *N* states (0* to (*N*-1)*) in Figure 3-8. The wakeup transition only happens when there are enough tasks accumulated in the TQ. Therefore the index of wakeup states starts from wu(*N*) instead of wu1 in the greedy policy. If *N*=1, the A&F policy model in Figure 3-8 becomes the greedy policy model in Figure 3-1.

Table 3-6: Analytical Solutions for the A&F Policy

$Q_{sd(n)} = \frac{\mu}{\lambda+\gamma}(\frac{\lambda}{\lambda+\gamma})^n Q_1$	Equation 3-12
$Q_{n*} = \frac{\mu}{\lambda}[1-(\frac{\lambda}{\lambda+\gamma})^{n+1}]Q_1$	Equation 3-21
$Q_{wu(n)} = \frac{\gamma Q_{sd(N)}}{\lambda+\delta}\sum_{k=0}^{n-N}[(\frac{\lambda}{\lambda+\gamma})^k(\frac{\lambda}{\lambda+\delta})^{n-N-k}]+(\frac{\lambda}{\lambda+\delta})^{n-N+1}Q_{(N-1)*}$	Equation 3-22
$Q_n = \sum_{k=1}^{n}(\frac{\lambda}{\mu})^{(k-1)}Q_1$ ($n \le N$)	Equation 3-23
$Q_n = \sum_{k=1}^{n}(\frac{\lambda}{\mu})^{k-1}Q_1 - \frac{\delta}{\mu}[\sum_{k=N}^{n-1}\sum_{s=k}^{n-1}(\frac{\lambda}{\mu})^{n-s}Q_{wu(k)}]$ ($n>N$)	Equation 3-24

$S_{sd} = \frac{\mu}{\gamma} Q_1$	Equation 3-16
$S_i = \frac{\mu}{\lambda}[N - \frac{\lambda}{\gamma}(1 - (\frac{\lambda}{\lambda + \gamma})^N)]Q_1$	Equation 3-25
$S_{wu} = \frac{\mu}{\delta} Q_1$	Equation 3-18
$S_a = \frac{\mu}{\mu - \lambda}[N + \frac{\lambda}{\delta} + \frac{\lambda^{N+1}}{(\lambda + \gamma)^N \gamma}]Q_1$	Equation 3-26

3.2.2.2 The Derivation of Analytical Solutions

Similarly as the derivation of the analytical solution of the greedy policy, (Active) state 1 is chosen as the delegate state and its probability (Q_1) is used to derive the probability distribution of the entire model.

The derivation details are given in Appendix II, and Table 3-6 lists the analytical solutions. It can be seen that these solutions become their counterparts in the greedy policy in Table 3-3 when N=1.

3.2.2.3 The Performance Analysis

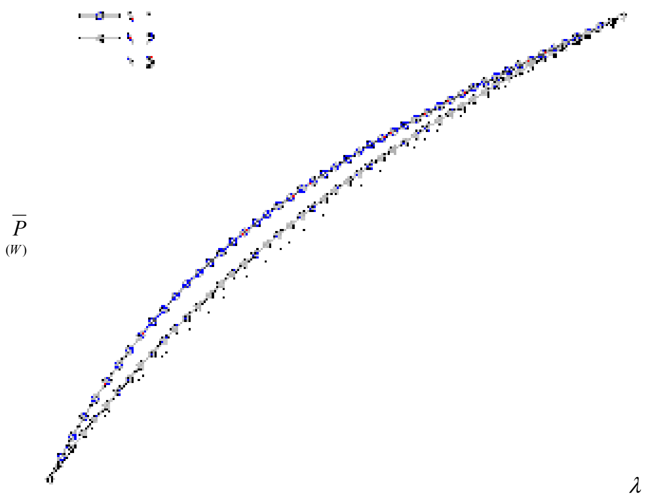


Figure 3-9: $\overline{P}$ with Different Accumulation Limit Ns

In this section, we use the two example SPs whose parameters given in Table 3-4 and Table 3-5 respectively to analyze the performance of the A&F policy. For the low transition cost SP like the IBM HDD, the average power curves for different accumulation limit *N*s are given in Figure 3-9. It can be clearly seen that $\overline{P}$ decreases continuously with the rise of *N* for all events arrival rate except the boundary values ($\lambda \to 0$ and $\lambda \to \mu$) while the improvement extent reduces at the same time.

The A&F policy can play more important role if it is implemented in DPM systems with high transition cost SP like the FUJI HDD. The power curves for different *N*s of the FUJI HDD are displayed in Figure 3-10.

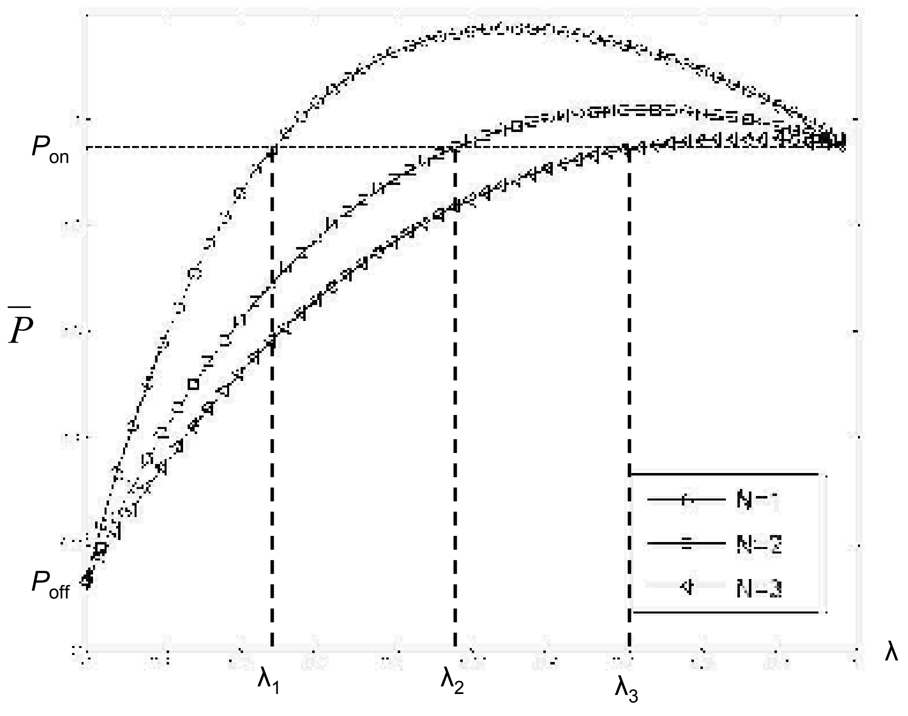


Figure 3-10: $\overline{P}$ for Different *N*s (continue)

First of all, the increase of accumulation in this case can also reduce the average power consumption in the SP continuously. The (*N*=1) curve describes the power performance when the SP is controlled by the greedy policy. With high transition cost, the greedy policy can only help the SP to reduce its power consumption when $\lambda<\lambda_1$. With the implementation of the A&F policy, the effective boundary (λ_2 for *N*=2 and λ_3 for *N*=3) extends greatly with the rise of *N* and the worse case power consumption reduces as well. All these properties show that the A&F policy has great advantage in power saving especially when implemented in the high transition cost SPs.

Figure 3-11 displays the latency performance for the IBM HDD when the A&F policy with different *N*s is implemented. The increase of *N* causes more deadline violations in the system, which is just the trade-off for the corresponding reduction in power consumption. According to the A&F policy, a new incoming event adding the n^{th} task to the TQ of a sleeping SP cannot activate the latter if $n<N$. The SP must wait for the availability of another $N-n$ tasks and the corresponding waiting time highly depends on the event arrival rate λ. That explains why the *APDV* curves for the A&F policy ($N>1$) drop sharply when $\lambda<0.5$. When $\lambda>0.5$, there are always many tasks accumulated in the TQ, and the rise in *APDV* curves mainly comes from waiting for other tasks' execution.

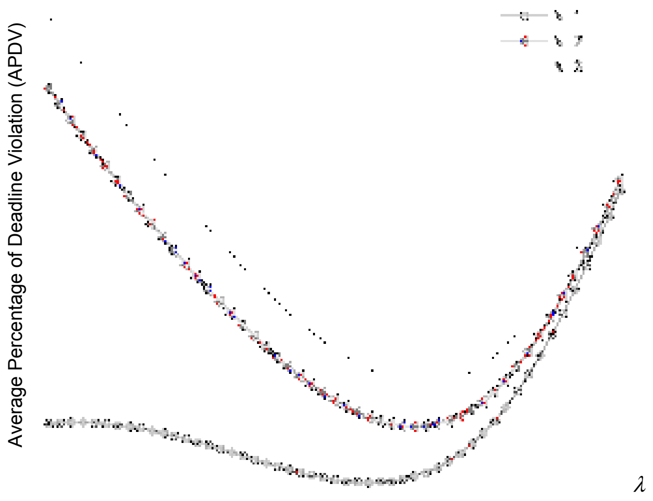


Figure 3-11: *APDV* Values for Different *N*s (*DL*=10/*μ*)

The latency performance for low transition cost SPs like the FUJI HDD can be analyzed similarly, and the result shows a similar feature as Figure 3-11.

In section 3.1, we introduced the Balance variable (Equation 3-11) which estimates both power and latency performance as a whole. When implemented in the same DPM system, a DPM policy which can achieve the minimum Balance value is the optimized policy for the implementation. For the A&F policy, the balance value is also important in determining a proper value for the accumulation limit *N* in the implementation.

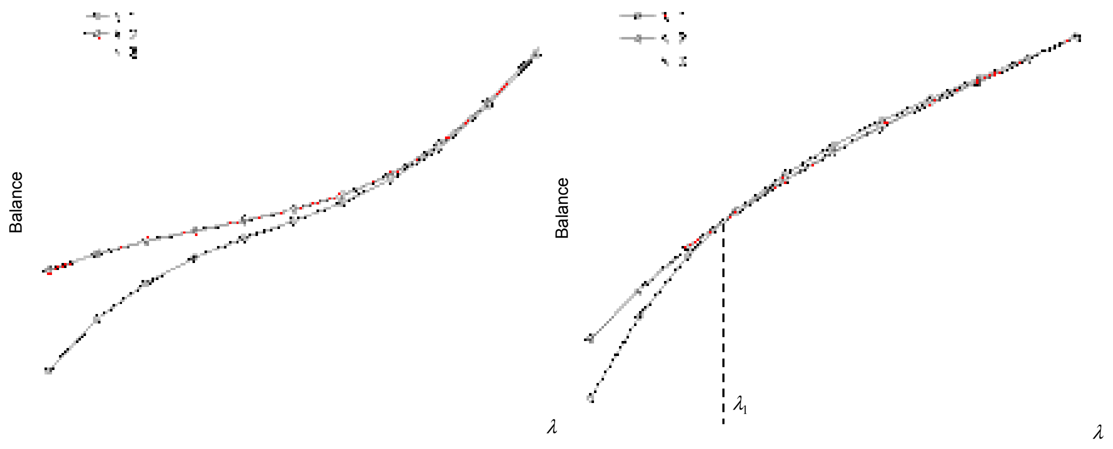


Figure 3-12: The Balance Value for Different *N*s (a) *TOL*=0.6 (b) *TOL*=0.4

Based on the power and latency analysis carried out in Figure 3-9 and Figure 3-11, we give the balance values for the same DPM systems with different A&F limits. In Figure 3-12, we first set *TOL* (Tolerance of Latency) to 0.6, which means the system engineers care more about latency than power in the system performance. The figure discloses that the greedy policy should be chosen as the optimized policy because it has the shortest latency extension. Next we reduce *TOL* to 0.4, the result shows that the A&F policy with large *N* serves as the optimized policy when $\lambda > \lambda_1$ because it can effectively reduce the power dissipation in the system.

As noted in Section 2.1.4, previous studies of the *N*-policy ignored events arrived during mode switching transitions. Therefore, all shutdown and wakeup states are omitted from their models. This omission not only causes inaccuracies in cost estimation of mode switching transitions, but also changes the entire power estimation. It is because the probability distribution of the entire system changes accordingly. According to the M/E$_k$/1 model of *N*-policy in Section 2.1.4.3, the probabilities of state 0* to (*N*-1)* are the same across the entire λ range ($Q_{i*}=Q_{j*}$ for $0<i,j<N$ and $P_{ik}^{0} = P_{jk}^{0}$ in Equation 2-25). In Figure 3-13, we compare the probabilities of states 0*, 1* and 2* (Q0*, Q1* and Q2* in the legend of Figure 3-13) for the A&F policy (*N*=3). It is clear that these probabilities are different across the entire λ range, which makes previous *N*-policy models inaccurate here.

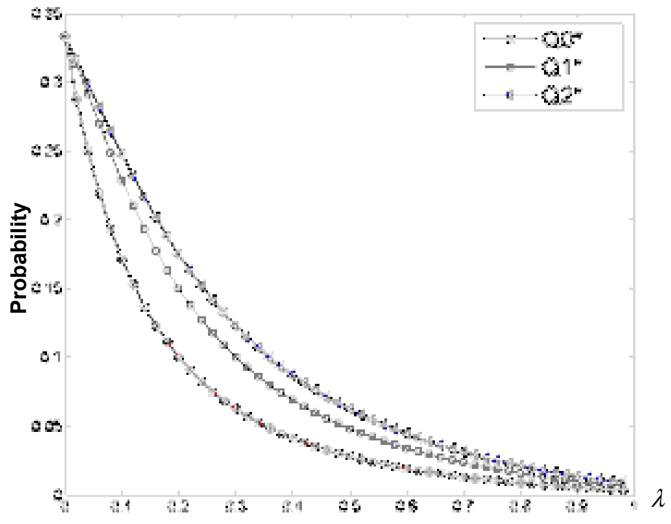


Figure 3-13: The Probability of Inactive States

3.3. DPM Systems with Multi Inactive Modes

Many processors/micro controllers have some additional inactive mode(s) ($I=\{M_0,\ldots,M_i \mid i>0\}$) other than the sleep mode. These additional modes try to give an SP a quicker response time to switch back to one of its active mode(s). The additional inactive mode(s) is often called idle/standby mode. Compared with the sleep mode, the SP in the idle mode has lower cost to switch from/to active mode(s), but consumes more power when it is in the mode. This mode is mostly used when the SP is under suspension and waiting to resume [beni99].

In the previous section, the greedy or the A&F policy is chosen to manage the wakeup transition. With multiple inactive modes in this section, we can use different policies or the same policy with different parameters (for example, the A&F policy with different *N*s) to switch among different modes. In this example, three modes, sleep, idle and active, are used in an SP (M={0,1,2}, I={0,1}, A={2}), and their mode switching transitions are shown in Figure 3-14.

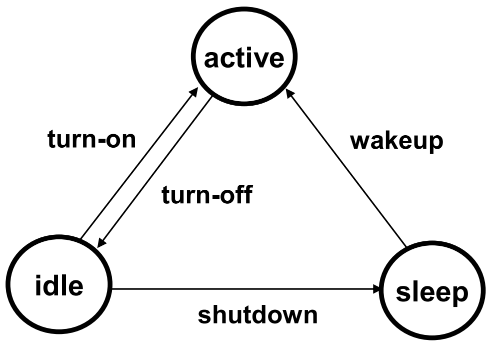


Figure 3-14: Mode Switching Transitions in DPM Systems with Multiple Inactive Modes

Once the wakeup transition ($M_{0,2}$) is completed, the SP starts a task execution. When the last task in the TQ is completed, the SP first switches to the idle mode. This transition ($M_{2,1}$) is called **turn-off** in this section. If new events arrive when the SP is idle, the SP switches back to the work mode for execution (the transition $M_{1,2}$ is called **turn-on** accordingly). Otherwise, after a Poisson time interval, the SP is shut down (transition $M_{1,0}$) and switched to the sleep mode to save power. In the example given in Figure 3-15, the A&F policy and the greedy policy are implemented to manage the wakeup transition (transition $M_{0,2}$) and the turn-on transition (transition $M_{1,2}$) respectively.

3.3.1 Markov model description

In Figure 3-15, the turn-on (ton1, ton2 and so on) and turn-off (toff0, toff1 and so on) states represent the corresponding transitions. Parameters α, β are used to represent the normalized transition rates of turn-on ($\chi_{1,2}$) and turn-off ($\chi_{2,1}$) respectively. When the greedy policy is implemented to control the turn on transition, only one idle state (0*, M_1) is needed to represent the case when the SP is in the idle mode.

When the TQ is empty, the SP may have low latency if it spends more time in the idle mode than the sleeping mode. Otherwise the SP can have lower power consumption. Therefore, different probability distributions among inactive modes can be used to adjust the power-latency trade-off. In Figure 3-15, ε is the rate of leaving idle mode for the sleeping mode. Large ε means short stay in idle mode and small ε means long stay in idle mode.

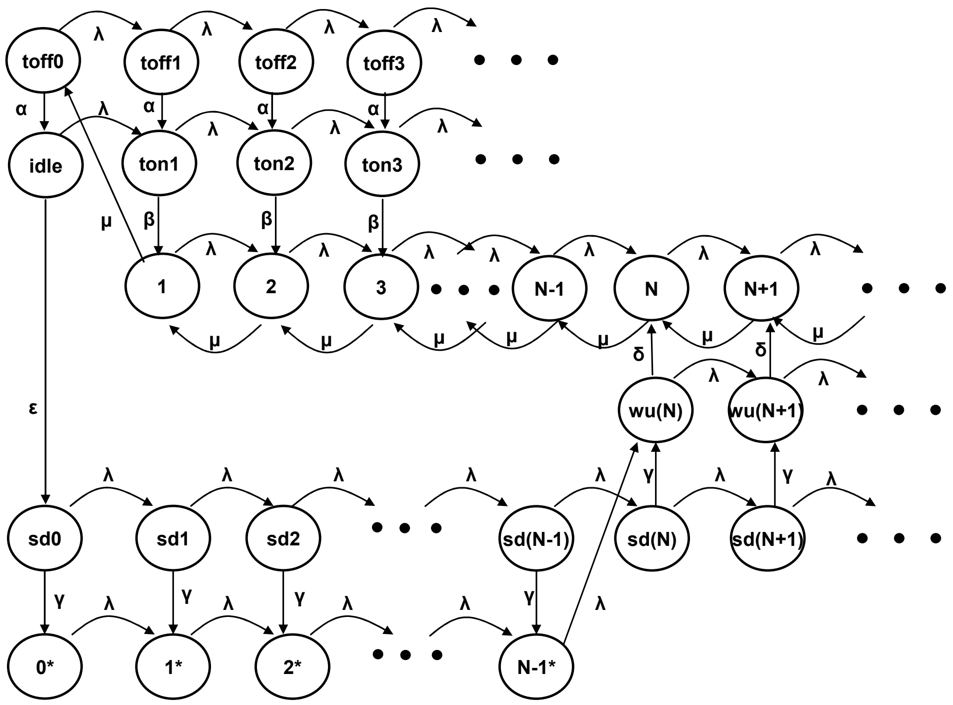


Figure 3-15: Markov Model for DPM System with Idle Mode

3.3.2 The Derivation of Analytical Solutions

Comparing Figure 3-15 with Figure 3-1 or Figure 3-8, we can find that the probability of the idle state (Q_{idle}), instead of Q_1, serves as the delegate state. The derivation of the corresponding analytical solutions is given in Appendix III. In Table 3-7, we conclude the analytical solutions for the model in Figure 3-15.

Table 3-7: Analytical Solutions for DPM Systems with Multiple Inactive Modes

$Q_{sd(n)} = \frac{\varepsilon}{\lambda+\gamma}(\frac{\lambda}{\lambda+\gamma})^n Q_{idle}$	Equation 3-27
$Q_{n*} = \frac{\varepsilon}{\lambda}[1-(\frac{\lambda}{\lambda+\gamma})^{n+1}]Q_{idle}$	Equation 3-28
$Q_{wu(n)} = \frac{\gamma Q_{sd(N)}}{\lambda+\delta}\sum_{k=0}^{n-N}[(\frac{\lambda}{\lambda+\gamma})^k(\frac{\lambda}{\lambda+\delta})^{n-N-k}]+(\frac{\lambda}{\lambda+\delta})^{n-N+1}Q_{(N-1)*}$	Equation 3-24
$Q_{toff(n)} = \frac{\lambda+\varepsilon}{\alpha}(\frac{\lambda}{\lambda+\alpha})^n Q_{idle}$	Equation 3-29
$Q_{ton(n)} = \frac{\alpha Q_{toff1}}{\lambda+\beta}\sum_{k=0}^{n-1}[(\frac{\lambda}{\lambda+\beta})^k(\frac{\lambda}{\lambda+\alpha})^{n-k-1}]+(\frac{\lambda}{\lambda+\beta})^n Q_{idle}$	Equation 3-30
$Q_n = \sum_{k=0}^{n-1}(\frac{\lambda}{\mu})^k Q_1 - \frac{\beta}{\mu}\sum_{k=1}^{n-1}\sum_{s=0}^{k}(\frac{\lambda}{\mu})^{k-s-1}Q_{ton(k)}$ $(n\leq N)$	Equation 3-31
$Q_n = \sum_{k=0}^{n-1}(\frac{\lambda}{\mu})^k Q_1 - \frac{\beta}{\mu}\sum_{k=1}^{n-1}\sum_{s=0}^{k}(\frac{\lambda}{\mu})^{k-s-1}Q_{ton(k)} - \frac{\delta}{\mu}\sum_{k=N}^{n-1}\sum_{s=0}^{k}(\frac{\lambda}{\mu})^{k-s-1}Q_{wu(k)}$ $(n>N)$	Equation 3-32
$S_{sd} = \sum_{n=0}^{\infty}Q_{sd(n)} = \frac{\varepsilon}{\gamma}Q_{idle}$	Equation 3-33
$S_i = \sum_{n=0}^{N}Q_{n*} = \{\frac{N\varepsilon}{\lambda}-\frac{\varepsilon}{\gamma}[1-(\frac{\lambda}{\lambda+\gamma})^N]\}Q_{idle}$	Equation 3-34
$S_{wu} = \frac{\varepsilon}{\delta}Q_{idle}$	Equation 3-35
$S_{toff} = \sum_{n=0}Q_{toff(n)} = \frac{(\lambda+\varepsilon)(\lambda+\alpha)}{\alpha^2}Q_{idle}$	Equation 3-36
$S_{ton} = \sum_{n=0}Q_{ton(n)} = \frac{\lambda}{\beta}\frac{\lambda+\alpha+\varepsilon}{\alpha}Q_{idle}$	Equation 3-37
$S_a = \{\frac{\lambda^2}{\mu-\lambda}[\frac{1}{\beta}+\frac{\lambda+\alpha+\varepsilon}{\alpha(\mu-\lambda)}+\frac{\lambda+\varepsilon}{\alpha^2}+\frac{\lambda+\varepsilon}{\alpha\beta}]+\frac{\lambda(\lambda+\alpha+\varepsilon)}{\alpha(\mu-\lambda)}-\frac{\lambda(\lambda+\alpha)(\lambda+\varepsilon)}{\alpha(\mu-\lambda)^2}$	
$+\frac{\varepsilon}{\mu-\lambda}[\frac{\lambda}{\delta}+\frac{\mu}{\mu-\lambda}+\frac{\lambda}{\gamma}(\frac{\lambda}{\lambda+\gamma})^N]\}Q_{idle}$	Equation 3-38

3.3.3 Performance Analysis

An example study has been carried out using the IBM HDD parameters in Table 3-5 including the idle mode. The result in Figure 3-16 shows that low power can be achieved by either increasing the accumulation limit *N* or increasing the value of *ε,* both of which can make the SP spend more time in the sleep mode than the idle mode. However, the latency curve gives the opposite conclusion and system engineers may end up using Equation 3-11 to balance their power & latency interests.

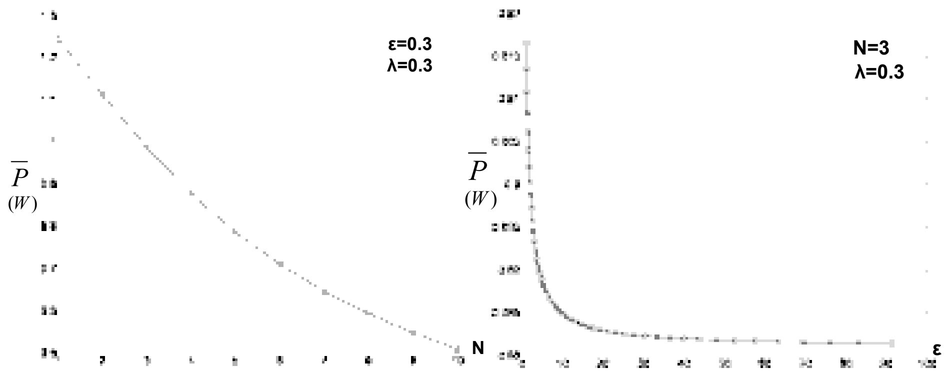


Figure 3-16: $\overline{P}$ with the Variation of *N* and ε

3.3.4 Further discussion

If more inactive states can be provided by an SP, the corresponding DPM system can provide more flexible power management for transitions from different inactive modes to the active mode, and/or for transitions among inactive modes. Besides the power management solution given in Figure 3-15, another possible solution is to use the A&F policy to control both the wakeup and turn-on transitions (Figure 3-17). Different accumulation limits (*N* for the A&F policy in the wakeup transition and *S* for that in the turn-on transition) can be implemented to control different transitions. When *S*>1, tasks corresponding to incoming events must be accumulated to activate the SP even when the latter is in its idle mode. Therefore, a group of idle states idle0, idle1 and so on are used in Figure 3-17 to describe the behaviour of the SP in the idle mode. The system engineers can use the same method introduced in this section to derive the analytical solution for this model, and find the optimal value of both *N* and *S* to achieve the optimized system performance in both power and latency.

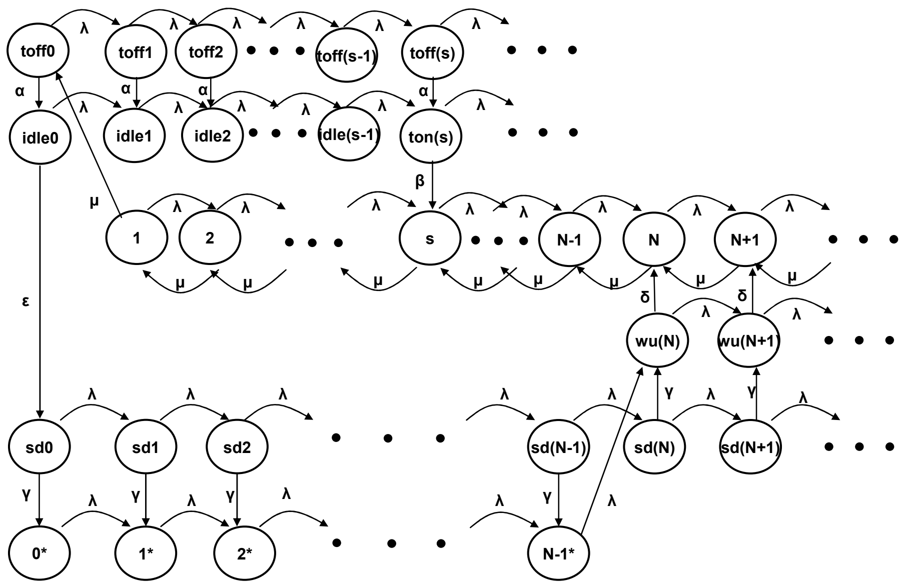


Figure 3-17: Markov Model for DPM Systems with Multi A&F Policy Control

The models given in this section can also be extended when more inactive modes are involved. From Table 3-5, it can be seen that another inactive mode ‘standby’ can be provided by the IBM HDD.

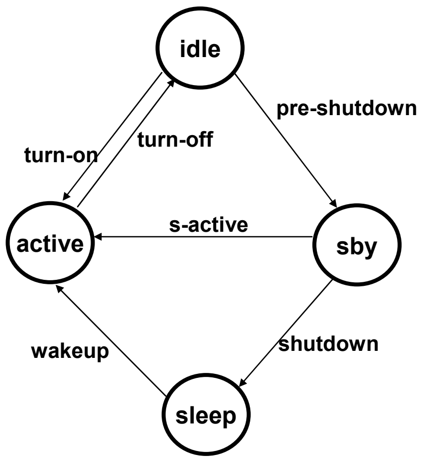


Figure 3-18: Mode Switching in DPM System with Enabled Standby Mode

Compared with the idle mode, the HDD under its standby mode consumes even less power while needing longer response time to be activated. Therefore, one possible power management solution is given in Figure 3-18. Once the TQ is empty, the SP starts a turn-off transition and move to the idle mode when the turn-off transition is complete. When the greedy policy is implemented in the idle mode, the SP is turned on as soon as one new event arrives. Without incoming events, the SP may be switched to the standby mode (which is shown as sby in Figure 3-18) and this transition is called **pre-shutdown** in our research. If the standby mode is also managed by the greedy policy, the SP is switched to the active mode when a new event arrives (Transition **s-active**), or to the sleep mode (Transition wakeup) otherwise. Similar to previous models like Figure 3-15, the A&F policy is used to control the wakeup transition and active the SP when enough events have been accumulated.

If α_1, β_1 represent the execution rate of pre-shutdown and s-active respectively, and ε_1 represents the rate of shutdown the SP to sleep mode, the Markov model for a DPM system under the control of Figure 3-18 is given in Figure 3-19.

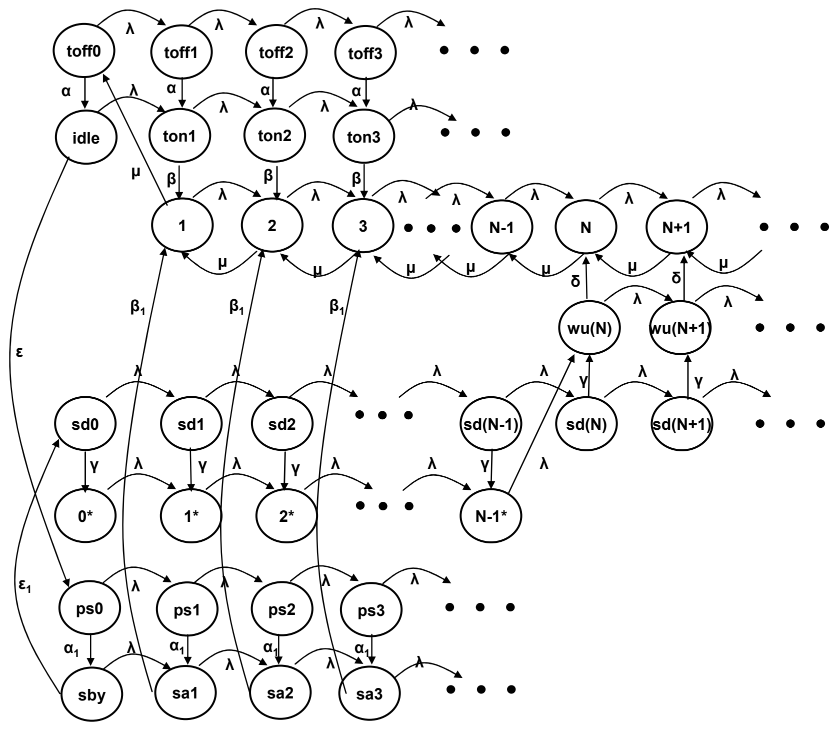


Figure 3-19: Markov Model for DPM Systems with Three Inactive Modes

3.4.DPM Systems with Multiple Active Modes

If more than one active mode is enabled in the DPM system ($A=\{M_i| i=1, 2, \ldots, r-1\}$ ($r>1$)), the SP can choose different execution speeds to process tasks with different power/latency requirements. DVS and DFS are the main techniques to switch operation modes. As introduced in Section 3.1, [karg05] gives a stochastic model of DVS with the pessimistic feature. In our work, we try to give the Markov model for the DVS system with the optimistic feature, which may have much wider implementation in real systems.

3.4.1 Markov model description

An example case is given where one sleep mode (M_0) and two work modes (M_1, M_2) are provided by an SP. The A&F policy is implemented to control the wakeup transition. Once activated, the SP first uses a low execution rate μ_L to process tasks in the TQ. Work mode M_2 is used only when the length of TQ is longer than some threshold H, and the SP in this case does task execution with a higher speed μ_H until the length of TQ becomes shorter than H. The SP moves to sleep when the TQ is empty and be activated until N new events are accumulated. Figure 3-20 gives the Markov model when $N \le H$ and the model for $N>H$ is in Figure 3-21.

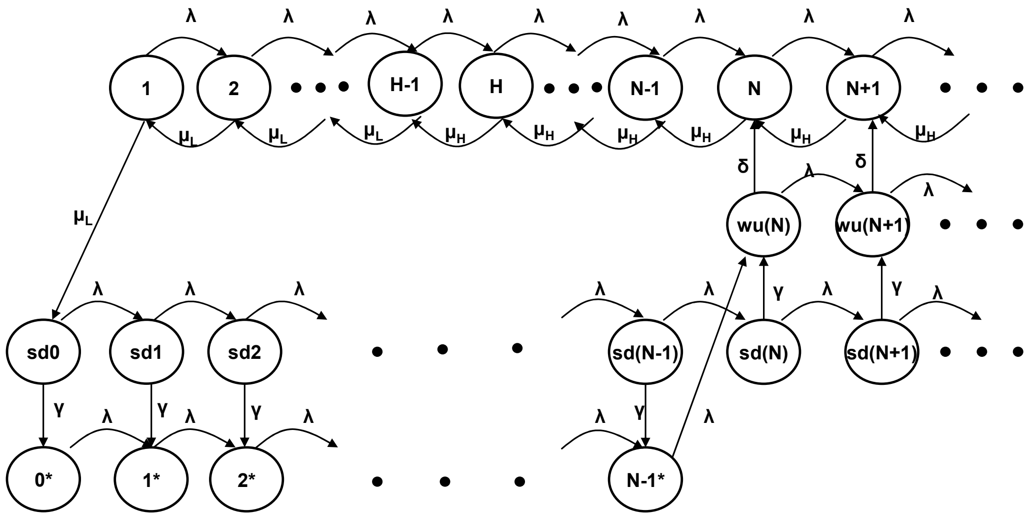


Figure 3-20: Markov Model for DPM/DVS System (*N*>*H*)

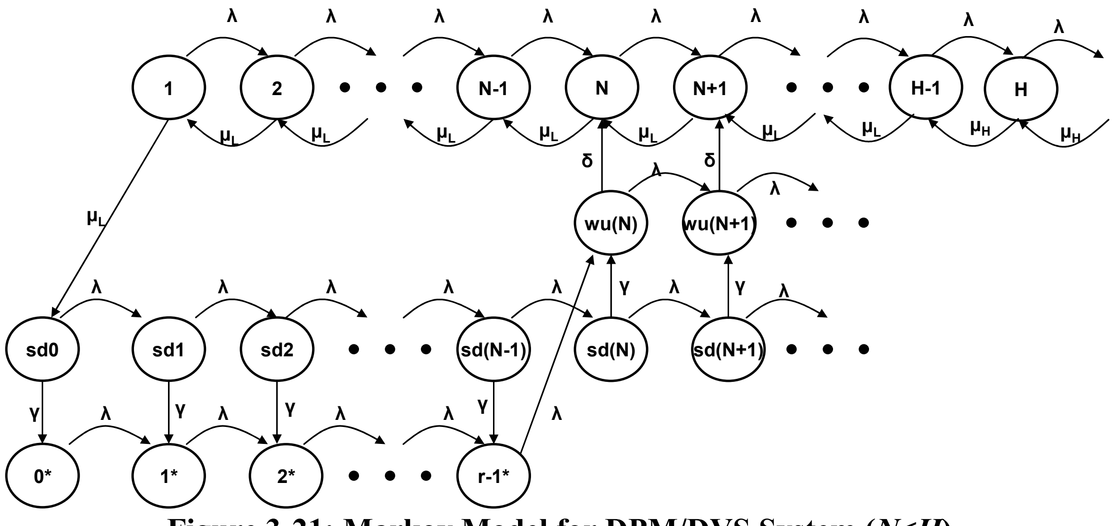


Figure 3-21: Markov Model for DPM/DVS System (*N*<*H*)

3.4.2 The Derivation of Analytical Solution

Similar to the derivation in the on-off DPM system, active state 1 is chosen as the delegate state, and the derivation of the analytical solution is given in Appendix IV. In Table 3-8, we list the analytical solutions of the model in Figure 3-20. When H=1 and $\mu_H = \mu_L = \mu$, the equations given below becomes their counterparts in Table 3-6.

Table 3-8: Analytical Solutions for DPM Systems with Two Active Modes

$Q_{sd(n)} = \frac{\mu_L}{\lambda+\gamma}(\frac{\lambda}{\lambda+\gamma})^n Q_1$	Equation 3-39
$Q_{n*} = \frac{\mu_L}{\lambda}[1-(\frac{\lambda}{\lambda+\gamma})^{n+1}]Q_1$	Equation 3-40
$Q_{wu(n)} = \frac{\gamma Q_{sd(N)}}{\lambda+\delta}\sum_{k=0}^{n-N}[(\frac{\lambda}{\lambda+\gamma})^k(\frac{\lambda}{\lambda+\delta})^{n-N-k}]+(\frac{\lambda}{\lambda+\delta})^{n-N+1}Q_{(N-1)*}$	Equation 3-41
$Q_n = \sum_{n=1}^{H-1}(\frac{\lambda}{\mu_L})^{n-1}Q_1$ (*n*<*H*)	Equation 3-42
$Q_n = (\frac{\lambda+\mu_L}{\mu_H})Q_{n-1} - \frac{\lambda}{\mu_H}Q_{n-2}$ (*n*=*H*)	Equation 3-43
$Q_n = \sum_{k=0}^{n-H}(\frac{\lambda}{\mu_H})^k Q_H - \frac{\lambda}{\mu_H}\sum_{k=0}^{n-H-1}(\frac{\lambda}{\mu_H})^k Q_{H-1}$ (*H*<*n*≤*N*)	Equation 3-44
$Q_n = \sum_{k=0}^{n-H}(\frac{\lambda}{\mu_H})^k Q_H - \frac{\lambda}{\mu_H}\sum_{k=0}^{n-H-1}(\frac{\lambda}{\mu_H})^k Q_{H-1} - \frac{\delta}{\mu_H}\sum_{k=N}^{n-1}\sum_{s=k}^{n-1}(\frac{\lambda}{\mu_H})^{n-s}Q_{wu(k)}$ (*n*>*H*)	Equation 3-45

$S_{sd} = \frac{\mu_L}{\gamma} Q_1$	Equation 3-46
$S_i = \frac{\mu_L}{\lambda}[N - \frac{\lambda}{\gamma}(1-(\frac{\lambda}{\lambda+\gamma})^N)]Q_1$	Equation 3-47
$S_{wu} = \frac{\mu_L}{\delta} Q_1$	Equation 3-48
$S_a = \{\frac{\mu_L(H-1)}{\mu_L-\lambda} - \frac{\lambda\mu_L}{(\mu_L-\lambda)^2}[1-(\frac{\lambda}{\mu_L})^{H-1}] + \frac{\lambda\mu_L[1-(\frac{\lambda}{\mu_L})^{H-1}]}{(\mu_H-\lambda)(\mu_L-\lambda)} + \frac{\mu_L}{\mu_H-\lambda}[N-H+1+\frac{\lambda}{\sigma}+\frac{\lambda^{(N+1)}}{\gamma(\lambda+\gamma)^N}]\}Q_1$ $(\lambda \neq \mu_L)$	Equation 3-49
$S_a = \{\frac{(H-1)H}{2} + \frac{\lambda(H-1)}{\mu_H-\lambda} + \frac{\mu_L}{\mu_H-\lambda}[N-H+1+\frac{\lambda}{\sigma}+\frac{\lambda^{(N+1)}}{\gamma(\lambda+\gamma)^N}\}Q_1$ $(\lambda = \mu_L)$	Equation 3-50

For the case $N<H$, the derivation of S_a is more complicated. One approximate analytical solution is just changing the position of N with H in Equation 3-49 or Equation 3-50. This approximation does not cause much difference from the accurate solution especially when $\lambda<\mu_L$ and we use this approximation value in our later analysis.

3.4.3 The Performance Analysis

For DPM systems with multiple active modes, the corresponding power-saving as well as latency depends on two parameters N and H. Large N and/or H brings low power dissipation and long latency in different degrees. Optimized system performance may be achieved by adjusting H as well as N.

Table 3-9: The Parameters of Example DVS System

f_L(MHz)	Pow_1(mW)	$Energy_{0,1}$(mJ)	$\chi_{0,1}$(KHz)	Pow_0(mW)
152	53	1.6	0.5	0
f_H(MHz)	Pow_2(mW)	$Energy_{1,0}$(mJ)	$\chi_{1,0}$(KHz)	
380	500	0.6	0.5	

For a case study using these models, we use an example SP whose key parameters are given in Table 3-9. Most of these parameters are based on information from IBM PowerPC 405LP [nowk02] in order to make the case study realistic.

In the study, μ_H is set to 1 and μ_L is set to 0.4 according to the relationship between f_H and f_L. Figure 3-22 displays the system performance with various parameters H and N. Figure 3-22(a) shows the power dissipation when λ=0.3, and it can be seen that the power dissipation drops when N and/or H increase. Next, we calculated the Balance value according to Equation 3-11 in order to find the parameter for the best performance. According to Equation 3-11, the optimized performance comes from the minimum value of Balance (λ, N, H, DL). Figure 3-22(b) shows the value of 1-Balance (λ, N, H, DL) and the peak value therefore indicates the optimized performance is achieved when H=7 and N=2 (λ=0.3, DL=20/μ_H, TOL=0.4). This optimized performance varies from system to system. For example, Figure 3-22(c) shows the optimized performance is achieved at H=3, N=2 when another group of parameters are used (λ=0.5, DL=20/μ_H, TOL=0.6).

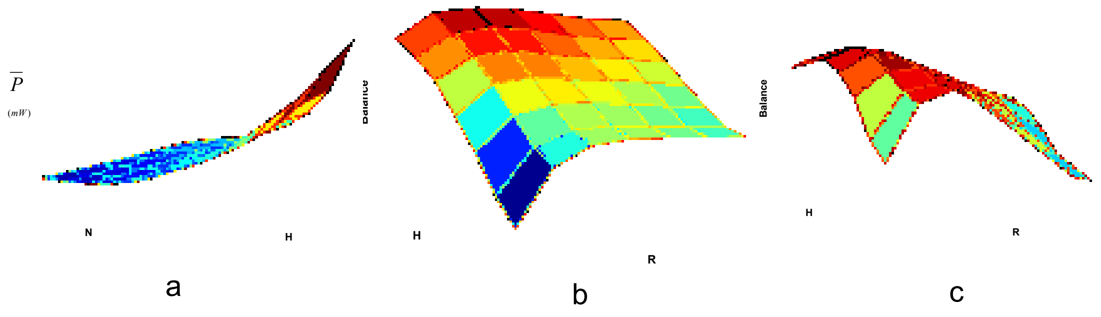


Figure 3-22: Analysis of DPM Systems with Multiple Active Modes

3.4.4 Further discussion

For an SP has three or more active modes, the Markov model in Figure 3-20 can be extended, and in this case a series of H values (H_1, H_2,... H_{r-1}, r is the number of active modes in the SP) are used and the SP changes to a new operation mode with faster speed when it reaches these H states. Although complicated, the model with multiple active modes can also be solved analytically.

In Section 3.3 and 3.4, we introduced the DPM systems with multiple inactive/active modes respectively. They can serve as the basic models that can be extended to study more complicated DPM systems, in other words, DPM systems with multiple inactive and active modes. For example, the Markov model in Figure 3-23 describes DPM systems with two inactive modes (sleep and idle) and two active modes (whose execution rates are μ_L and μ_H respectively). The models developed from the basic models given in Section 3.2 to 3.4 can help system engineers to do power-latency analysis so as to optimize system performance.

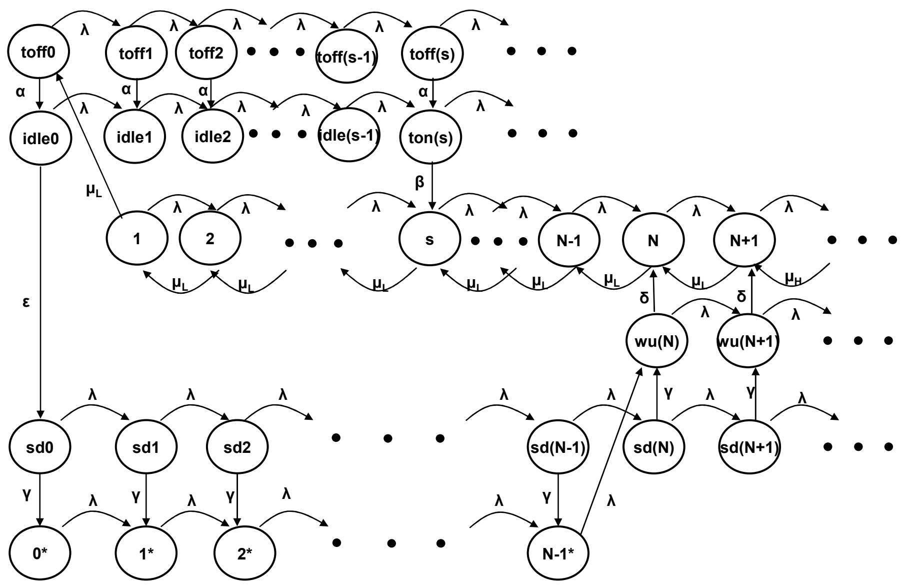


Figure 3-23: DPM Systems with Multiple Inactive/active Modes

3.5.Fine Grain Model for On-off DPM Systems

All studies in this chapter so far take the control execution in the power manager (PM) as **cost free** in both energy and latency. The PM, as the event/energy watch dog of the entire DPM system, never sleeps. Although the power dissipation in the PM is small compared with that in the SP, the former's total energy consumption may not be negligible given enough time accumulation. Furthermore, the control circuits of an SP in many portable devices have extended to include some frequent routine executions such as task scheduling. This design can first give the energy hungry SP more time to sleep. Besides, because hardware scheduling can be many times faster than software scheduling, this design can also reduce the system latency. In this case, a control unit (CU) is employed to provide Event Handling (EH), Power Management (PM) and Task Management (TM) to the SP. In these cases, the power consumption in a CU cannot be simply ignored. Finally, the power consumption in a PM (or a CU) varies with different DPM policies' implementation. The efficiency of a DPM policy should be judged by its power saving in the **entire** DPM system, not only that in an SP. All these reasons give us enough motivation to make a new fine-grain power analysis of DPM systems with full consideration of the cost in the CU.

3.5.1 The Fine Grain Structure of a DPM System

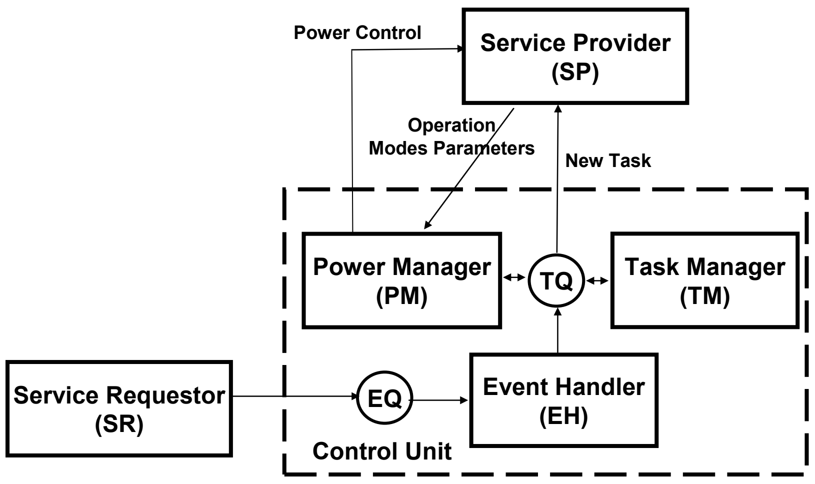


Figure 3-24: The Fine Grain Structure of a DPM System

The nondeterministic events incoming and scheduling provides enough Markovian characteristics to the CU, and it enables us to integrate the CU execution into our previous stochastic models which previously only represents the SP states. In this case, the structure of the DPM system introduced before is refined in Figure 3-24.

In such a system, events in the EQ first access the EH, where they are responded to and released after their corresponding tasks are activated and added to the TQ. Tasks in the TQ are used by the PM for the processing of a mode switching decision, and they are also used by the TM for scheduling. A new task selected by the TM is loaded into the SP for processing.

When considered as not cost free, a power on-off control can also be used to a CU itself for power saving. Although the EH should be always on to respond to the stochastically incoming events, the PM can be shut down after the SP is woken up because no power control is needed in this case. Similarly, the scheduler in the TM can be powered off when the SP is inactive. It is because the scheduling work has no meaning if no task is executed in the SP. Therefore, the execution in the CU highly depends on whether the SP is inactive or active. The length of the EQ is chosen to represent the status of the CU in the Markov model. If m is the current length of the EQ, symbol m is used to represent the status of the CU when the SP is active or carrying 'active to active' transitions and m^* is used to represent the status of the CU when the SP is inactive or carrying non 'active to active' transitions. Although some components like PM may be turned on/off during the operation of the CU for power saving, the speed for the on/off transitions is much faster (100 times or more) than the mode switching in the SP. Therefore, these on/off transitions are taken as instantaneous and no more states are used to represent them in the Markov model.

Combined with the dual indexes (n, $M_{i,j}$) for SP states representation, triple indexes (m, n, $M_{i,j}$) are used to represent the Markov states for a DPM system when the CU is not regarded as cost free.

$$\overline{P} = \sum_{i\in M} Pow_i \sum_m \sum_{n=0}^{L_T} Q_{m,n,M_i} + \sum_{\substack{i,j\in I,or\\ i\in I,j\in A,or\\ i\in A,j\in I}} Energy_{i,j}\chi_{i,j} \sum_m \sum_{n=0}^{L_T} Q_{m,n^*,M_{i,j}}$$

$$+ \sum_{i\in I} Pow_{CUI} \sum_m \sum_{n=0}^{L_T} Q_{m^*,n,M_i} + \sum_{\substack{i,j\in I,or\\ i\in I,j\in A,or\\ i\in A,j\in I}} Pow_{CUI} \sum_m \sum_{n=0}^{L_T} Q_{m^*,n,M_{i,j}} + \sum_{i\in A} Pow_{CUA} \sum_m \sum_{n=0}^{L_T} Q_{m,n,M_i}$$

Equation 3-51

With the consideration of CU cost, the average power consumption ($\overline{P}$) in Equation 3-51 is modified from Equation 3-1.

The first two components are used to calculate the power consumption in the SP and the rest are used to calculate the power consumption in the CU (Pow_{CUI} and Pow_{CUA} are the power dissipation in the CU when the SP is inactive and active respectively). The modification in latency (*APDV*) can be carried out similarly.

3.5.2 Fine grain Markov model for on-off DPM system

In this section, we try to extend the Markov model for on-off DPM system to integrate executions in the CU. The alias names of states shown in the Markov model are given in Table 3-10.

Table 3-10: Alias Index Used in Fine Grain On-off DPM Systems

Standard	(m^*, n^*, M_0)	(m, n, M_1)	$(m, n^*, M_{0,1})$	$(m^*, n^*, M_{1,0})$
Alias	mn^*	mn	wu(mn)	sd(mn)

In the example model, the A&F policy is implemented in the DPM system. With the explicit representation of CU execution, the number of states in the Markov models increases from n to n^2. This makes it difficult to show the full model in one figure. Therefore, each of Figure 3-25 to Figure 3-28 describes one single tile of the fine-grain Markov model of the A&F policy.

Figure 3-25 shows the inactive states of the new fine-grain model. The black nodes in the figure are the inactive states while the white nodes are the connected states in other groups. Suppose one event comes when the system is in the state 00*. The EQ becomes 1 and the TQ keeps 0 so the system moves to state 10*. The arrival of the new event triggers the CU to start processing with the rate of μ_1. If the processing in the CU is completed before the arrival of the next event, the system moves to the state 01*. Otherwise, it moves to the state 20*. The more events that have been executed in the CU, the more tasks accumulate in the TQ. Given enough time, the system may reach state $i(N\text{-}1)^*$ ($i>0$), and the PM activates the SP as soon as one more task is added to the TQ (The meaning for parameters like λ,δ,γ and so on have been introduced in Section 3.2.1).

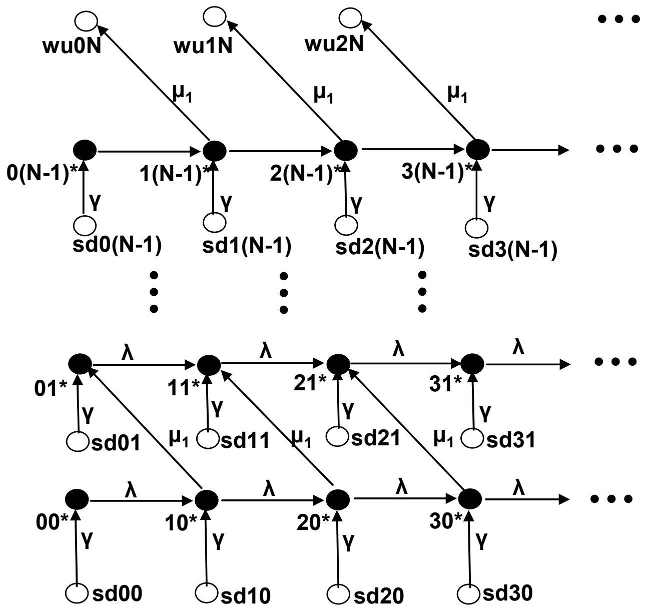


Figure 3-25: The Tile of Inactive States

The tile of wakeup states of the model is given in Figure 3-26. Once there are N tasks in the TQ, the SP starts its wakeup process and the PM part of the CU is shut down to save power. At the same time, TM starts to sort tasks for the execution in the SP. The parameter μ_2 is used as the execution rate of the CU in this case. If the system is in state wu*ij* (*i>0, j>N*), its state movement has three possible directions: It may move to the state wu(*i*+1)*j* when one new event comes, or move to the state wu(*i*-1)(*j*+1) when the scheduling of *j*+1 tasks is complete, or move to the active state *ij* when the wakeup process finishes.

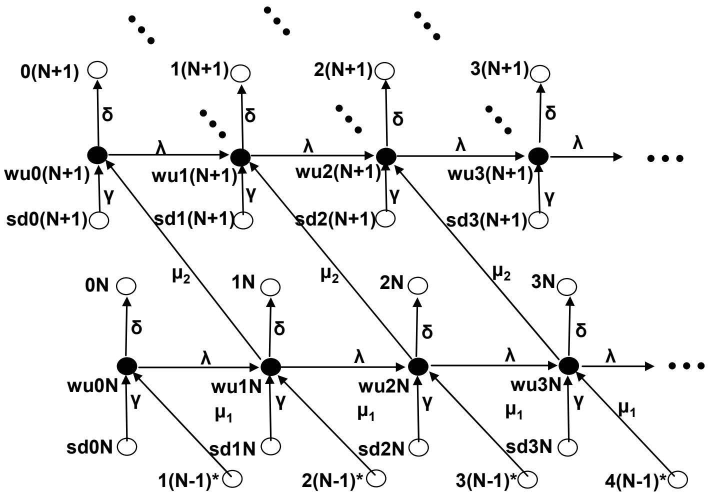


Figure 3-26: The Tile of Wakeup States

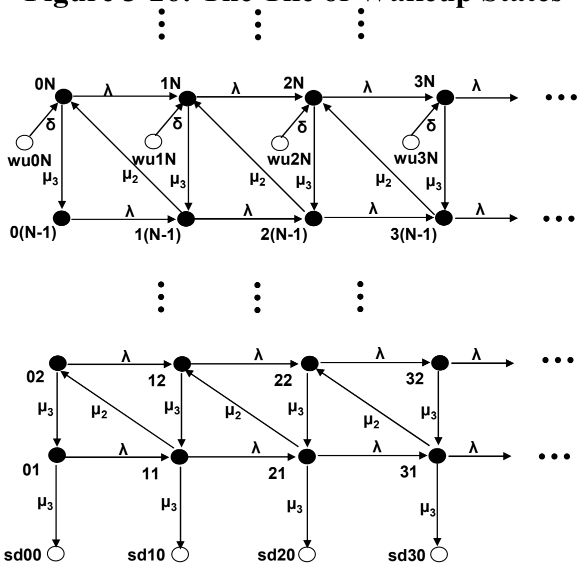


Figure 3-27: The Tile of Active States

Figure 3-27 describes the behaviour of the system when the SP is active. The parameter μ_3 is used as the execution rate of the SP. The system in the state *ij* (*i*>0, *j*>0) can move to one of three neighbour states: It may move to the state *i*(*j*-1) when the execution in the SP is completed, or move to the state (*i*-1)(*j*+1) when the CU completes a new task scheduling, or move to the state $(i+1)j$ when a new event arrives in the system. If the system moves to the state $i1$, the SP is shut down and move to the shutdown state sd$i$0 when the execution of the last task is complete.

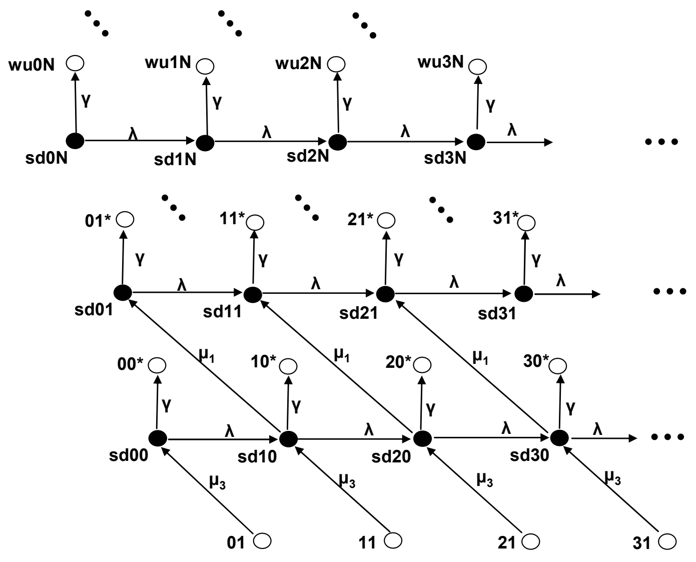


Figure 3-28: The Tile of Shutdown States

Figure 3-28 is the last tile of the system, which reflects the movement of the system when the SP is shutting down. The TM is powered off because no more service is provided in the SP and the PM is activated again to carry out A&F calculation. Therefore, the execution rate in the CU becomes μ_1 again. If the system is in state sdij ($j<N$), it moves to inactive state $ij*$ when the shutdown processing is complete. On the other hand, if the system is in state sdij ($j \geq N$), the SP is activated immediately after the shutdown processing finishes because N or more tasks are already in the TQ.

3.5.3 The Derivation of Analytical Solutions

Different from the Markov models introduced before, no one delegate state can be found whose probability can represent the probabilities of others. Therefore, only numerical solutions can be achieved given the length of the TQ in the calculation.

3.5.4 Performance Analysis

In previous sections, we show that the A&F policy can trade more latency for power when the CU is taken as cost free. However, when this assumption cannot be satisfied, the analysis becomes complicated. When a simple DPM policy is implemented, the cost in the CU is small. For example, when the greedy policy is used for the on/off control, the PM needs no more than a group of OR gates to make the activation decision. Both the latency and power cost in the PM is extremely small. On the other hand, complicated computation is needed when some advanced policy is implemented. For example, when the A&F policy is used (N>1), adders are needed in the PM circuits to calculate the length of TQ, and arbiters [byst00] are also used to deal with the metastability caused by simultaneous arrival of events. The complexity in PM circuits increases both power dissipation and system latency. The fine grain model introduced in this section can be used to balance the trade-off between the overhead in the CU and power saving in the SP, and find the optimized policy which can reduce the power consumption in the entire system.

To demonstrate the usage of the fine grain model, the example SP in Table 3-5 is used again in the analysis (suppose only sleep and active modes are used). Table 3-11 gives the parameters of the CU for different policies.

Table 3-11: Parameters of the CU

Greedy Policy					
μ_1	1000	μ_2	100	P_{CU}	141.6mW
A&F Policy (N=4)					
μ_1	100	μ_2	100	P_{CU}	200mW

In our analysis, we simply assume the maximum length of TQ (and therefore EQ) is 7 and further coming events are discarded when the TQ is full. Figure 3-29(a) describes the power consumption in the example DPM system when the greedy policy is implemented. When the straight line in the top of the figure stands for P_W, (1) is the

gross power gain of the DPM system, and (2) is the **net** power gain of the system. The power overhead paid in the CU is given in (3).

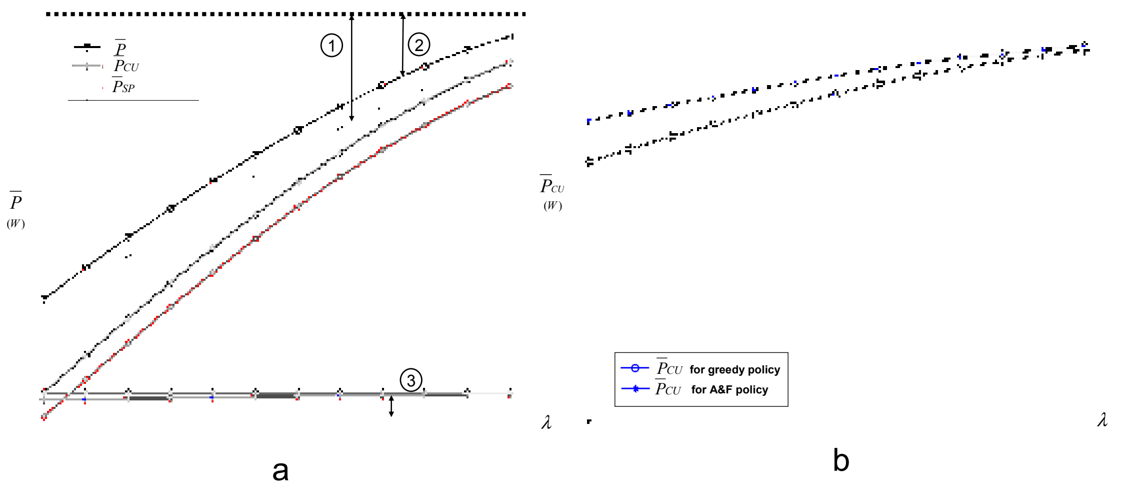


Figure 3-29: Power Analysis of Fine Grain DPM models

Figure 3-29(b) compares the power overheads in the CU ($\overline{P}_{CU}$) when different policies have been implemented. It is clear that the A&F policy CU (*N*=4) consumes more power than the greedy policy CU. However, the overhead paid for the A&F policy can be worthwhile because the $\overline{P}$ of the A&F policy is smaller than that of the greedy policy even when the power cost in CU is taken into consideration (Figure 3-30).

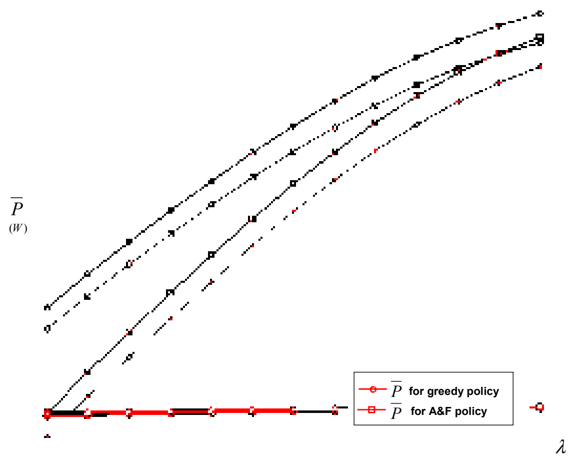


Figure 3-30: Power Cost of Different Policies

The fine-grain model can also help analyze of the system latency. For example, as shown in Figure 3-31, when the curve in the bottom of the figure represents the *APDV* value (*DL* is set to $10/\mu_3$) when the SP is always on, (1) and (2) serves as the gross and net extra latency cost by DPM control when the A&F policy (*N*=4) is implemented.

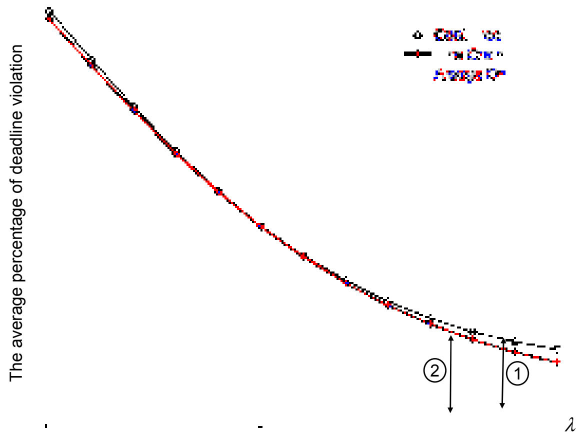


Figure 3-31: The Latency Performance of A&F Policy (*N*=4)

Conclusions and Further Discussion

In this section, the fine grain model of an on-off DPM system was presented and analyzed. When some advanced processor serves as the SP in a DPM system, the execution of the PM is extended from simple on-off decisions to managing the switching among various modes provided by the SP. Markov modelling of such DPM systems when the PM is taken as cost free has been thoroughly explored in section 3.2.2 and section 3.2.3. The fine grain model designed for the on-off DPM systems can be easily extended to represent DPM systems with multiple inactive and/or active modes.

The fine-grain Markov model in this section can also help hardware designers to improve their circuits. The same high-level DPM policy can have various implementation circuits in the PM/TM with different power/latency parameters. The analysis given in this section can help hardware designers to compare the performance of these circuits as well as their influence on the entire system.

With DPM control, IP cores are triggered by incoming events, such as data, signal and energy tokens. For the more explicit energy driven systems [kans03], the SP, which stands for the main processor in the system, can provide the execution only when enough energy is available. The CU in this case represents the energy-harvesting unit which is always alert to the environment and carries out the harvesting execution when the energy that can be collected is higher than that is consumed in the task execution. Energy instead of data events is accumulated in this unit and the activation of the main processor depends on this accumulation. We believe that our current model, which is derived mainly with data and signal events in mind, may be further refined and modified to better suit energy driven systems. An immediate next task is to develop a more systematic and coherent representation of harvested energy as atomic events as well as example models of CUs from real-world energy harvesting systems.

Chapter 4

Hierarchical CPN Models for a VSB

With stochastic models built in Chapter 3, the A&F policy shown to have great potential to reduce power consumption in all kinds of SPs (no matter how many operation modes they have), and even when the PM (or CU) part of a DPM system is not cost free. The Parameter *N* in the A&F policy can be used to trade off power against latency. The simple A&F mechanism makes this policy easy implement in hardware without much complexity in circuit design. Therefore, this policy is chosen as the power management policy in the design of a Self Timed Event Processor (STEP), which can help an IP core to work as a Virtual Self-timed Block (VSB) in an asynchronous SOC context.

The Fine Grain model of a DPM system introduced in Section 3.4 also discloses the high concurrency of a DPM system (now specifically a VSB). The execution of existing events in the CU (now a STEP) may be carried out concurrently with the handling of new incoming events. Similarly, the execution of tasks in the SP (now specifically an IP core) may be processed concurrently with the execution of events in the CU. When no global clock is available, a VSB has to deal with nondeterministic cases brought not only by the asynchronous environment, but also by the asynchronous operations in different parts of STEP and its IP core.

Therefore, the design of a reliable and high performance STEP becomes difficult if designers go directly into hardware design in gate level, as some important interaction patterns may be easily ignored when designing such a complex system. Some potential flaws brought by the nondeterministic nature of the environment may not happen frequently, but the lack of an adequate solution brings great hidden danger for the implementation of STEP. It is essential to provide methods that enable debugging and testing of the entire system (or at least some central parts of the system) prior to implementation and deployment.

Coloured Petri Nets (CPN) is a language for the modelling and validation of systems in which concurrency, communication, and synchronisation play a major role [jens07]. The construction and analysis of CPN can be carried out in CPN Tools, which is an industrial-strength software tool. Users can take advantage of this tool to simulate the behaviour of the modelled system, and to verify its properties by means of state space methods and model checking.

Modelling of a complex system is always carried out in levels. It is difficult for a system designer to implement every detail of its system in a complex model. If some errors are contained in the model, it is time consuming to find the real cause of an error, because the error may have propagated through the system. Therefore, dividing a complex system into different modules in a hierarchy and using the top-down design flow to carry out the model design is an efficient modelling method. Besides, models at different levels of a design hierarchy have their own usages. Models in the lower or lowest level of the hierarchy present a clear and detailed description of executions of the corresponding system. They can be mapped to hardware components and circuits using direct mapping [shan02] or synthesis [sing06]. These models are implementation oriented and must be re-designed if the corresponding implementation is changed. Models at the higher level of the hierarchy, although abstract and having no direct connection with hardware circuits, remain robust when the implementation changes, and can be used for the analysis of similar systems different only in some detail.

Although many solutions can design models in a hierarchy, CPN is chosen because it can clearly show the concurrent execution in different parts of one model. With CPN models, the design idea of STEP with the implementation of the A&F policy can be realized in the form of signals and data processing. Simulations and State space checking is used to prove the correctness of CPN models at different levels.

4.1.A top Level Model of a Virtual Self-timed Block in CPN

According to the introduction in Section 3.4, a CU (now a STEP) is roughly divided into an **Event Handler** (EH) for handling stochastically incoming events, a **Power Manager** (PM) which gives mode switching decisions to the IP core and a **Task Manager** (TM) which selects a suitable task from the TQ and loads it to the IP core when necessary. Although the A&F policy can be implemented into all kinds of DPM systems, currently we assume only a simple IP core with just on/off modes to be implemented in a VSB design. A top level CPN model (Figure 4-1) is built in this section to show the basic connection among different parts of a VSB.

4.1.1 The Color Set Description

Three colours have been declared and used in this model. Colour **BIT** is declared to have only value ‘0’ and ‘1’, which is used to indicate the signal changes in a VSB. When the corresponding system is a hardware system, the colour BIT becomes the fundamental colour in the CPN model, and places with the other colours used in high level model like Figure 4-1 will be eventually decomposed into a combination of places with BIT colour set in the lowest level model.

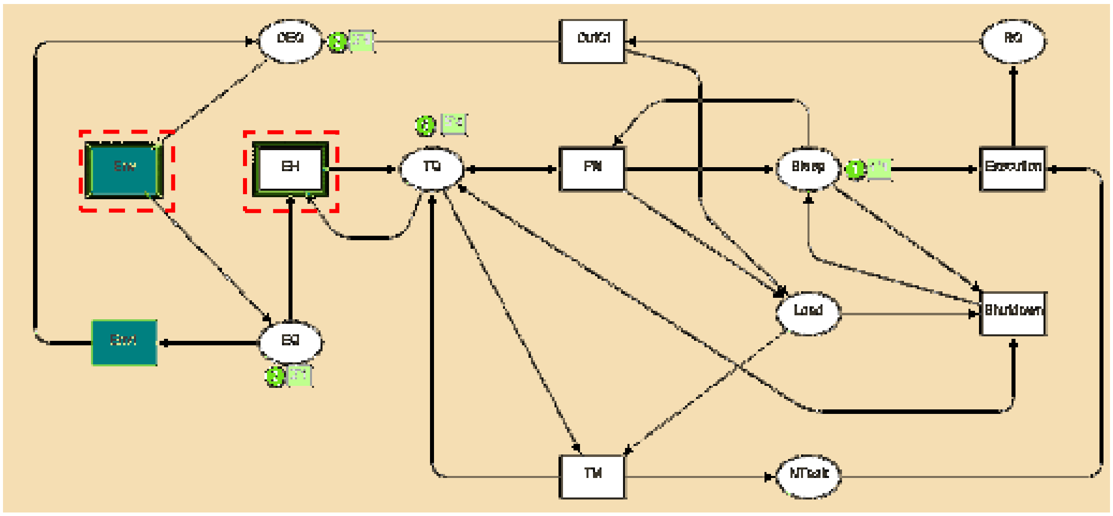


Figure 4-1: Top Level Model of a VSB

Colour **TASK** is declared to represent tasks that can be carried out in the IP core. In the top level model, all tasks have the same abstract meaning. Therefore, the TASK colour is declared as BIT whose token value represents whether the task is ready for execution in the IP core (value ‘1’) or not (value ‘0’). In the lower level model, the TASK colour should be specified when the execution of different tasks is not identical. In this thesis, all tasks are assumed to be independent, which means any execution sequences of these tasks are acceptable (although different execution sequence may bring variation in system performance).

Colour **EVENT** is declared to represent events accessing VSBs in an SOC frame. Similar to the declaration of colour TASK, EVENT is declared as BIT in the top level model since all events are identical. In this thesis, an event is an open concept. It can represent energy, data processing requests, communication requests or others. Therefore, the abstract declaration of colour EVENT in the top level model helps the model to have a wider representation, and the colour of EVENT can be re-declared according to the specified concept of EVENT in different implementations. Different from the colour TASK, an EVENT '1' token represents an event arrived in the current VSB and an EVENT '0' token indicates either an event for the current VSB is not ready, or an event that has no relationship with the current VSB.

4.1.2 The Model Description

After the introduction of the colour declaration, we can now describe the model in Figure 4-1. Tokens in the place **EQ** represent all incoming events waiting to be responded to by a STEP. Similarly, the place **TQ** represents the status of all tasks that can be executed in the IP core. An initial token L`0 is attached to the TQ place. The integer constant L is used to indicate the number of tasks embedded in the IP core of the current VSB. The initial value '0' indicates that all L tasks are not ready for execution and waiting for the incoming of their corresponding events.

When there is at least one '1' EVENT token in the EQ place and one '0' TASK token in the TQ place, the transition **EH** is enabled. The occurrence of this transition removes one EVENT token from the EQ place, indicating that 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 task is ready for execution. When a VSB is designed for data processing, the possible asynchronous/synchronous data transform performed in an asynchronous wrapper is also represented by the occurrence of the EH transition.

The EH transition is enabled again when further events arrive, and each occurrence updates one '0' token in the TQ place to '1'. When the *i*`1++(*L*-*i*)`0 marking shows in the TQ place, it means that *i* out of *L* tasks in the IP core are ready for execution. However, when no more '0' tokens can be found in the TQ place, it means that all tasks provided by the IP core are ready for execution. If new events come at this time, they cannot be responded to by the STEP until their corresponding tasks have been processed in the IP core. Therefore, the EH transition becomes disabled when no '0' TASK token is available in the TQ place.

The place **Sleep** is declared as BIT colour set and its token indicates the status of the IP core in the current VSB. By the initial token '1', the IP core is indicated to be inactive initially. When the A&F policy is implemented in the STEP, another constant integer *N* is declared in the model. Therefore, the transition **PM** is enabled only when there are more than *N* TASK '1' tokens in the TQ place. The TQ place and the PM 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 occurrence of the PM transition toggles the token value in the Sleep place to '0', which indicates the wakeup processing in the IP core. It adds one BIT '1' token to the **Load** place as the consequence of the completion of the wakeup processing. As mentioned in Section 3.5, the A&F policy is only used when the IP core is inactive and the PM circuits should be shut down to save power as soon as the IP core is activated. This design idea is reflected by the expression of the arc directed from the Sleep place to the PM transition, which indicates the latter transition can only be enabled when the token value in the Sleep is ‘1’.

Although the first occurrence in the EH transition adds one ‘1’ token to the TQ place, the **TM** transition is enabled only when the occurrence of the PM transition puts one token in the Load place. It is because the TM is supposed to be powered off when the IP core is sleeping. The availability of one token in the Load place indicates the completion of the wakeup processing in the IP core, so the TM transition is enabled afterwards. The occurrence of this transition consumes one ‘1’ token in the Load place and one TASK ‘1’ token in the TQ place. It adds one TASK ‘1’ token to the **NTask** (meaning New Task) place to indicate one new task chosen from all ready tasks is loaded to the IP core for execution. At the same time, one TASK ‘0’ token is added to the TQ place, indicating the chosen task in the NTask place is not scheduled before its execution is completed in the IP core.

If the IP core is inactive, TASK ‘1’ tokens in the TQ place are used by the PM and TM transitions for occurrence in sequence. However, if the IP core is active, no task accumulation in the PM is needed and new added TASK ‘1’ token is only be used for the enabling of the TM transition.

When three execution units EH, TM and PM in the STEP are represented by three transitions with the same name in the model, their concurrent executions can be clearly represented by the simultaneous enabling of these transitions. For example, when $N<L$, the transition TM is enabled when a token is available in the Load place and the transition EH may be enabled at the same time if another EVENT ‘1’ token appears in the EQ place (Figure 4-2(b)). Concurrent executions can benefit a VSB

since parallel processing can reduce the latency of the VSB, but they may also bring hidden dangers as well. For example, two occurrence sequences exist when both TM and EH transitions are enabled simultaneously. If the EH transition occurs first, it adds another task for scheduling. While the first occurrence of the TM transition indicates the new ready task is not used during current scheduling. The random occurrence sequences of the two transitions cause no trouble in the top level model since all tasks are regarded identical, while they can bring hazards in the scheduling result when tasks are thought to be different. Therefore, a low level model should solve this hazard brought by concurrent execution in the STEP.

When one TASK token is available in the NTask place, the **Execution** transition is enabled. The occurrence of this transition indicates the execution of the current task in the IP core, and it adds one TASK '1' token to the **RQ** (means Result Task Queue) place. Generally speaking, the result generated by the execution of one task in the current VSB either releases some system resources like the data bus, I/O port or battery power, etc, or generates new data or requests. Therefore, the completion of one task can trigger some event, so as to enable some other task to be executed. Therefore, the post-processed TASK token in the RQ place is used to enable the **OutCt** (means Output Control) transition which represents all necessary preparation of a new event (for example, request signal generation, data path preparation, browsing the route table to determining the destination VSBs and so on). Finally one EVENT '1' token is added to the **OEQ** (means Output Event Queue) place and sent to the environment. The occurrence of OutCt transition also adds one token to the Load place which enables the TM transition to choose another task for the IP core's execution.

When all TASK ‘1’ tokens have been consumed in the execution transition, the **Shutdown** transition is enabled. Its occurrence toggles the token in the Sleep place to ‘1’, which means the shutdown process is completed.

Without a global clock inside the VSB, the operation in the IP core as well as in the STEP can be carried out concurrently. This concurrent execution can also be reflected by the simultaneous enabling of transitions in the CPN model. For example, transitions Execution and EH may be enabled simultaneously, which means that the STEP needs to respond to new incoming events when the IP core is processing. This concurrent processing can improve the performance of the VSB. However, some concurrency may also bring problems to the VSB. For example, when no more TASK ‘1’ token is in the TQ place and one EVENT ‘1’ token is in the EQ place are available, both Shutdown and EH transitions are enabled (Figure 4-2(c)) and different occurrences of the two transitions have different consequences. If the Shutdown transition occurs first, the new TASK ‘1’ token added by the occurrence of the EH transition is used in task accumulation. Otherwise, the new TASK ‘1’ token is directly used for scheduling in the TM transition. The random occurrence of the two transitions may bring contradictory operations in the VSB when a new task is enabled during the shutdown processing. The IP core may be confused about whether to continue the shutdown processing or start a new wakeup processing. This confusion may cause data loss or even more serious consequences and should be avoided. Therefore, some solutions should be added to the low level model of the IP core control interface so that the shutdown processing cannot be disturbed before its completion.

4.1.3 The Environmental Set Description

The places and transitions introduced so far construct the top level model of a Virtual Self Timed Block. In order to check the behaviour of the model and verify its properties, some places and transitions must be added so as to simulate the behaviour in the environment. These places and transitions are highlighted by the dark shade so as to differentiate from their counterparts describing the current system.

Equipped with environmental places/transitions, the model given in Figure 4-1 describes a closed system, and the event sent by the current VSB to the environment finally stimulates some other event to come back to the current VSB. Transitions **Env** and **Env1** are used to describe this procedure by means of an EVENT type variable **event**. Therefore, given one token in the OEQ place (no matter what value it has), the transition Env is enabled and the occurrence of this transition indicates the corresponding event is sent out to the environment. As the events incoming is modelled as a stochastic (in most cases Markovian) process, how quickly the outgoing of the event from the current VSB to the environment can stimulate an event from the environment to the current VSB is uncertain. Therefore, a random function **P()** is used in the expression of arc from the Env transition to the EQ place. The declaration of this function is given below:

fun P() = poisson (2.5)

This function uses the random number generator **poisson** provided by CPN Tools to generate a random integer number which follows the 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 features of the implementation environment.

Therefore, when the random number generated by the P() function is greater than 1, an EVENT '1' token is added to the EQ place after the occurrence of the Env transition, which means one new event arrives at the current VSB and waits to be responded to. When the random number generated by the P() function is no more than 1, an EVENT '0' token is added to the EQ place accordingly. This means the corresponding new event is not available for the usage in the current VSB. An EVENT '0' token enables the Env1 transition and the latter's occurrence moves the token to the OEQ place. Therefore the marking of *i*`1++*j*`0 in the OEQ place indicates that there are *i* events ready to be sent out to the environment from the current VSB and *j* events are relaying in the environment. How quickly the value of these EVENT tokens can become '1' in the EQ place is determined by the value of λ in the P() function.

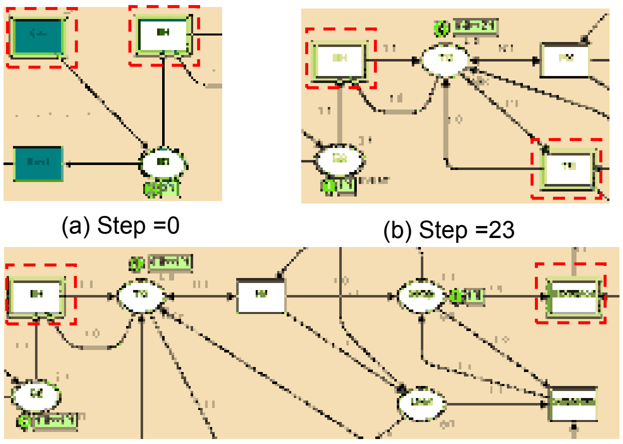
4.1.4 Initial Marking

Figure 4-1 also shows the initial marking M_0 of the model. One '0' token is given to the Sleep place suggesting that the IP core is inactive in the initial state. When *L* is set to 5, five '0' TASK tokens are given to the TQ place, which indicate none of the five tasks are ready for execution. Two '1' EVENT tokens are set to the EQ place, showing that two incoming events are waiting to be responded to by the STEP. Even when the two events are responded to by their corresponding tasks, the IP core cannot be woken up since *N* is set to 3. Therefore, the activation of the IP core needs the arrival of at least one EVENT token initially in the OEQ place.

4.1.5 Simulation

With the integration of environment places and transitions, we can use the simulation function provided by the CPN Tools to check the behaviours in the current model. Figure 4-2 shows several segments of the model when concurrency properties in different parts of the VSB are disclosed (Every enabled transition is highlighted by a dotted rectangle).

Figure 4-2(a) shows the concurrent execution in the STEP (the enabling of the EH transition) as well as that in the environment (the enabling of the Env transition). Figure 4-2(b) presents the concurrent execution within the STEP (the enabling of both the EH and TM transitions). Figure 4-2(c) shows the concurrent processing in the STEP (the enabling of the EH transition) as well as in the IP core (the enabling of the Execution transition).



(a) Step =0 (b) Step =23

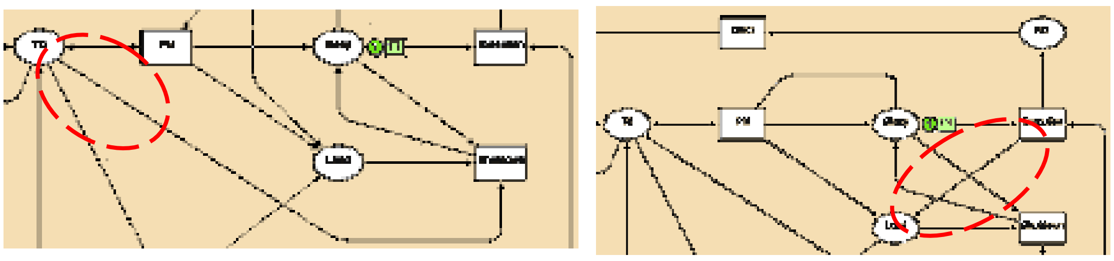
(c) Step =6

Figure 4-2: Simulation of the Top Level CPN Model

The simulation can also help the designers to correct errors in their models. For example, one double-headed arc is used to connect the Shutdown transition and the TQ place because five ‘0’ TASK tokens are checked but not consumed by the IP core to make shutdown decisions. However, designers can easily miss the arc from the Shutdown transition to the TQ place (Figure 4-3(a)), and the consumption of TASK

‘0’ tokens in the occurrence of Shutdown transition makes further enabling of the EH transition impossible.

When simulation is carried out at the top level with this error, it terminates after a certain number of steps because in that case no more transitions are enabled (called dead marking or dead lock). Therefore, a dead marking in the simulation is used to detect errors in model design stage. However, because of the randomness brought by the function P(), this simulation termination may not happen within 100 or even more steps. Five independent simulations have been carried out when the model has the design error. In these simulations, the CPN Tools took 103, 202, 159, 394, 941 steps respectively to achieve the dead marking. Few designers 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 is hidden in the design. Therefore, we need some other more reliable function tool to prove the correctness of the model.



(a) Error Case A (b) Error Case B

Figure 4-3: Possible Errors in the Top Level Model

Furthermore, the exposition of some other errors cannot be detected based on simulation termination. In Figure 4-3(b), the arc directing from the OutCt transition to the Load place in Figure 4-1 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 is enabled simultaneously with the enabling of the OutCt transition. If the Shutdown transition occurs first, the OutCt transition is disabled. It means no more executions for new events preparation are forbidden because the IP core is sleeping, and the corresponding event may be missed or duplicated sent. Unfortunately, this error cannot be found by the method of simulation termination because the concurrent enabling of both OutCt and Shutdown transitions does not make the model reach a dead marking. Errors like this are more easily ignored by the designers.

4.1.6 State Space Checking

In this case, the state space checking provided by the CPN serves as a more reliable method to find possible errors and prove the dynamic properties of the models. In our design, the BIT colour set is the most fundamental colour which may be mapped to electronic level or edge signals in hardware design. Therefore, no more than one BIT token can be held in the same place under any circumstances, and multiple BIT tokens in the same place indicate errors in the model design. The **Boundedness Properties** in the state space report can be used for this check. For places with other colour set, the Boundedness Properties shows all possible token values in one place, and the user can check if any illegal values appear in the model.

All transitions in a CPN model should be enabled at least once (otherwise the transition should be removed), therefore no transitions should be reported as Dead Transitions in the **Liveness Properties** of the state space report. In case some Dead Transitions are reported in the report, designers can use the Dead Markings given in the same report to trace the possible errors. Now we try to examine the state space report about the top level STEP model.

```
Statistics
----------------------------------------------------------------------
 Occurrence Graph                     Scc Graph
     Nodes:  177                        Nodes:  1
     Arcs:   471                        Arcs:   0
     Secs:   1                          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'RQ 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'RQ 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:  All

 Liveness Properties
----------------------------------------------------------------------
  Dead Markings:  None
  Dead Transitions Instances: None
  Live Transitions Instances: All
```

According to the report, no transition in the model is dead. The Best Integer Bounds show all BIT places hold no more than one token at any time. Therefore the corresponding circuits give no conflicting indications to the executions of a VSB. The state space report can help the user to find possible errors of the model. Figure 4-3 presents two errors which may easily happen in the design and these errors are checked out in their state space reports respectively.

When ErrorA in Figure 4-3 contains in the model, the corresponding report is shown below (all identical items with the report of the correct model are omitted):

```
Statistics
----------------------------------------------------------------------
 Occurrence Graph                     Scc Graph
     Nodes:  149                        Nodes:  35
     Arcs:   387                        Arcs:   80
     Secs:   0                          Secs:   0
     Status: Full

 Boundedness Properties
----------------------------------------------------------------------
…

 Home Properties
----------------------------------------------------------------------
  Home Markings:  [109]

 Liveness Properties
----------------------------------------------------------------------
  Dead Markings:  [109]
  Dead Transitions Instances: None
  Live Transitions Instances: None
```

It can be seen that one Dead Marking [109] is highlighted in this report and no live transitions exist in the model. Because no dead transitions exist in the model, it means that all transitions can be enabled at least once. However the occurrence of some transition causes an abnormal marking, so that no more transitions can be enabled since then. Since the dead marking is a home marking, it means this abnormal marking always happens no matter what occurrence sequences may happen. Therefore, transitions that may be concurrently enabled are highly impossible to be the cause of dead marking. This analysis can help the designer finally find the error in the arc between Shutdown transition and the TQ place.

When ErrorB in Figure 4-3 happens, the corresponding report is shown below (all identical items with the correct model report are omitted):

```
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 means that 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 that 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 into 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.

4.1.7 Conclusions and Further Discussion

A top level CPN model of a VSB (including a STEP and an IP core) was presented in this section. Both simulation and state space function tools provided by the CPN Tools were used to check the correctness of the model. The abstract declaration of both EVENT and TASK colours make the top level model robust when events are specified as different concepts in various implementations.

Although abstract, this model clearly shows the integration of the accumulation & fire policy in the operation of the STEP. It also indicates the possible concurrent executions between different parts of the STEP, or between the STEP and its corresponding IP core. This concurrency may bring parallel processing in the VSB. However, it may also cause hazards which may affect the VSB’s performance if no corresponding solutions are given.

The abstraction at the top level model prevents further discussion about the influence of concurrency to system design and gives no direct guidance to the hardware design. In the following sections, we decompose the top level model into several connected segments and use a lower level CPN model to specify each part in detail.

4.2. The Design of a Power Manager in CPN

In this section, we try to specify one segment of the top level model, which centres on the EH transition (Figure 4-4). As indicated by the top level model, this segment is mainly used to respond to incoming events and update the status of the corresponding tasks.

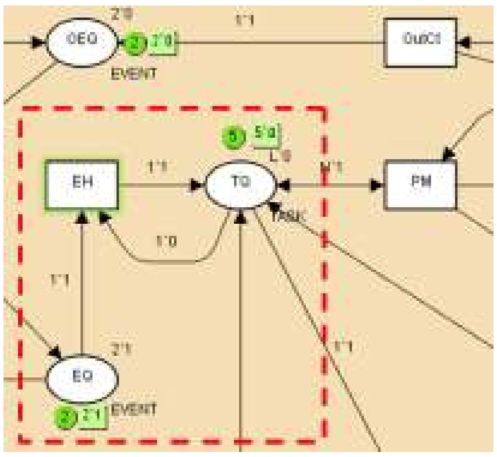


Figure 4-4: The Event Handler Segment in the Top Level Model

In the top level model, every occurrence of the EH transition can only consume one EVENT token in the EQ place. It means all incoming events must wait in a queue to be responded to by the STEP even when they arrive simultaneously from different directions. In this thesis, we assume that all STEPs are point-to-point connected, and one point-to-point connection between two STEPs is called a communication **Channel**. When a STEP has more than one input Channel, arbiter(s) become indispensible to create an event queue when simultaneous events arrival occurs. The number of arbiters used in the STEP increases with the Channels' number in C_n^2, and the corresponding circuits' costs in both power dissipation and latency increase dramatically. A better solution should enable multiple events to be responded to in parallel.

Moreover, the occurrence of the EH transition in the top level model updates the value of one task token from '0' to '1', which means every incoming event makes 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 doubled amount of information), events from different Channels are highly likely to indicate the execution of the same task (but with different parameters). 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.

4.2.1 A Matrix Structure of Event Handler

When we take the two problems into consideration, a matrix structure used in the Butler coprocessor's design [camp97] is a good reference for the design of the EH part in the STEP (Figure 4-5).

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

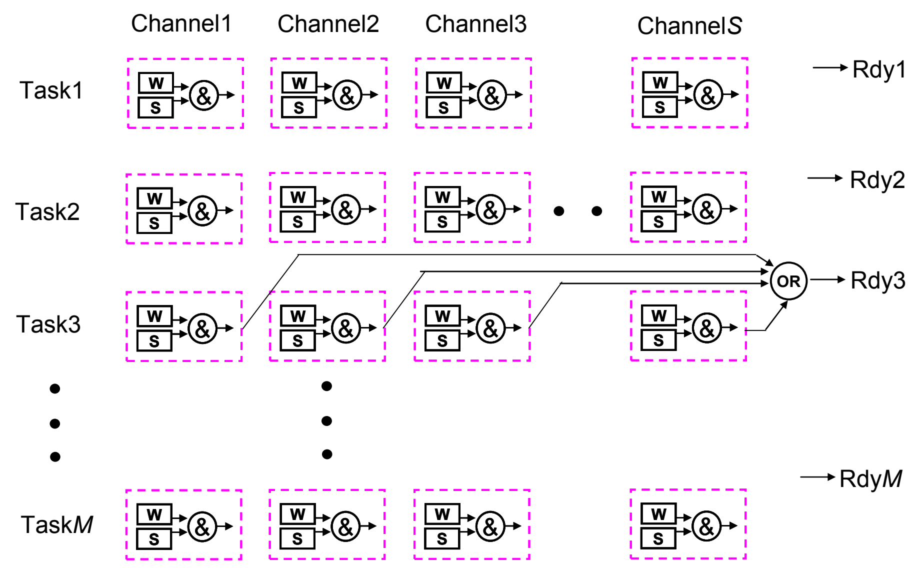


Figure 4-5: The Matrix Structure in the Butler

If there is at least one $U_{i,j}$ 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 4-5) becomes valid. All ready tasks are called **candidates** and the number of candidates is used in the Power Manager of the STEP for accumulation when the IP core is inactive and they are also used in the Task Manager for scheduling, which will be introduced in the following sections in detail. One and only one candidate can be chosen and loaded into the IP core for execution each time, and the ready signal of the corresponding task is withdrawn afterwards so that the task cannot be a candidate for next scheduling.

The structure within every $U_{i,j}$ relies on the implementation of the VSB. When the VSB is used for data processing, the execution of one task needs the combination of both operation codes and the data for operation. An incoming event in this case indicates that the corresponding data is available. The operation codes of the corresponding task are always ready for processing except when they are just under processing, or they are prevented from execution by other tasks in case of suspension, interruption or synchronization etc [masc87]. Therefore, two 1-bit variables **wait** and **stim** (which are written as W and S for short in Figure 4-5) are used in every unit of the matrix. The wait bit is set when the operation codes of the corresponding task are ready for execution, and it is reset otherwise. Similarly, the stim bit is set when the event (mainly the corresponding data) is accessible and it is reset otherwise. A task *i* becomes a candidate task and its ready signal becomes valid only when at least one $U_{i,j}$ unit of the matrix has both stim and wait bits set.

The matrix structure gives high expansibility to the STEP. When used in different environment or to cooperate with different IP cores, the parameters of the matrix *M* and *S* may be changed accordingly. However, the Event Handler part can be easily adjusted by adding/deleting several units in the matrix while the entire structure keeps the same.

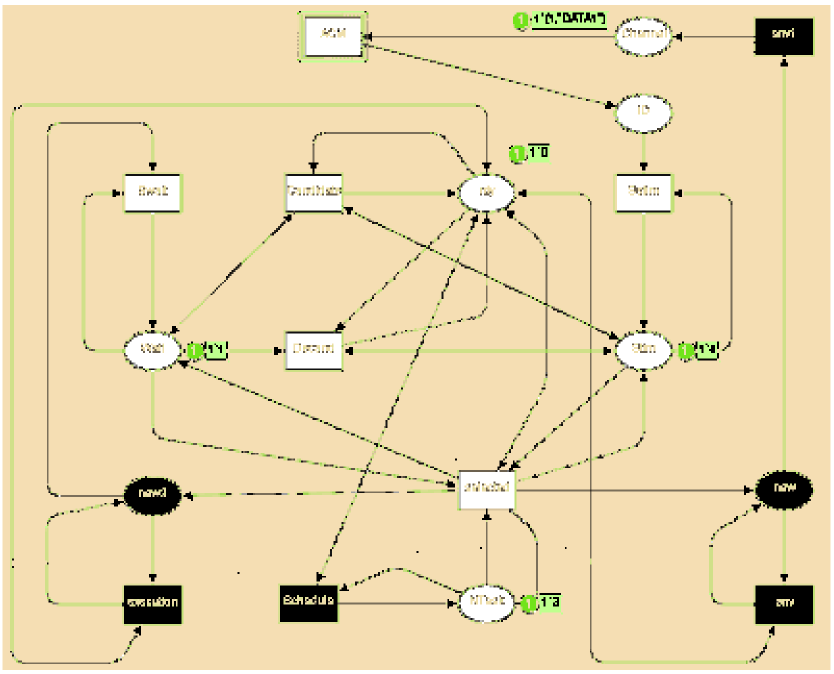


Figure 4-6: CPN Model of One Unit in the Event Handler

Although the matrix structure has been designed in the Butler coprocessor [camp97], the previous design goes directly to gate level without any modelling work. For the sake of better integration with other parts of the STEP, the model of one unit of the matrix is designed in CPN (Figure 4-6).

4.2.2 The Color Set Description

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

color TASK = int with 0 .. Max

where **Max** is a constant standing for the maximum ID number used in the current VSB.

When data is transferred among different domains with different clock frequencies, the Asynchronous Communication Mechanism (ACM) serves as an efficient and safe method used in many implementations and is used in the VSB design. Because the CPN models of ACM memories has been designed in [gorg08], an abstract **DATA** colour is declared as the colour string (as the set of all text strings) whose content is used to describe the property of the corresponding data.

color DATA = string

Therefore the colour EVENT is re-declared as:

color EVENT = product TASK*DATA

This means that an EVENT token is composed of a TASK token and a DATA token. The TASK token indicates which operation is used to process the data represented by the DATA token. The colour BIT keeps the same declaration in this model (as well as other models in the chapter).

4.2.3 Model Description

In Figure 4-6, 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 enables the **ACM** transition. This transition represents the data transfer/transform carried by the STEP when ACM is used. The specification of this transition is shown in [gorg08]. The occurrence of this transition generates a TASK token to the **ID** place, which indicates the completion of the data preparation for the task suggested by the token value.

The constant ID1 in Figure 4-6 is declared as a constant integer which represents the ID number of the task represented by the current unit. A guard [task=ID1] is attached to the transition **Sstim** (means Set stim bit) to make sure the latter can only be enabled by a TASK token (which is one part of an EVENT token) valued in '1' (ID1 is currently declared as 1). The occurrence of the Sstim transition updates 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 (which means the corresponding codes in the IP core are ready for execution), the transition **Candidate** is enabled and the occurrence of this transition updates 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 into the IP core. The variable **ntask** represents the token value in the Ntask place. When the token value in this place becomes '1', the transition **selected** is enabled because task1 is loaded into the IP core for execution. The occurrence of this transition resets the value of the tokens in both the stim and wait places, and the transition **Decand** is enabled in sequence. The occurrence of the Decand transition resets the token value in the Rdy place to ‘0’, which means task1 is no longer a candidate for scheduling and the corresponding ready signal becomes invalid.

4.2.4 The Environmental Set Description

Similar to the top level model in Figure 4-1, environmental places/transitions are highlighted by a 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 influences 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 uses the random integer number generator **discrete** provided by CPN Tools to generate a random integer number from 1 to 5. The generated number indicates the ID number of the newly selected task. A guide [ntask<>ID1] (means ntask is not equal to ID1) is attached to the Schedule transition to make sure that the scheduling work (as well as the execution of tasks in the IP core) is carried out until task1 is chosen (after that the scheduling result does not influence the current model until the token value in the Rdy place becomes ‘1’ again).

The execution of the selected transition also generates two tokens: one in the **new** place and the other in the **new2** place. The cooperation of the place new with the transitions **env** and **env1** is used to simulate the stochastic generation of another event corresponding to task1 from the same channel. The description of these places/

transitions is shown in places/transitions with the same names in the top level model in Figure 4-1. CPN function **P1()** (as well as **P2()** in the expression of the arc 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 that the execution of task1 is complete so that the wait bit is set by the occurrence of the **Swait** transition.

4.2.5 Simulation and State Space

Because of the random token value arranged by functions P1() and P2(), either the transition Sstim or Swait can be enabled first (or they are concurrently enabled), which reflects the nondeterministic operations of the STEP. CPN simulation is used to verify the token flow in the current model and CPN state space report is given in Appendix V.

4.2.6 Conclusions and Further Discussion

In this section, a Matrix structure used in Butler processor was modelled in CPN for event handling and task storage. A pair of stim and wait bits is used to judge whether its corresponding task is ready for processing in the IP core. As soon as a task becomes a scheduling candidate, a corresponding ready signal becomes valid and this signal is used in other parts of the STEP for power management or task management.

Defined as a string colour, a DATA token has only an abstract meaning in the model. When it is defined as a BIT color in a lower level model, the ACM model [gorg08] can be integrated to the current model for hardware design.

4.3. The Design of a Power Manager in CPN

In this section, we specify the segment of the top level model which focuses on the realization of the A&F policy (Figure 4-7).

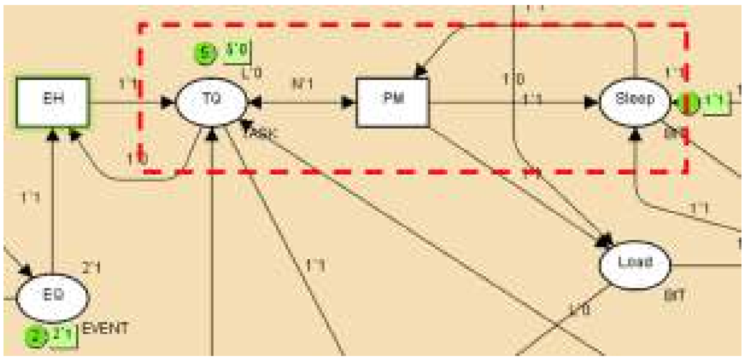


Figure 4-7: The Segment of the PM part in the Top Level Model

According to the previous section, a ready signal becomes valid as soon as the corresponding task becomes a candidate task. A simple A&F realization in the STEP is to count 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, there is no pattern to indicate how their corresponding ready signals become valid. The STEP must be alert to any change in ready signals so as not to miss any new ones in the accumulation. On the other hand, a valid ready signal is withdrawn by the reset in the stim & wait bits in the Event Handler part. Since the PM part in the STEP cannot disable any ready signals after accumulation counting, the PM needs to know which ready signal is new (not be counted in the accumulation) and which one is old (already counted in the accumulation) in the counting.

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.

4.3.1 The Model Description

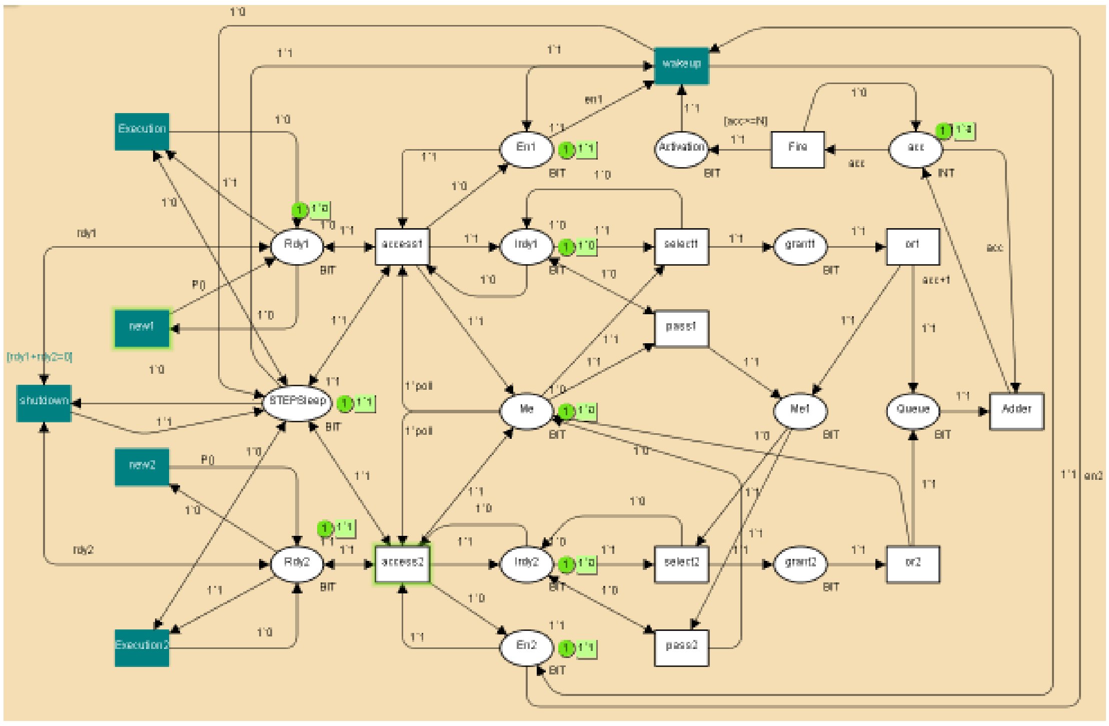


Figure 4-8: The CPN model of the PM

Figure 4-8 presents the CPN model of the PM part in a STEP where only two example tasks are considered. Token values '1' or '0' in the **Rdy1/Rdy2** places indicate whether the ready signal for task1 or task2 is valid or not. A BIT token in the **En1/En2** places is used to record whether the corresponding ready token has been used for accumulation. A '1' token in En1/En2 place means that the corresponding ready '1' token has not been used for accumulation, and therefore the latter token can enable the **access1/access2** transition.

The occurrence of the access1/access2 transition updates the token value in **Irdy1/Irdy2** place to ‘1’ respectively, indicating a new ready token is ready to be counted. The occurrence of the access1/access2 transition also toggles the token value in En1/En2 place to ‘0’ so that a ready ‘1’ token can only access to the current model once and duplicated counting is avoided in this model.

As demonstrated in the top level model, the task accumulation is needed only when the corresponding IP core is inactive. Therefore, transitions access1/access2 can be enabled only when the token value in the **STEPSleep** place is ‘1’.

When only one accumulation value is kept in the PM, all ready signals can be added to the accumulation value only in sequence. Therefore arbiters are indispensible in the current model. One easy solution is to build an arbitration array for all ready signals like Figure 4-9(a). If M is the number of tasks, this solution uses C_M^2 arbiters. Given a big number M, the number of arbiters and corresponding logic gates increase dramatically.

Another improved solution (Figure 4-9(b)) is inspired from the ring based arbiter introduced in the multi arbiter systems section in the book of [kinn07b]. In this case, a polling token circles 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 experience any collision during 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.

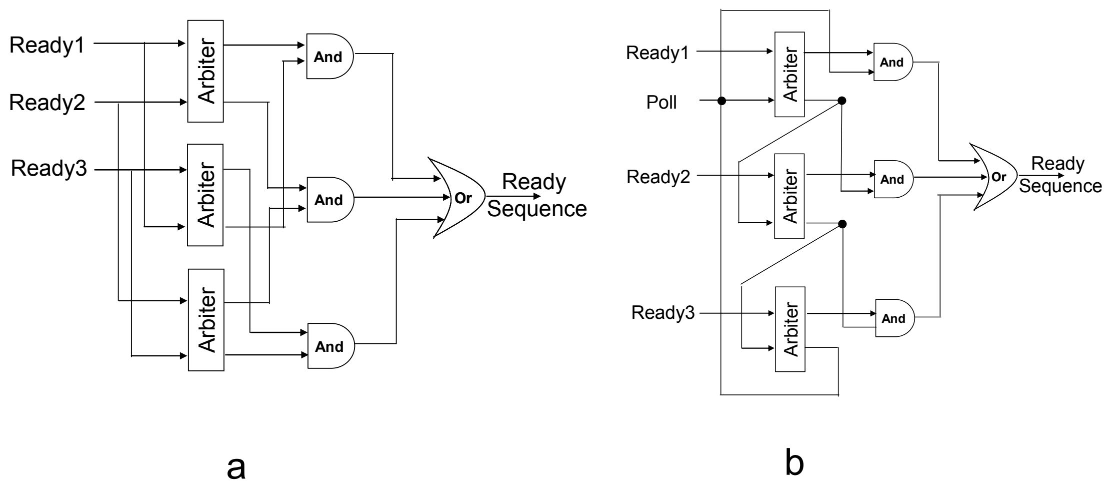


Figure 4-9: Two Arbitration Solutions in the PM Part

In the current CPN model, the polling token is held in the **Me/Me1** places and the variable **poll** is used to represent the flow of the polling token. 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 enables the transition select1. The occurrence of the transition first grants the ready token for accumulation, and then passes the polling token to the Me1 place. If the token value in Irdy1 is ‘0’, the transition pass1 is enabled accordingly and pass the polling token directly to the Me1 place. The occurrences of the pair **select2** and **pass2** transitions are carried out in a similar way and they return the polling token to the Me place. After adding all current valid ready signals to the accumulation, the polling accumulation ends after the occurrence of select2/pass2 transition for power saving, and it begins 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 tokens in the Me place between the access1/access2 transition and the select1/pass1 transition. When two (or more) tokens in the Rdy*i* (*i*=1,2) places become '1', their corresponding access*i* transitions are enabled. After one access*i* transition occurs, the polling token in the place Me becomes '1' and enables one of the pair select1/pass1 transitions. Therefore, both the other access*i* 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 access*i* transition is disabled until the end of one round of polling accumulation. This occurrence sequence reflects the situation when the polling token is first granted by the arbiter, and the valid ready signal is added to the accumulation result next time when the polling token arrives. Otherwise, if the other access*i* transition occurs first, the token polling increases 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 realizes the accumulation of several tasks.

The **or1/or2** transitions represent the OR gate in Figure 4-9(b), and the execution of one or1/or2 transition adds 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 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 increases the accumulation by 1. One guard [acc>=*N*] is attached to the **Fire** transition to make sure one token is added to the **Activation** place only when the token value in the acc place is no less than the accumulation limit

N (*N* is set to 2 in the current model). The occurrence of the Fire transition resets the token value in the acc place to ‘0’ to prepare for the next accumulation processing.

4.3.2 The Environmental Set Description

The environmental transition **Wakeup** represents the wakeup processing in the IP core, and its occurrence sets the token in the STEPSleep place to ‘0’ to disable all transitions in the current model. The occurrence of this transition also sets the tokens for 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.

On the left side of Figure 4-8, environmental transitions **Execution1** and **Execution2** represent the executions of task1 and task2 in the IP core respectively. These two transitions may be enabled concurrently, and the random occurrences of these transitions represent the different scheduling results generated by the STEP. The occurrence of each Execution transition resets 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*i* place are ‘0’ and its occurrence reflects the shutdown processing in the IP core when none of the tasks are ready for execution.

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 the function **P()** (which is used in the top level model) is used to make the generation of tokens in Rdy*i* place stochastically. These environmental transitions/places generate all possible combinations of input tokens to and consume output tokens from the current system.

The state space report in Appendix VI gives the correctness verification of the current model.

4.3.4 Conclusions and Further Discussion

In this section, a Power Manager CPN model was built where the A&F policy is used to give on-off control to the IP core. An enable token is used for every ready token to avoid duplicated counting. A polling accumulation, which is inspired by ring-based arbiters, is used in accumulation with a limited number of arbiters. Although only two tasks are involved in the current model, the model can be easily extended to more realistic cases when tens of tasks are embedded in an IP core.

In the current model, all tasks are thought to have the same priority since only task numbers are accumulated. Realistic tasks always have different priorities since they may have different deadline requirements. In this case, their priorities instead of task numbers are accumulated in the Power Manager. Therefore, a sleeping IP core can be activated by one high priority task or several low priority tasks in different situations. Although the priority based A&F policy increases the complexity of the PM circuits, it could enable the IP core to have better performance.

4.4. The Design of a Task Manager in CPN

When an IP core completes its wakeup processing, it tries to load a new task from the accumulated ones. In this section, we try to specify one segment (Figure 4-10) in the top level model which focuses on the scheduling execution.

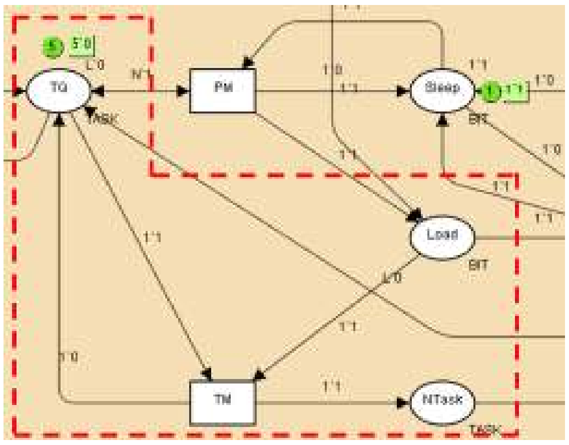


Figure 4-10: The Segment of TM in the Top Level Model

4.4.1 A Priority Based Round Robin Scheduling Policy

When more than one task is ready for execution, some scheduling policy is needed for task selection. In this section, a **Priority Based Round Robin** policy works as an example scheduling policy when we build the CPN model.

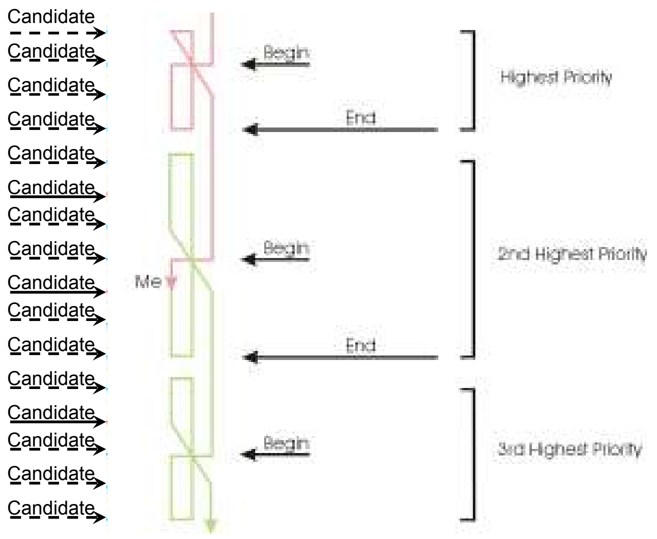


Figure 4-11: The Priority Based Round Robin Policy

Figure 4-11 is the figure used in the introduction of Priority Based Round Robin policy [camp04]. Arrows on the left of the Figure keep a list of all tasks in the IP core sorted by priority. A dotted arrow represents an invalid scheduling candidate (the corresponding task is not ready for execution) and a solid arrow indicates a valid candidate. Scheduling always starts from the highest priority group towards the lowest priority group, so as to give tasks with a higher priority more opportunity to be loaded in the IP core. For tasks in the same priority group, the scheduler uses 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 which has been loaded into the IP core most recently is marked as a **last** task. In Figure 4-11, the last task in every priority group is pointed by the Begin arrow. Polling scheduling starts from the next task to the last task in the highest priority group and ends when the first valid task is found. If no valid candidate can be found in this group, the scheduling point jumps to the last task in the second highest priority group to carry out a 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) is sent to the IP core.

4.4.2 The CPN Model of Scheduling

A CPN model (Figure 4-12) is built to show the scheduling execution of Priority Based Round Robin policy. In Figure 4-12, 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 because one '1' token is given to places **Last1** and **Last3** each while the 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 round robin scheduling in each group. A new scheduling is enabled by a '1' token in the Me place, and simulation results in

Figure 4-13 show the scheduling procedure (a dotted rectangle in each figure is used to indicate the enabled transition).

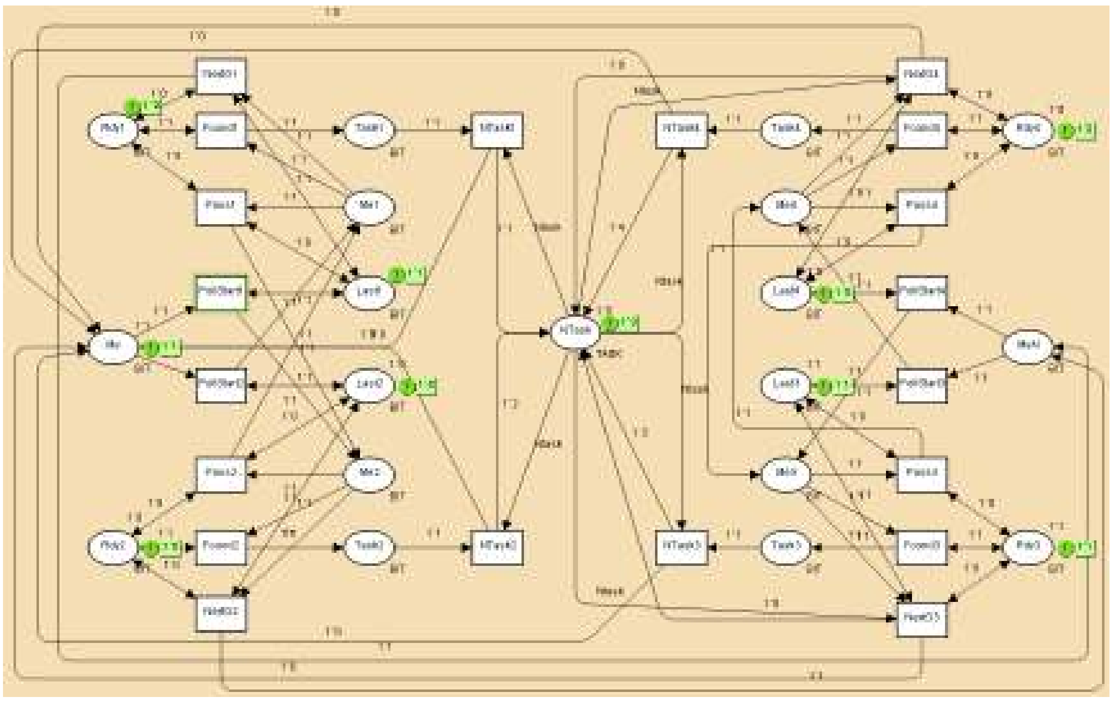


Figure 4-12: The CPN Model for Scheduling

The token in the place Me first enables 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 adds 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 4-13(a)) and the occurrence of this transition passes 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 can be found in this group. The scheduling moves on to the next priority group. Therefore, the **NextG1** instead of **pass1** transition is enabled by the token in the Me1 place (Figure 4-13(b)).

The occurrence of the NextG1 transition generates one token to the MeN place which enables transition **PollStart3** because task3 is the last task in the group (Figure 4-13(c)). After the occurrence of the PollStart3 transition, the polling token is moved to the Me4 place to check the status of task4. With one '0' token in the Rdy4 place, the transition **pass4** is enabled (Figure 4-13(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 4-13(e)).

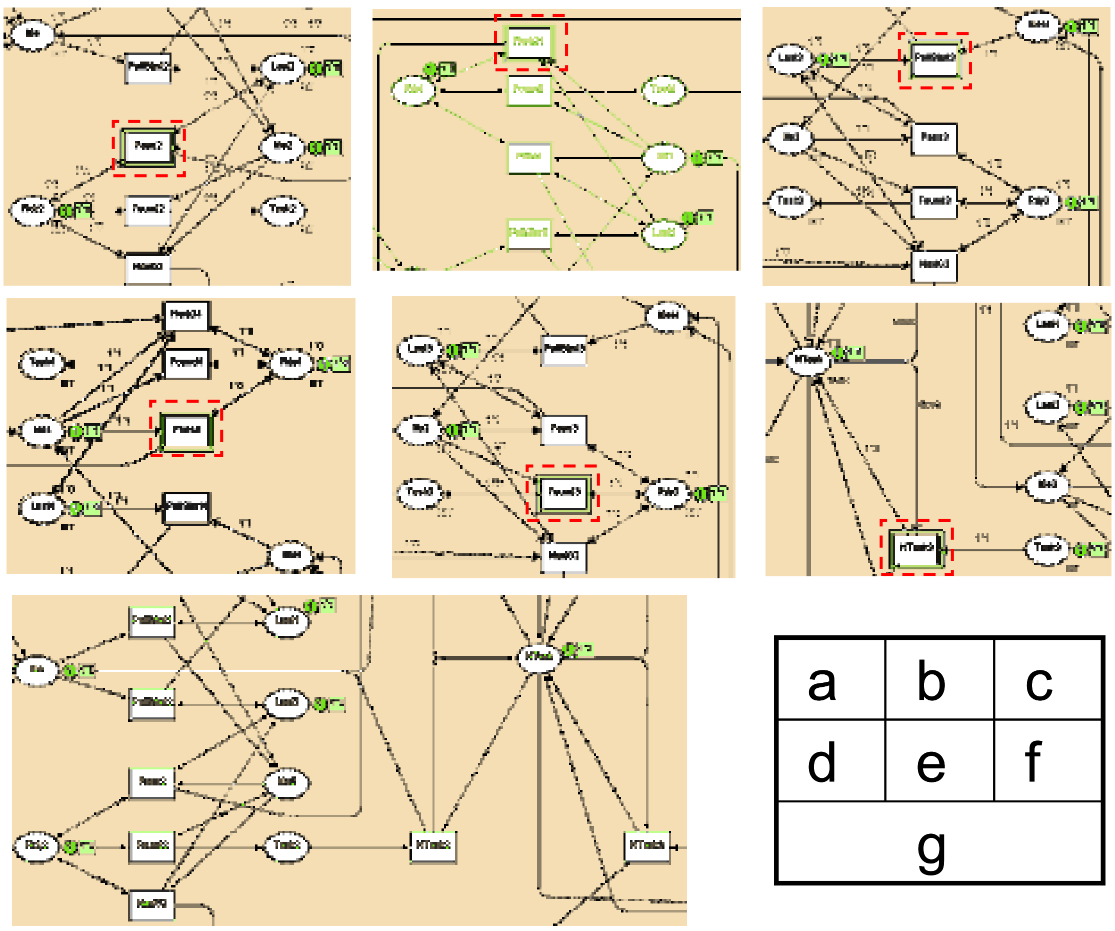


Figure 4-13: Simulation Steps in the scheduling

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 4-13(f)). Its occurrence updates 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 4-13(g)).

Given any combination of token values in the four Rdy*i* places, the scheduling flow is similar. Next, we 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 4-13(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 4-14) because no candidate task can be found in this priority group. Without any lower priority group available, it means no task is ready to be executed in the IP core. Therefore, the occurrence of NextG3 (or NextG4 when task4 is the last task in the group) resets the token value in NTask place by '0' to indicate the IP core that no more new tasks can be loaded.

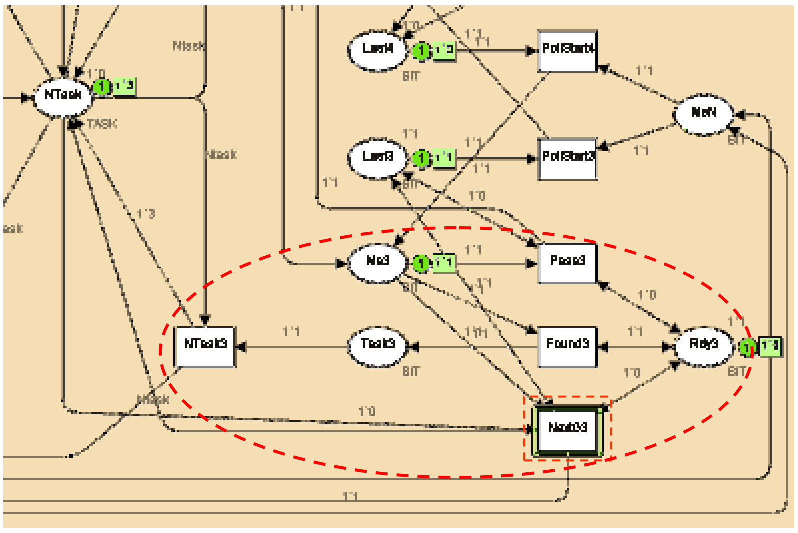


Figure 4-14: No Ready Tasks in the Scheduling

In Figure 4-14, transitions and places with the 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 4-14). 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.

4.4.3 The CPN Model of the Task Manager

Although the transitions and places in Figure 4-12 can successfully carry out a priority based round robin scheduling policy, more places and transitions are needed to guarantee the safety and correctness of scheduling. Figure 4-15 gives one example model of the Task Manager in the STEP and its test environment when only two tasks (and one priority group) is used in the scheduling.

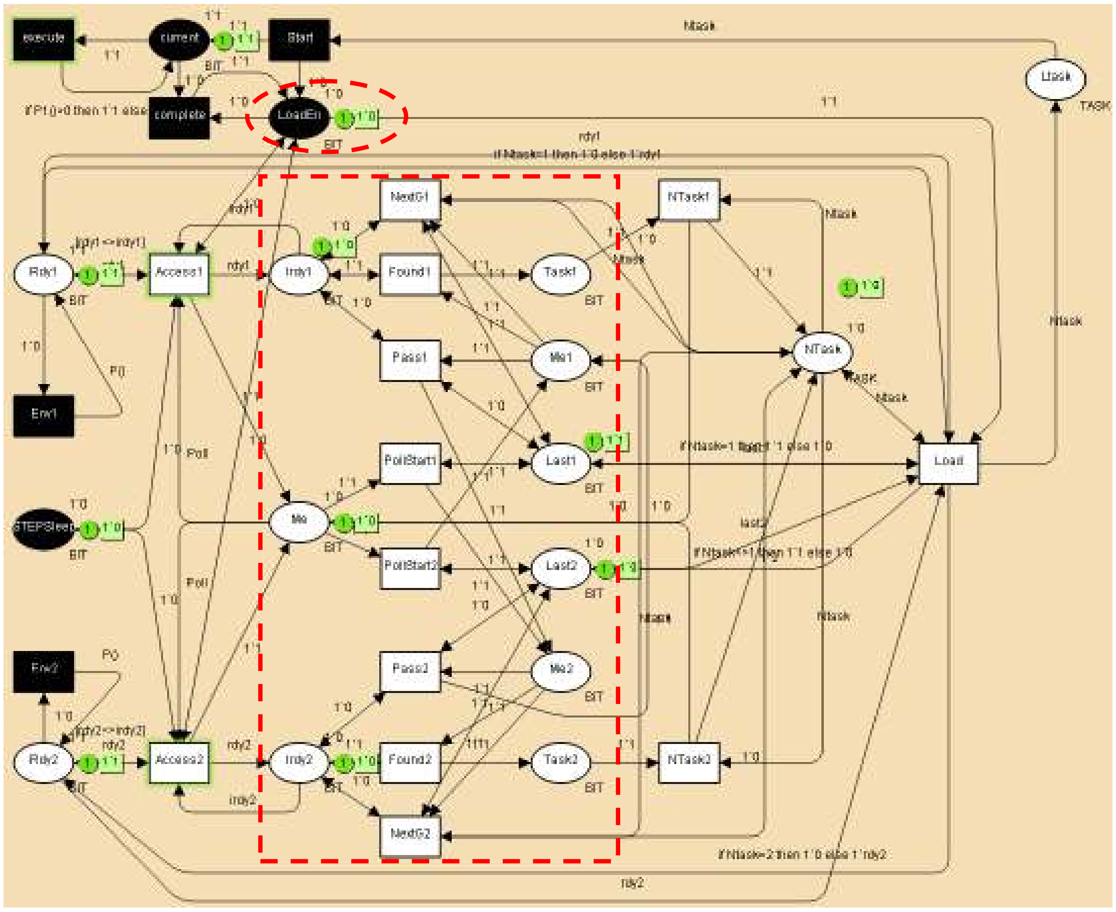


Figure 4-15: The Full CPN model for the Scheduling

The environmental place within the dotted circle is named **LoadEn** whose token ‘1’ represents the task loading a request from the IP core. A ‘1’ token in the LoadEn place enables 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 is 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 arrive at 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 and task loading execution are carried out simultaneously. Suppose the scheduling updates the token value in the Ntask place from ‘1’ to ‘2’. Whether task1 or task2 is loaded into the IP core depends on whether the scheduling transitions or the loading transition occurs first. This uncertainty confuses the IP core and may cause serious consequence. A safer design enables scheduling only when no load request is available. In Figure 4-15, 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. Another enabling precondition of transitions access1 and access2 is the existence of a ‘0’ token in the **STEPSleep** place, which means that scheduling is enabled only when the IP core is active.

The tokens held in places **Irdy1** and **Irdy2** indicate the status of the 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 respectively. With the guard [irdy1<>rdy1] and [irdy2<>rdy2] in the access1/access2 transitions, scheduling only begins when some changes happen to the ready signals. When at least one of these transitions occurs, the token value in the Me place is updated to ‘1’. The scheduling executions are modelled by places and transitions within the dotted rectangle which has been introduced in the last section in detail.

According to the model, when more than one ready token is toggled concurrently, some competition exists in the occurrence of the Access*i* transition and some scheduling transition within the dotted rectangle, since the occurrence of a scheduling transition consumes the token in the Me place so as to block any further occurrence of the access transition until the end of the scheduling. This behaviour of the model reflects the competition between the validation of a ready signal and the arrival of the round robin polling signal. However, given that no valid LoadEn signal is generated from the IP core, different occurrence sequences of these transitions do no influent the scheduling result.

The occurrence of the Load transition not only loads the ID number of the new task into the IP core, but also updates the status in the STEP. The task loading enables the EH to reset the wait & stim unit in the matrix. In Figure 4-15, the expression of the arc from the transition Load to the place Rdy1 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 is reset to '0'. Otherwise, the token value stays the same as before. Furthermore, any token reset in the Rdy1/Rdy2 place enables the Access1/Access2 transition when the IP core starts executing the new task, and the token value in the LoadEn place becomes '0'. Therefore, new scheduling is 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 also resets the last task in every priority group if it changes. If no task is found to be ready for execution, the occurrence of the Load transition resets the last task to its default status (for example, in the current model, task2 is the default last task in its group).

4.4.4 Environmental Set Description

When one token is added to the Ltask place, the environment transition **Start** is enabled. It indicates that the IP core starts the execution of the new task. Therefore one token '1' is given to the place **current,** which indicates that one task is processing. The occurrence of the Start transition gives one '0' token to the LoadEn place, which means the task loading procedure is completed. The function P1() (which is the same as the P1() function in the EH unit model in Figure 4-6) 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. Therefore the token value in the LoadEn place is updated to '1', and a new task is loaded afterwards. If the taken 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. 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 a new event which in turn validates the corresponding ready signals again.

All these environmental transitions/places generate all possible combinations of input tokens to and consume output tokens from the current system. The state space report in Appendix VII gives the correctness proof of the current model.

4.5. The Design of an Output Control and Interface in CPN

In this section, we try to specify the segment in the top level model which takes charge of the wakeup/shutdown procedure and the output event control (Figure 4-16).

As introduced in Chapter 2.1.1, both shutdown and wakeup processing of an IP core have cost in time and power. It is possible that some changes are happening in the environment during the shutdown or wakeup transitions. For example, some new events may arrive at the current VSB during the shutdown processing. If they are handled by the EH, their corresponding tasks become ready and generate an activation token in the PM part according to Figure 4-8.

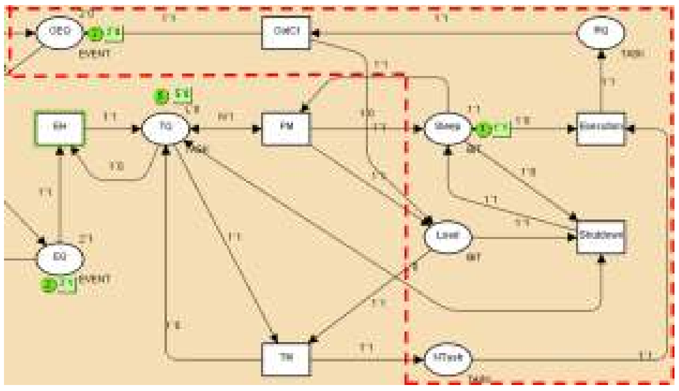


Figure 4-16: The Output Control Segment in the Top Level Model

This token may confuse the IP core whether to continue the shutdown processing, or abandon it to carry out the wakeup processing instead. One important job of the interface part is to make sure that both shutdown and wakeup transitions in the IP core can be carried out without being disturbed by the environment.

On the other hand, the Interface part can improve the entire VSB’s performance by applying parallel processing in the STEP when the IP core is waking up or shutting down. As introduced in Section 4.3 as well as in Section 4.4, the PM part is shut down when the IP core is active and the TM part is shut down when the IP core is inactive. In this section, we focus on whether they should be active during the wakeup and/or shutdown processing. In our design, the PM part is activated as soon as the shutdown transition starts (given its result does not disturb the execution of the shutdown processing in the IP core). Therefore, events that arrive during the shutdown transition are accumulated without delay. Similarly, the TM part is activated as soon as the wakeup transition starts, so that task scheduling can be completed before the IP core is ready for task processing. When these improvements are considered, the PM as well as the TM part cannot use the Sleep token provided by the IP core to control its execution. It is because the token in the Sleep token only toggles when the wakeup/shutdown transition completes rather than starts. Therefore, a new token is required to be generated from the interface to indicate the beginning of the corresponding transitions.

As indicated in the top level model, the output controller generates 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 becomes 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.

4.5.1 The Model Description

Figure 4-17 presents the CPN model of the Interface and Output Control part in the STEP where only two example tasks 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 4.2.

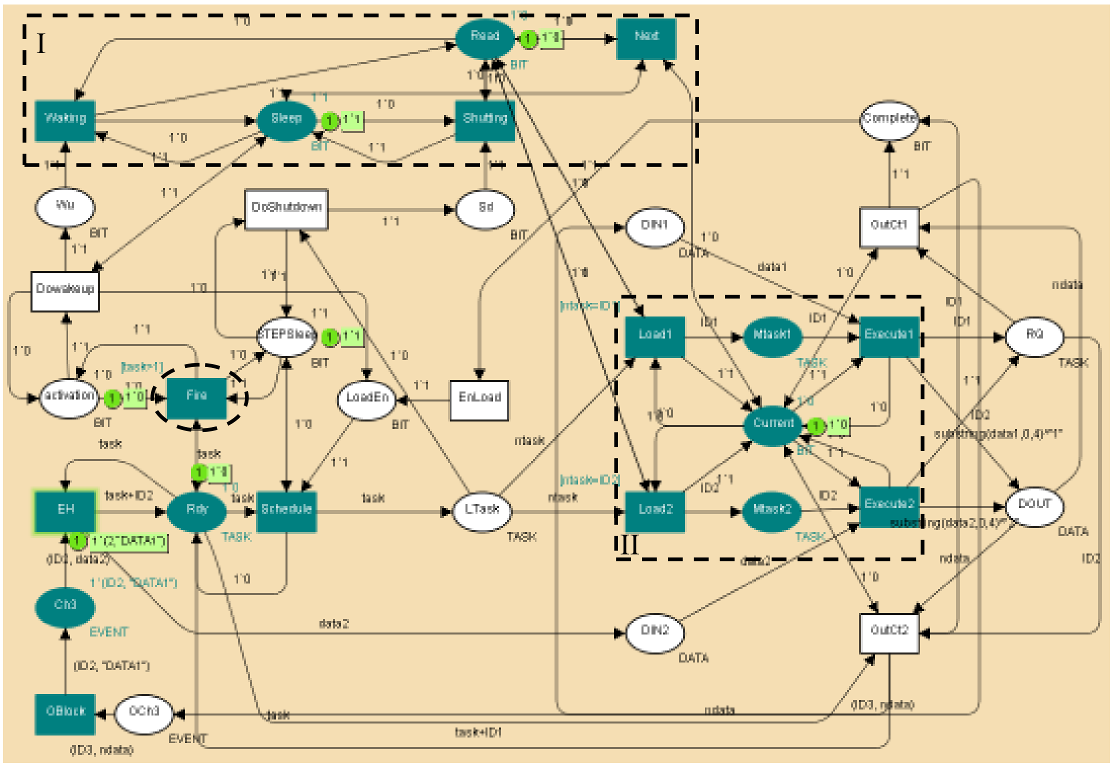


Figure 4-17: The CPN Model of Output Control and Interface

The **Fire** transition highlighted by the dotted ellipse has been introduced in the PM part in Section 4.3 (the accumulation limit is set to 1 to simplify the current model). The occurrence of this transition gives one '1' token to the **Activation** place. At the same time, it also updates the token value in the **STEPSleep** place to '0'. The token in this place indicates the beginning of wakeup processing when its value is '0' or shutdown processing when its value is '1'. Therefore, the token in this place is used in the PM model in Figure 4-8. All transitions in the PM part are disabled immediately without waiting for the completion of the wakeup processing in the IP core. Similarly, the token in this place is used in the TM model in Figure 4-15 to enable task scheduling to be carried out in parallel with the waking up of the IP core.

The availability of a '1' token in the activation place enables the **DoWakeup** transition and the latter's occurrence gives a '1' token to the **Wu** place. The enabling of the DoWakeup transition also depends on the token value in the Sleep place. If the token in the Sleep place is '0' (which means the IP core is active), the DoWakeup transition is not enabled even when the token in the active place is '1'. It means no wakeup command is issued by the STEP when the IP core is active or shutting down, even when enough tasks have been accumulated in the PM part. This design allows an IP core to complete its mode switching transitions without being disturbed by the environment.

The occurrence of the Dowakeup transition also generates one token to the **LoadEn** place so that the scheduling result can be loaded to the **LTask** place. The states and transitions within the dotted rectangle I represent the operations in an IP core for wakeup, shutdown and task loading. For example, the **Waking** transition in the upper left edge of Figure 4-17 is enabled when a '1' token is available in both **Wu** and **Sleep** places. Its occurrence reflects the completion of the wakeup transition in the IP core and it toggles the token in the Sleep place to '0'. A '1' token is also added to the **Read** place, which indicates that the IP core sends a Read signal to its STEP to load the ID number of a new task from the LTask place.

Transitions and states within the dotted rectangle II represent the task executions in the IP core. Once the task ID number is loaded into the IP core, the IP core uses the number to search for the corresponding codes segment. If task1 is loaded into the IP core, transition **Load1** is enabled since one guard [ntask=ID1] is attached to this transition.

The occurrence of this transition turns the token value of the **Current** place to ‘1’, which is used to indicate that task execution starts. The availability of tokens in **Mtask1** and **DIN1** places represents both the codes and the data are ready so that the transition **Execute1** is enabled. The occurrence of the Execute1 transition reflects the completion of task1’s execution in the IP core. Both the result data and ID number of the current task are saved afterwards. Therefore, one ‘1’ token is added to the **RQ** place, and one DATA token is added to the **DOUT** place. The expression in the arc from the Execute2 transition to the DOUT place is written as ---- substring(data1,0,4)^ “2”, which replaces the last character of the input DATA token to “2”, which is the ID number of task2. For example, if the input DATA token is “DATA1”, the output DATA token in the DOUT place is “DATA2”. This arc expression is used to simulate the data processing in the IP core. The occurrence of the Execute1 transition also resets the token value in the Current place as the symbol of task completion. The execution of task2 in the IP core is described by transitions of **Load2** and **Execute2**, whose enabling and occurrence are similar to their counterparts introduced before.

The token toggle in the Current place activates the output control processing in the STEP. One basic job of the output control is task routing, which means to find which task is supposed to use the result data and in which VSB the task (called **target task** later) is located. If the target task is not in the current VSB, the current VSB needs to use the ACM method to transfer the data to the new VSB and load the ID number of the target task into the communication channel to the new VSB. When the target task is in the current VSB, data transfer is either avoided if the IP core can just update the start address of the input data for the target task by that of the result data, or it can be easily carried out within the same clock domain. In order to represent both cases, in the model given in Figure 4-17, the target task for task1 is supposed to be task3, which is located in another VSB and that for task2 is task1 which is in the same VSB.

Now we can continue the description of the CPN model. If the token value in the RQ place is '1' (which means the completed task is task1), the transition **OutCt1** is enabled as soon as the token in the Current place becomes '0'. The occurrence of this transition adds a TASK '3' token (ID3=3) as well as a "DATA1" DATA token to the **OCh3** place, which indicates the signal and data transfer taken place in the 3rd Output Channel of the STEP. The occurrence of the environmental transition **OBlock** is used to reflect the execution of task3 in another VSB which in turn generate an event to the current VSB in the 3rd Input Channel. Therefore, a new EVENT token is available in the **Ch3** place.

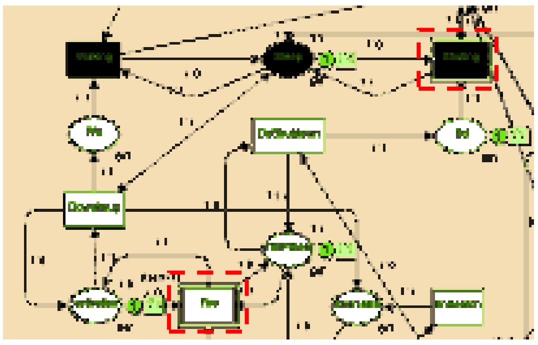
If the token value in the RQ place is '2' (which means the execution of task2 is just complete), the transition **OutCt2** is enabled and its occurrence moves the DATA token in the DOUT place to the DIN1 place, which means the result data of task2 is just ready for the execution of task1 (as the target task of task2). The token value in the **Rdy** place is increased by 1, which indicates task1 is ready for scheduling. Compared with the output control token flow for task1, the output control for task2 does not involve the occurrence of both OBlock and **EH** transitions. It means that one target task can be quickly ready for execution if it is located in the same VSB as the task who generated the input data.

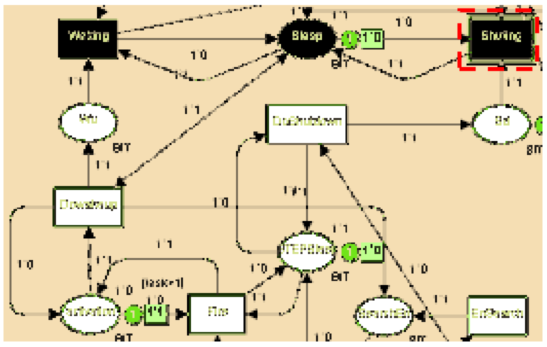
The occurrence of OutCt1/OutCt2 transition adds one token to the **Complete** place, which enables the **LoadEn** transition in sequence to load a new task. The token toggle to ‘0’ in the Current place also enables the **Next** transition, whose occurrence issues a Read token to load the task to the IP core.

If the value of the new token in the LTask place is ‘0’, it enables the **DoShutdown** transition and the occurrence of this transition turns the token value in the STEPSleep place to ‘1’ which means the transitions in the PM part is enabled (and the transitions in the TM part is disabled) afterwards. The occurrence of the DoShutdown transition also give one token to the **Sd** place, and the token in this place enables the **Shutting** transition whose occurrence represents the completion of the shut down processing in the IP core. A new wakeup command can be issued from the STEP from then on.

4.5.2 Simulation and State space

All concurrent execution between the STEP and its IP core can be seen by simulation. For example in Figure 4-18, both Shutting and Fire transitions are enabled (All enabled transitions are highlighted by dotted rectangles) which indicates the concurrent processing between the PM part in the STEP and the IP core. In order not to disturb the IP core, different occurrences of the two transitions should achieve the same result. If the shutting transition occurs first, the token in the sleep place becomes ‘1’. The occurrence of the Fire transition afterwards gives one ‘1’ token to the activation place which enables the Dowakeup transition. On the other hand, if the Fire transition occurs first (Figure 4-18(b)), the ‘1’ token in the activation place cannot enable the Dowakeup transition because the token in the sleep place is still ‘0’. The latter is only enabled after the occurrence of the shutting transition. The same result achieved from different occurrences indicates the Interface can protect the IP core from environmental interruptions when a mode switching is carried on.





a b

Figure 4-18: Concurrent Operations in a VSB

More verification can be achieved from the state space report in Appendix VIII.

4.5.3 Conclusions and Further Discussion

In most cases, the function performance in a portable device is decomposed into executions of several tasks in different computation components. The Output Control part of the STEP is used to connect executions of these tasks in different VSBs to fulfil the performed function. The CPN model in this section describes the structure of the Output Control (as well as the Interface) part and highlights the different event transfer when the target task has different locations.

The CPN model in this section is still sketchy since this part of STEP is highly implementation oriented. For example, a more delicate route-map is needed to easily get the location and ID number of the target task when tens or more tasks are embedded in an IP core. When task priority instead of task number is used in the accumulation in the PM part, the Output Control part also transfers the priority of the target task to the new VSB (if the target task is not in the current VSB). All this information should be implied to a lower level CPN model when the implementation is specified.

4.6. Conclusions and Future Work

In this chapter, a top level VSB model was built in CPN, which highlighted concurrent processing among different parts of a VSB. Four second level CPN models were also presented, each of which focused on one key part of the STEP. These models are used to specify executions in the STEP and avoid unnecessary concurrency in the VSB which may do harm to the system performance. Simulation and State space tools provided by CPN Tools have been used to prove the correctness of these models.

If a synthesis method is considered, we may continue the design of CPN models at the lower levels until BIT is the only colour of all places, and CPN models finally become their counterparts in normal PN. One problem of this method is the state explosion, since tens of BIT tokens may be needed to represent one EVENT or TASK token shown in the top level model.

Furthermore, all CPN models given in this section involve only the minimum number of tasks. Although this can make the model more clear for demonstration purpose, it cannot be implemented directly on a real system since each IP core may involve tens or more tasks in most cases. When these tasks are modelled in the CPN model of a real system, the number of states may be out of the power of CPN state space calculation.

Therefore, a state space check for a BIT token based CPN model which can directly reflect the circuit design is not a practical solution. Instead, we may use the high-level CPN models about different parts of the STEP to guide the design of these parts and use simulation to partially prove the correctness of our model.

Chapter 5

The Construction of SOCs with VSBs in MATLAB

With CPN models given in Chapter 4, all important concurrency properties of a Virtual Self-timed Block have been demonstrated. However, the modelling work is not carried out to the lowest level because the lowest level model depends on the detail about the IP core that cooperates with the STEP in a VSB. On the other hand, the power efficiency of a VSB, which is the main concern of system performance, is not being analyzed in CPN models in Chapter 4 since the power property relies on the implementation of a VSB for real systems.

In this chapter, an example implementation of a VSB is designed where some tasks embedded in VSBs are designed to carry out data processing in an SOC content. This example is used not only to demonstrate the cooperation of VSBs in an SOC, but also provides a test bench for the power analysis of a VSB.

In Chapter 3, the A&F policy has been proven to have great potential to trade latency for power. However, the analysis was based on Markov processes assumption. No proof has so far been given of whether this policy can still be efficient when the

Markov assumption is weakened or not even satisfied. Actually, few users know in advance whether their systems follow Markov process, let alone the necessary parameters for the power/latency calculation used in Chapter 3. Therefore, the test bench designed in this chapter has great usage for the efficiency verification of the A&F policy in real implementations.

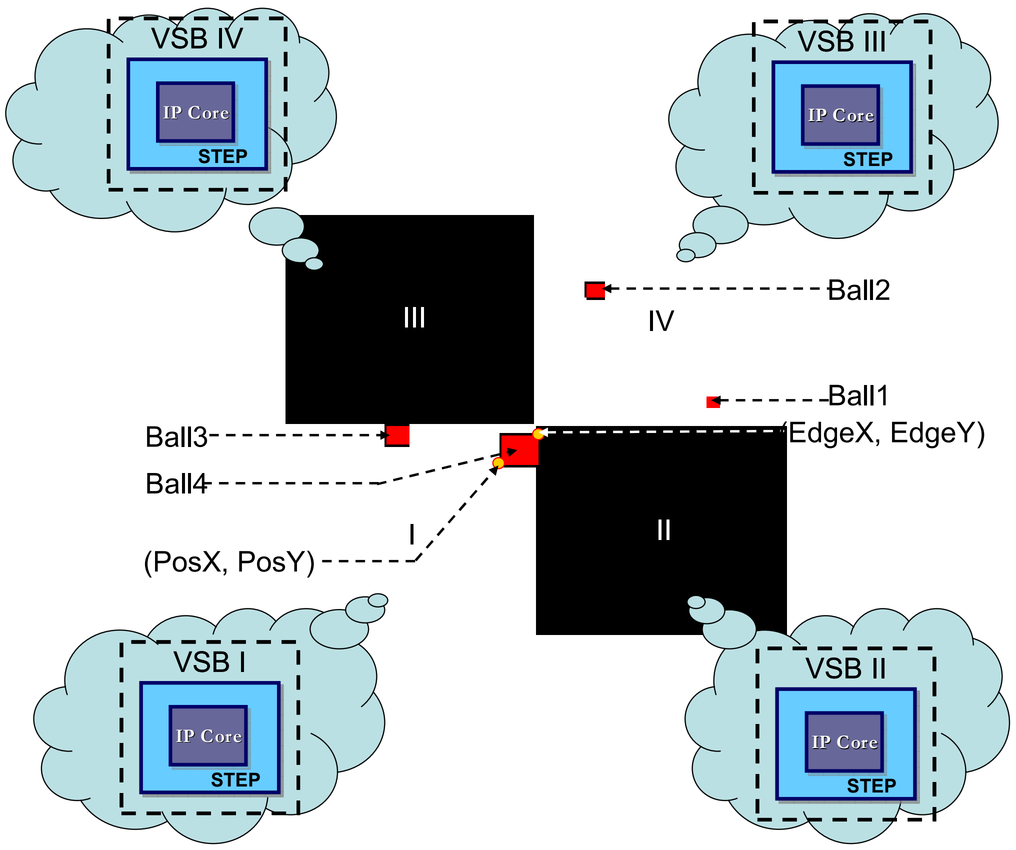


Figure 5-1: The Implementation of Ball Game

The example implementation used in this chapter is called **Ball Game** (Figure 5-1). Four balls of different size move in a playground with different speeds but identical mode. The entire playground has been evenly divided into four parts, called playground I, II, III, IV respectively. Four VSBs are employed, and each VSB is used to control the ball movement in one playground. Four tasks are in the IP core of every VSB whose codes provide the movement control of the corresponding balls. Different codes may be used in the four tasks to provide random or history based movement, but they all need to avoid ball collision (two balls are overlapping) in the movement.

When some ball moves across the border between two playgrounds, an event is generated to hand over the control of the ball to another VSB and the parameters of the ball is transferred to the VSB by the way of ACM.

If no balls contain in one playground (like playground III in Figure 5-1), the IP core corresponds to the playground is shut down to save power and the playground is patched in black colour accordingly. When and how to activate the IP core for task processing depends on the DPM policy implemented in the STEP of each VSB.

We prefer to use MATLAB Simulink rather than CPN Tools to design models for the example implementation of ball game. CPN Tools is good for presenting concurrent executions, but has limited power to present the execution sequence varied with time/sample elapsing. However, the execution property in the time dimension, which shows the probability distribution among different operation modes of an IP core, is of vital importance for the power analysis. On the other hand, MATLAB Simulink provides powerful observation of signal variation in time dimension.

Secondly, the state space checking provided by CPN Tools can only deal with limited number of states. When four VSBs with realistic IP cores are used, the total number of states may be too big to be calculated in state space checking. On the other hand, with the checking of state space about one VSB in chapter 4, any further state space checking with multiple identical VSBs is not needed. Even if some errors happen in the connection among VSBs, they can be easily found by the simulation provided by MATLAB Simulink.

Finally, MATLAB Simulink can provide visual observation of the four VSBs' execution by showing the ball movement on the screen while CPN Tools cannot do this.

Therefore, MATLAB Simulink is chosen to realize the example implementation of a ball game in this chapter. The implementation detail is as follows: A 100*100 pixels area is used as the entire playground of the ball game. Therefore each VSB controls a 50*50 pixels area. Four squares with different sizes represent the four balls in the game, and five parameters are used to describe one ball's movement. **PosX** and **PosY** are the positions of the bottom left edge of the ball on the X and Y axis respectively. **Width** represents the size of the ball and **Speed** indicates how fast the ball moves with each step. **History** remembers the direction of the ball's last movement. Four numbers (0,1,2,3) are used for the History information, which represent moving left, right, up, and down respectively. Table 5-1 gives the initial parameters of all balls in our example system.

Table 5-1: Initial Parameters of Four Balls

	PosX	PosY	Width	Speed	History
Ball1	84	54	4	4	1
Ball2	60	80	6	6	2
Ball3	20	45	8	8	3
Ball4	43	40	10	10	2

The position of the upper right edge of a ball is used to calculate if it is crossing the border of one playground. For example in Figure 5-1, Ball4 is just crossing from playground I to playground II, and VSB II takes control of this ball accordingly.

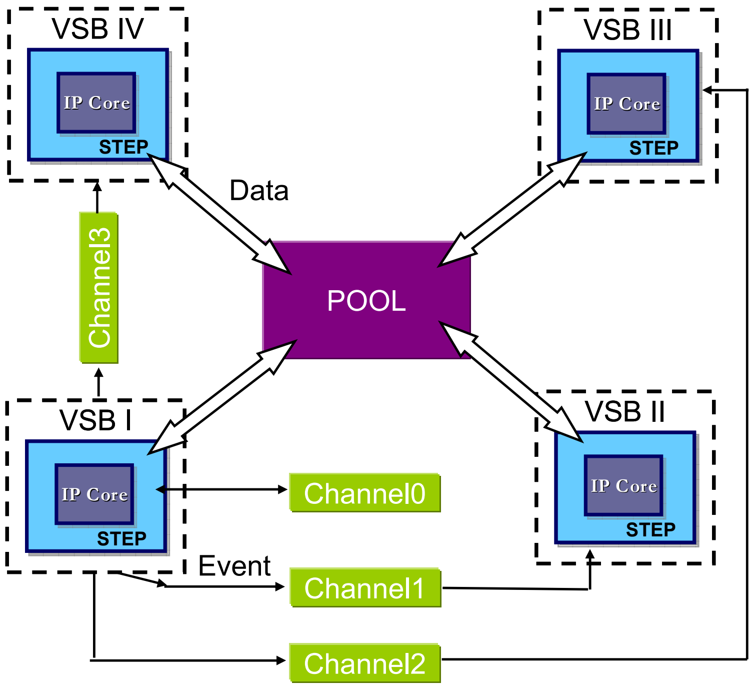


Figure 5-2: Data and Event Communication in the ball game

In the current version of the ball game, constant values are given to the width and speed of each ball and kept in each task program in IP cores. The other three parameters, PosX, PosY and History, are updated from time to time. When one task in a processor tries to decide the next position of its corresponding ball, it needs to consult all balls' positions no matter whether they are controlled by the same processor or not. Therefore a public POOL type ACM [xia02] (Figure 5-2) is used to save updated position of each ball for the consultation of possible all processors in the SOC. On the other hand, four Channel type ACM [xia02] contain in each VSB for the event communication with other VSBs. For each VSB, Channel0 is reserved for events sent to the same processor, while the other three Channels are used for event communication between processors in different VSBs (Figure 5-2 only describes Channels for VSB I, Channels for other VSBs can be built in the similar way).

According to the introduction of MATLAB Simulink in Section 2.3, a Simulink model can be built either with the construction of basic components provided by the Simulink Library, or by writing S-function codes. We construct the STEP of a VSB by basic components provided by the Simulink Library while using S-function to write task codes embedded in IP cores. This is because hierarchical design of the STEP from basic components can provide more observable signals to show the properties of different parts of a STEP, and it is more valuable for the real VLSI design guidance. On the other hand, an IP core design is not part of our research, we care only about functions rather than the circuit detail of an IP core.

Figure 5-3: The Design of a Virtual Self-timed Block in MATLAB

Since all VSBs in the implementation are identical, Figure 5-3 gives only the architecture of VSB I which controls the ball movement in playground I in MATLAB Simulink. The five subsystem blocks, PM, EH, TM, Interface and Output Control, are the five basic parts of a STEP. The IPCore block contains four tasks, each of which controls the moving of one ball. The task programs are written as S-Functions so as to integrate with other blocks to give a unified simulation result. In order to reduce wires and connections in the Simulink model, multiple input/output ports are used and the number contained with the braces [] indicates the wire indexes integrated by the port. For example in Figure 5-3, the first input port Ch1[1:4] represents the four input signals from input Channel 1 and the fourth input port DataIn[1:24] represents 24 data input signals.

In the following sections, the structure of the five parts of a STEP as well as the flow charts of the task embedded in the corresponding IP core part will be introduced in detail. These MATLAB models can be seen as the implementation of the CPN models in Chapter 4 and the simulation results in time/sample dimension of different signals in each part will be displayed accordingly.

5.1. The Design of Event Handler Part in MATLAB

As four VSBs are used in the system and each VSB's IP core contains four tasks, the Event Handler Part of each VSB is built by 4*4 Wait&Stim nodes altogether (Figure 5-4). The first input port Chs[1:4][1:4] is a multiplexed input port, which indicates there are four input Event Channels, and each Channel uses one-hot coding to indicate the ID number of the driven task. The first three Channels are used to connect with the other three VSBs and the last Channel is used as the feedback channel to receive events generated from the same VSB. Therefore, the event signals coming from the Chs[1:4][1:4] port are decomposed into 16 stim signals to set their corresponding stim bit in Nodes 1 to 16. Signals from Wait[1:4] and Reset[1:4] ports set the wait bit or reset all Wait&Stim nodes in one column respectively. The operation in each node is the realization of the CPN model in Figure 4-6. The subsystem block 4OR models the logic OR gate with four inputs, and signals from the output port Ready[1:4] indicate which task is ready for processing.

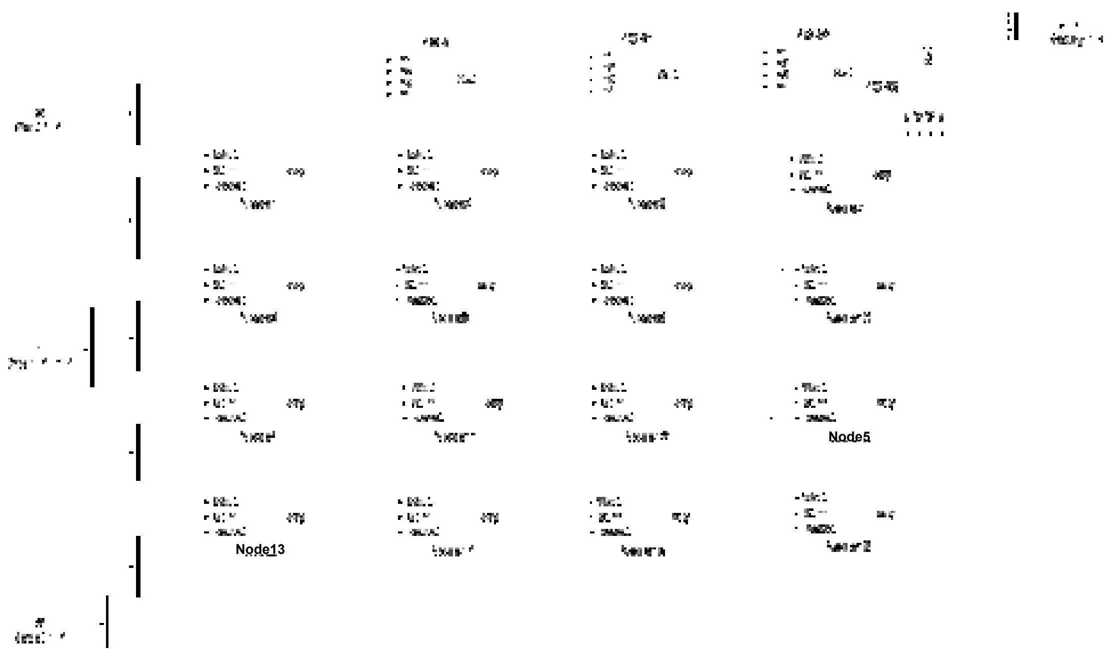


Figure 5-4: The Design of EH Part in MATLAB

According to the initial positions of the four balls indicated by Figure 5-1, both Ball3 and Ball4 are in playground I. Therefore the IP core in VSB I is activated and task3 and task4 are executed to control their corresponding balls to move one step further. Based on their initial history parameters, Ball3 moves one step up and Ball4 moves one step right. They are out of playground I afterwards. The IP core in VSB I is shut down accordingly. Although short, the period presents all executions that may be carried in one VSB. Therefore we use simulation provided by the MATLAB Simulink to observe the signal variation in different parts during this period. The simulation result for the EH part is given in Figure 5-5.

In Figure 5-5, Wait16 indicates the status of the wait bit of Node 16. Since no task is processing in the IP core in the initialize stage, all wait signals keep high (1). The initial setting generates two events in the feedback Channel to activate task3 as well as task4. Therefore, Stim16, as the stim bit for Node 16, becomes valid. When both the stim and wait bits of Node 16 are set, Ready4 (the 4th output signal in Ready[1:4]) becomes valid accordingly (2). After the IP core is fired (which is controlled by the A&F part that will be introduced in Section 5.2) and task4 is chosen to be loaded into the IP core (which is controlled by the Scheduler part that will be introduced in Section 5.3), Reset4, as the 4th signal from input port Reset[1:4] becomes valid. This signal clears both stim and wait bits in Node 16 (3) so that Ready4 becomes invalid accordingly (4). After the execution of task4 in the IP core, Wait4 becomes valid again (5) indicating the corresponding task codes can be loaded into the IP core afterwards.

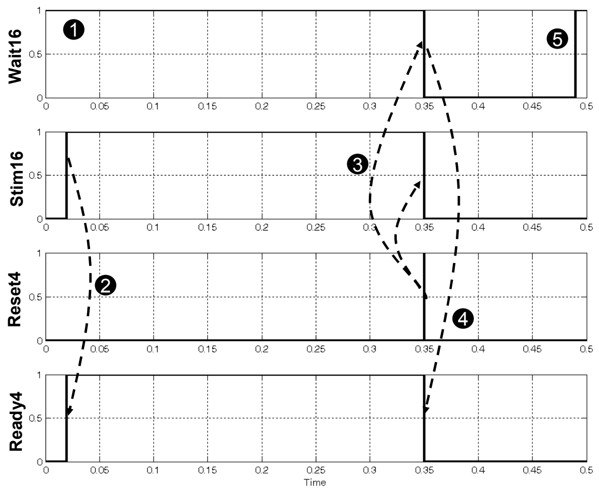


Figure 5-5: Simulation Result of the EH

5.2. The Design of PM Part in MATLAB

Figure 5-6 shows the design of the PM Part in a VSB which is the realization of the CPN model of the PM in Section 4.3. Each **access** subsystem block is the realization of the corresponding access transition in the CPN model of Figure 4-8. Each **Polling** block is the realization of the corresponding select/pass transitions in Figure 4-8.

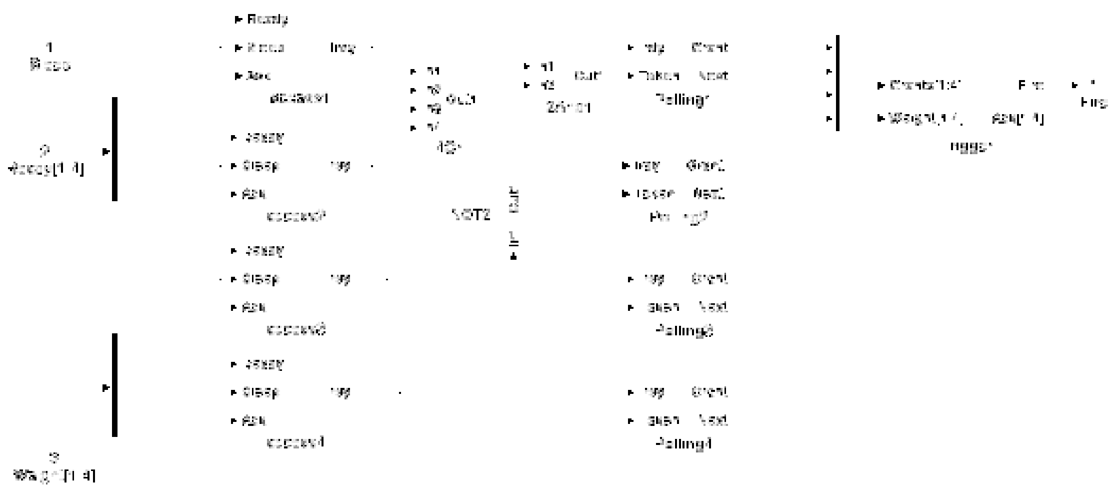


Figure 5-6: The Design of PM in MATLAB

The Trigger block in Figure 5-6 is based on the adder and Fire transitions in Figure 4-8. Moreover, the weight of each task rather than the number of tasks is accumulated in this block according to the suggestion given by the future work in Section 4.3. Currently, fixed weight is given to each task and the realization of the trigger block is given in Figure 5-7.

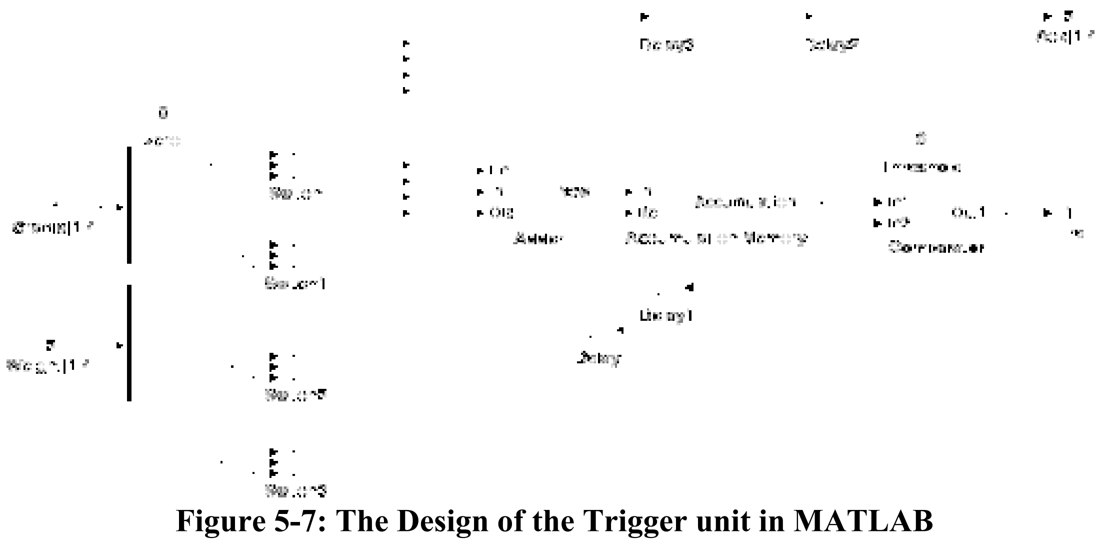


Figure 5-7: The Design of the Trigger unit in MATLAB

If any grant signal from Grants[1:4] becomes valid, the corresponding weight number from Weight[1:4] is added to the accumulation data which is kept in the Accumulation Memory block. Initially, weight number 5 and 3 are given to task3 and task4 respectively. Given Threshold 6, the accumulation of these two tasks can activate the IP core. A fire signal is sent through the output port Fire accordingly.

The simulation result of the PM part is given in Figure 5-8. The validation of Ready3 (as the 3rd signal of Ready[1:4]) disables Enable3 (as the enable signal for Ready3) so that the corresponding weight can only be added to the accumulation once(1). Irdy3, as the output signal of access3 in Figure 5-6, becomes valid accordingly (2). When at least one valid ready signal is captured, the polling accumulation, which was introduced in Section 4.3, begins. When Token3, as the 2nd input signal for Polling3 block, becomes valid, it indicates that the polling token arrives to check if Rdy3 is valid (3).

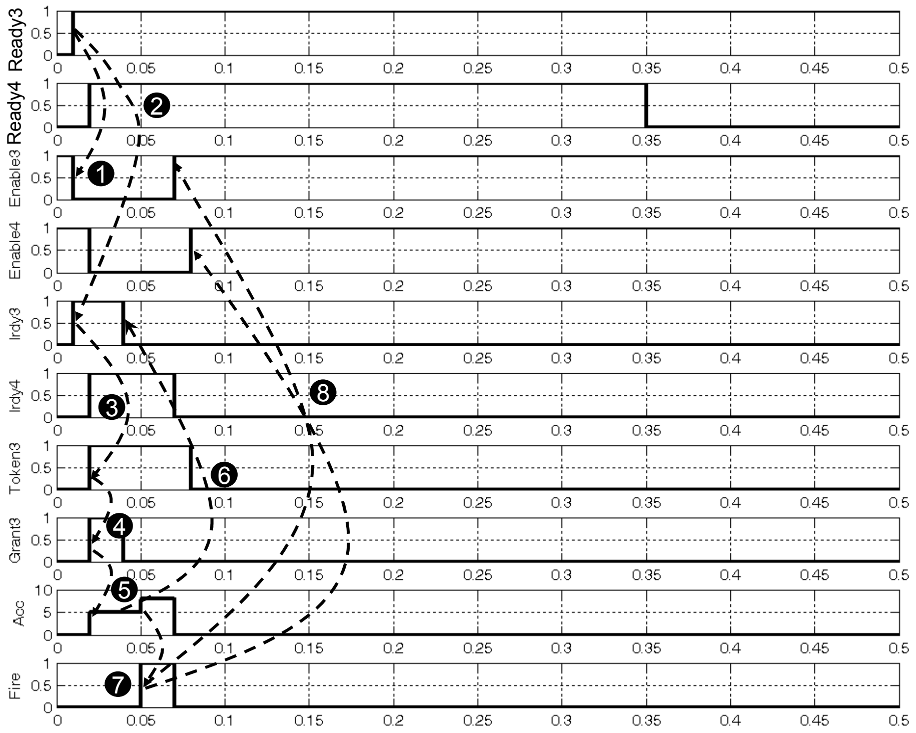


Figure 5-8: The Simulation Result of the PM Part

Since Ready3 becomes valid earlier than Token3, the former signal is granted. Therefore Grant3, as the 2nd output signal of Polling3 block, becomes valid (4). The grant signal enables the corresponding weight for accumulation and Acc, as the accumulation result, increases to 5 (5). After the accumulation in Acc, Irdy3 is withdrawn (6) and so is Grant3. The weight of task4 can be added to the Acc afterwards in the similar executions. When the Acc value becomes 8, the Fire signal becomes valid accordingly (7). When the sleep signal becomes 0, both Enable3 and Enable4 becomes valid again to cooperate with further incoming valid ready signals (8).

5.3. The Design of TM Part in MATLAB

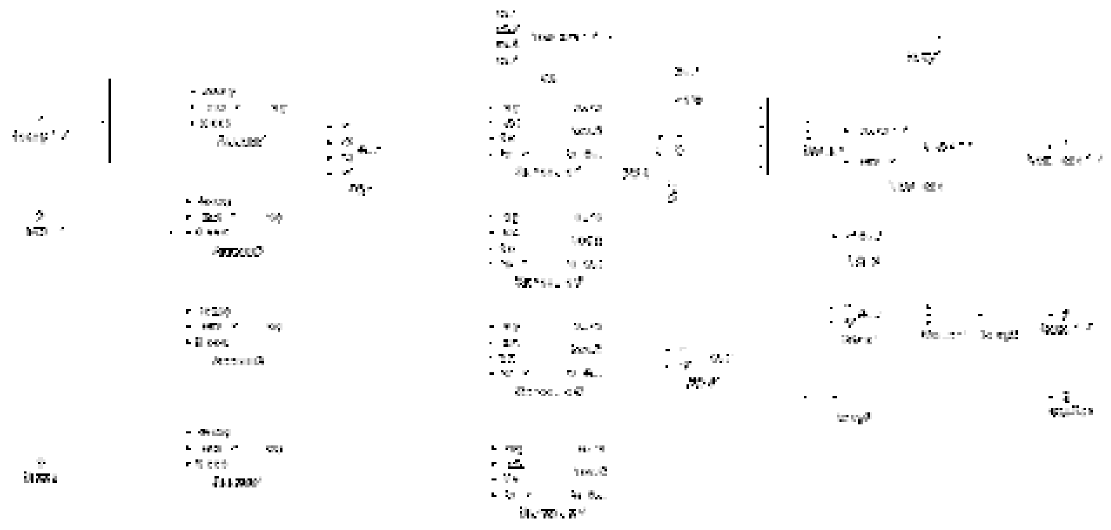


Figure 5-9: The Design of Scheduler in MATLAB

Figure 5-9 introduces the design of the TM part in MATLAB, which is the realization of the CPN model of TM in Section 4.4. Each **Access** subsystem block is the realization of the corresponding Access transition in the CPN model in Figure 4-15. Similarly Each **Scheduler** block is the realization of the corresponding scheduling transition group (combined by transitions Found*i*, PollStart*i*, Pass*i*, NextG*i* *i*=1,2,3,4) in the CPN model of Figure 4-15. Similarly as in Figure 4-15, task1 and task2 have higher priority than task3 and task4 in the current model. Task1 and task3 are set as the initial last task in each task group. The scheduling result is given in the Found output port of each scheduling block. The block named **Last** in Figure 4-15 is one part of the realization of the Load transition in Figure 4-15, which is used to update the information of the last task in each group. The **NewTask** block is the other part of the realization of the Load transition in the corresponding CPN model, which focuses on loading the ID number of the new task into the IP core.

Because of the complexity of the scheduler part, the simulation result is given in Figure 5-10 and Figure 5-11 respectively. In Figure 5-10, Ready3 and Ready4 represent the 3rd and 4th ready signals in Ready[1:4]. When Ready3 becomes valid, Found3 (the Found output signal of the 3rd Scheduler block) stays invalid because scheduling is forbidden when the IP core is sleeping (1). When the sleep signal becomes invalid, both Irdy3 and Irdy4 (as the output signals of the 3rd and 4th Access block respectively) signals are enabled by their corresponding ready signals (2). The Me2 signal is the input signal for both Scheduler3 and Scheduler4 blocks in their 3rd input port. The validation of this signal indicates that neither task1 nor task2 is ready for execution. This signal enables Found4 (as the Found output signal in the 4th Scheduler Block) since Task3 is initially set as the last task (4). The scheduling result is loaded into the address bus and sent to the IP core when the LoadEn signal becomes valid (5) and the detail of task loading will be introduced in Figure 5-11. The loading of the new task also resets the corresponding ready signal in the EH part (which is introduced in section 5.1). When Ready4 becomes invalid, the Irdy4 becomes invalid accordingly (6). Therefore Found3 becomes valid since Ready3 is the only valid ready signal at this time (7).

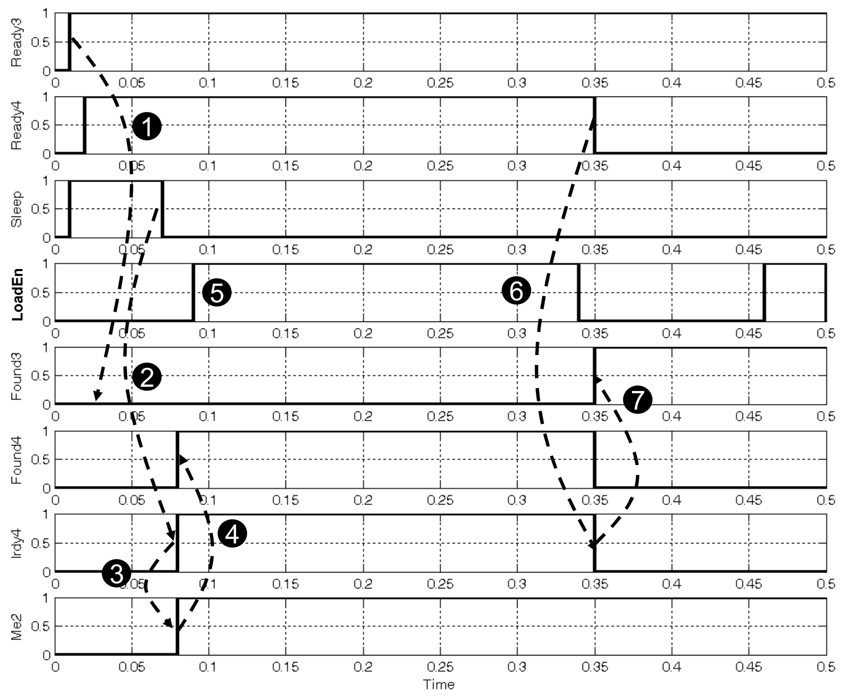


Figure 5-10: The Simulation Result of Scheduling (1)

Figure 5-11 focuses on executions in the scheduler part during the task loading processing. After the scheduling result is achieved (1), NTask4, as the 4th signal of NewTask[1:4], becomes 1 when the LoadEn signal becomes valid (2). The LoadEn signal is withdrawn when task4 has been loaded to the IP core. The withdrawal of this signal updates the record of the last task in every priority group. In current case, Last4 becomes valid and Last3 becomes invalid at the same time, because task4 is carried in the IP core (3). The withdrawal of LoadEn also triggers the Reset signal to update the record in the Event Handler. In the current case, a pulse of Reset4, as the 4th signal of Reset[1:4] is generated (4) to clean the Stim&Wait bits in Node 13 to Node 16 of the EH part in Figure 5-4. This reset operation makes Ready4 in Figure 5-10 invalid, and the change in ready signals triggers another scheduling, which chooses task3 as the new candidate for IP core's execution (5). This task is chosen as the new task to the IP core when the next valid LoadEn signal is issued (6).

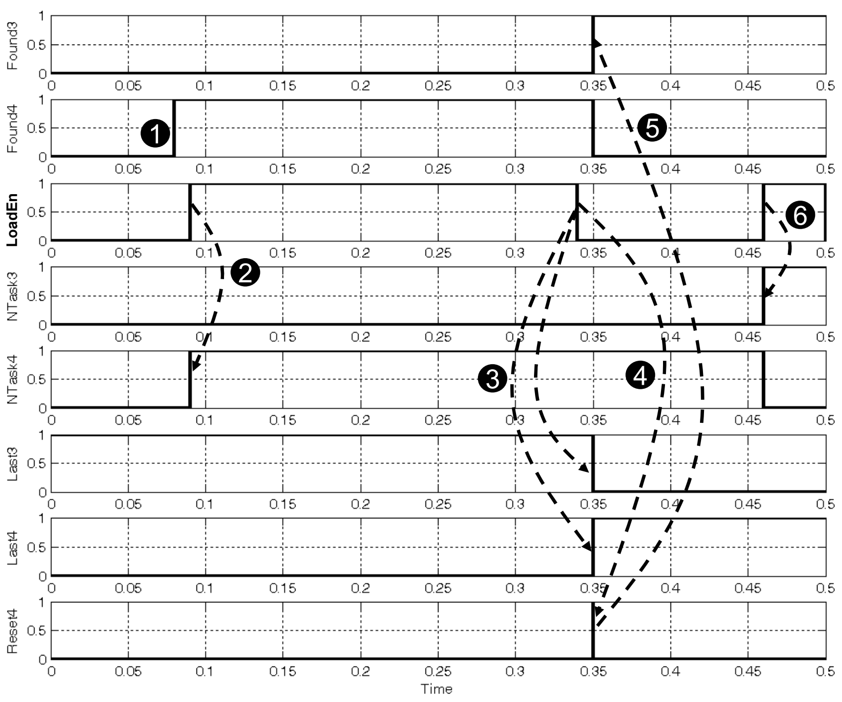


Figure 5-11: The Simulation Result of Scheduling (2)

5.4. The Design of Interface in MATLAB

Figure 5-12 is about the design of Interface part in MATLAB. The explanation of the execution in this part can be found in the corresponding CPN model in Figure 4-17 for detail.

The simulation result in this part is given in Figure 5-13. The validation of the Fire signal from the PM part sends the wakeup signal to the IP core, given that the IP core is sleeping (1). Although the sleep signal from the IP core cannot be toggled because the wakeup processing in the IP core just begins, the STEPSleep signal becomes 0 without delay (2). This signal disables the execution in the PM part and enables scheduling in the TM part. Therefore the TM can generate the scheduling result before the IP core completes its wakeup processing. A LoadEn signal is issued afterwards (3). Tasks, as the 4th row in Figure 5-13, is the output of the 4OR block which is connected with NewTask[1:4]. Therefore, when the signal Tasks is 1, it means a non-zero task ID number is loaded in the NewTask[1:4] (4).

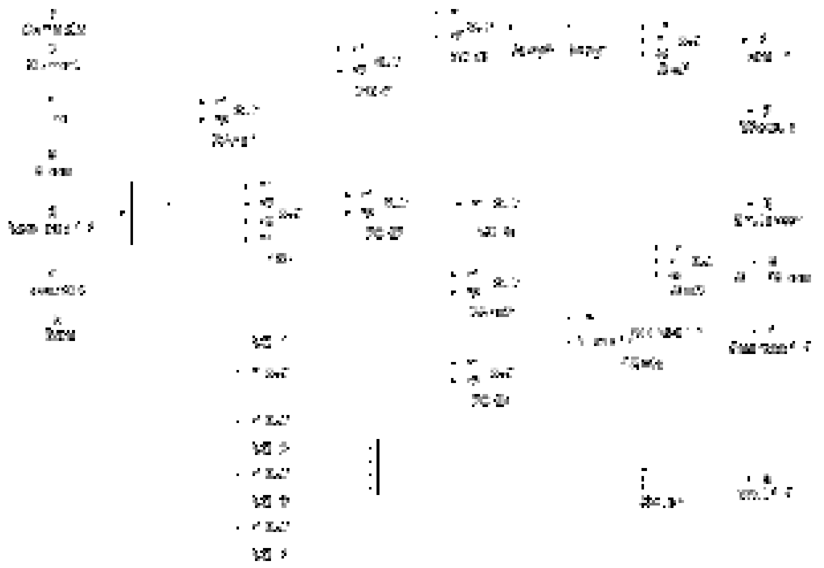


Figure 5-12: The Design of Interface Part in MATLAB

The sleep signal becomes 0 and a Read signal is generated accordingly (5) when the wakeup completes. This read signal reads the ID number of the new task chosen from the STEP to the IP core. If the IP core starts executing the corresponding task, the Current signal becomes valid (6), which withdraws the LoadEn signal (7). The following pulses in the Read signal are generated during the execution of the current task in the IP core. When the current task is completed, the output control unit, which will be introduced later in Figure 5-14, decides which VSB will control the ball movement corresponding to the current task. After the decision is made, a complete signal is issued, and this signal triggers the issuing of another LoadEn signal (8). According to Figure 5-1, both task3 and task4 can only be enabled once in VSB I because their corresponding balls move outside of playground I after one step. Therefore, the 3rd valid LoadEn signal cannot find any valid task ID number from the TM (9). In this case, a Shutdown signal is issued to IP core to start the shutdown processing (10). At the same time, the STEPSleep signal becomes 1 to start operations in the PM part and disable operations in the TM part (11). However, the IP core is woken up by the fire signal again after the shutdown processing in the IP core is completed (12).

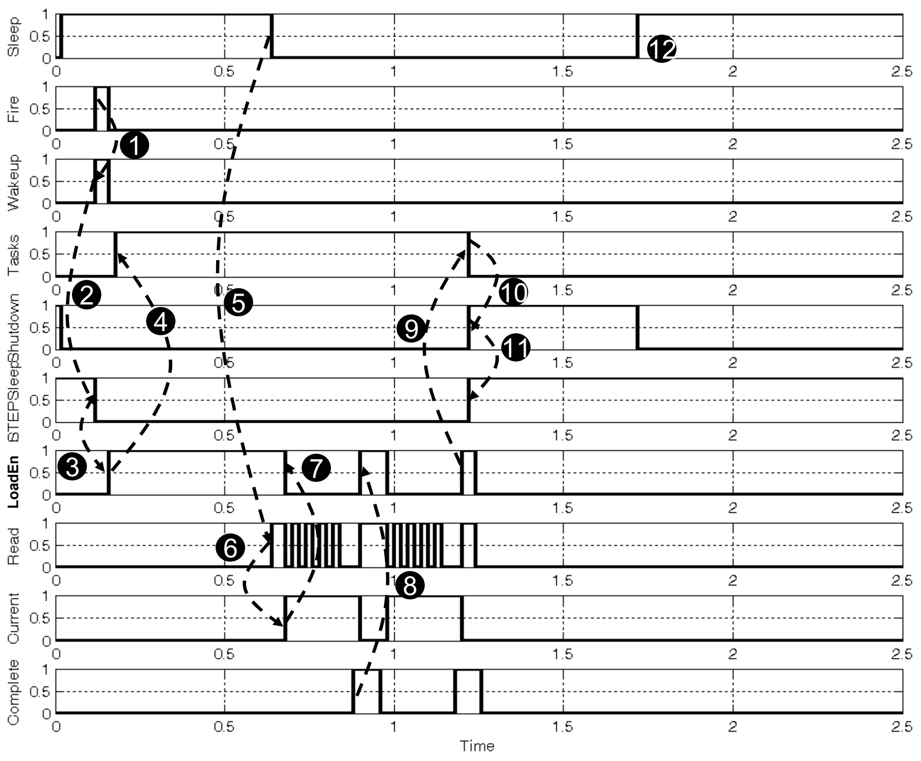


Figure 5-13: The Simulation Result of the Interface Part

5.5. The Design of the Output Control Part in MATLAB

Figure 5-14 shows the design of the Output Control part in MATLAB. When the current new position of one ball is calculated, its parameters are loaded into the data bus to be transferred to the ACM (which will be introduced in section 5.6). Therefore, the DeMux block is used to derive the PosX and PosY information from the data bus. Two comparators are used to calculate which VSB takes charge of the ball whose parameters are given on the data bus. When the decision is made, the ID number of the ball (also the ID number of the corresponding task) in the Address[1:4] port is sent to the corresponding Output Channel (Och*i*[1:4], *i*=0,1,2,3 and Och0[1:4] is the feedback channel). At the same time, a Complete signal is issued to the Interface part to enable the next task loading.

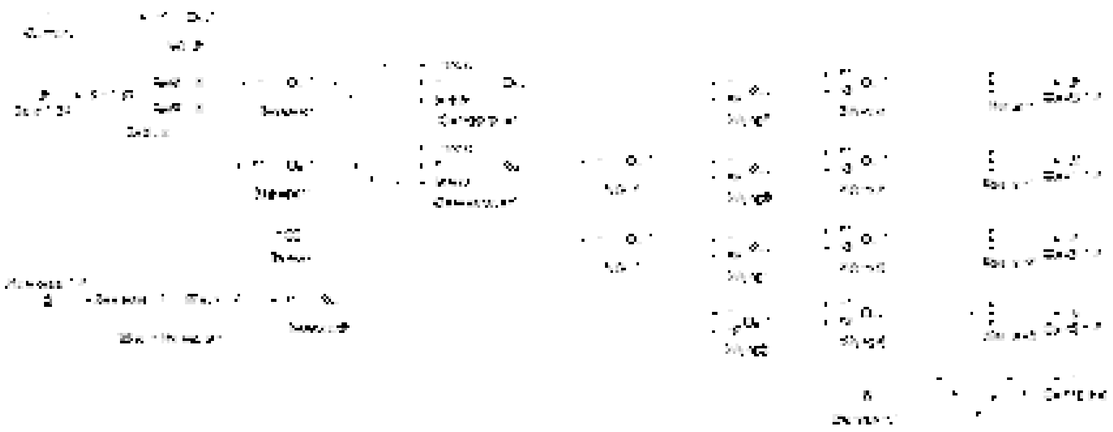


Figure 5-14: The Design of OutputControl Part in MATLAB

Figure 5-15 shows the simulation result of this part. The Enable signal is the signal in the first input port of both the Comparator and Comparator1 blocks. EdgeX and EdgeY is the X and Y position of the upper right edge of the current ball which is calculated in Comparator and Comparator1 blocks. When the Enable signal becomes 1, the two position parameters are used to decide which VSB controls the current ball (1).

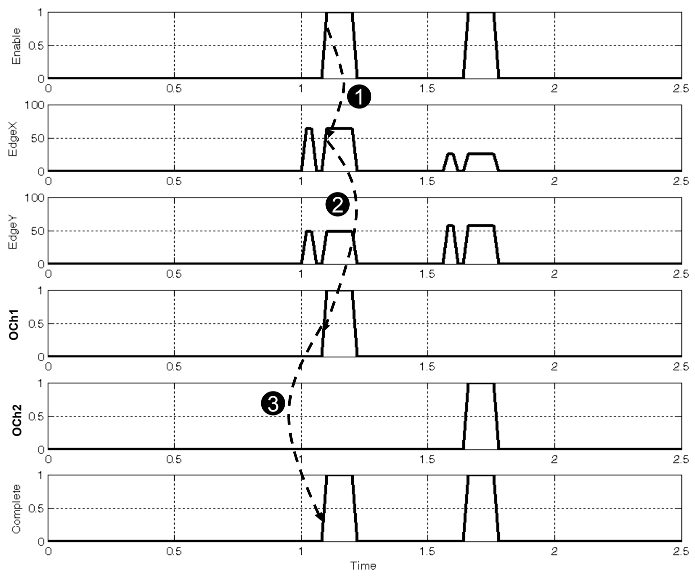


Figure 5-15: The Simulation Result of the OutControl

Och1 in Figure 5-15 is the output signal of block 2And2 in Figure 5-14. The validation of this signal indicates that the current task is sent out as an event from output channel 1 (2). At the same time, the Complete signal becomes valid (1) to enable the Interface part to start another task loading.

5.6. The Design of the IP Core Part in MATLAB

Figure 5-16 shows the design of the IP Core part in MATLAB. Two subsystem blocks are contained in this part. One is called **OS** which takes charge of IP core's wakeup/shutdown according to the commands from the corresponding STEP. It loads a new task ID number from the STEP. This part is designed based on the transitions and states in the dotted rectangle I of the CPN model in Figure 4-17. The block of **Tasks** is the combination of four embedded tasks which is shown in Figure 5-17, and this block design is based on the dotted rectangle area II of the CPN model in Figure 4-17.

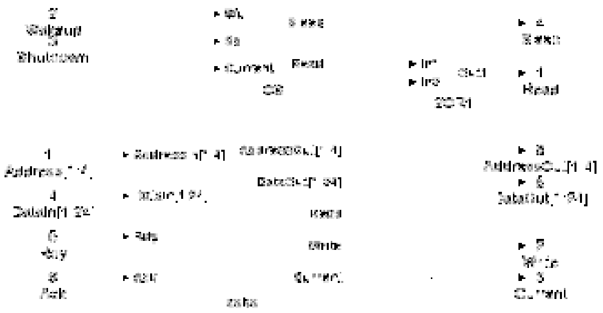


Figure 5-16: The Design of IP Core in MATLAB

As discussed before, since we only care about the function of the IP core, five S-functions are used in this part to realize both the OS block as well as the four tasks embedded in this IP core. In Figure 5-17, four blocks Input*i* (*i*=1,2,3,4) create input vectors *u* for each task S-function. Similarly, the four blocks Output*i* (*i*=1,2,3,4) derive output vectors *y* from each task and turn them into signals that can be used in the other parts of the MATLAB model. The four S-Function blocks named as Task *i* (*i*=1,2,3,4) are the embedded codes for each task.

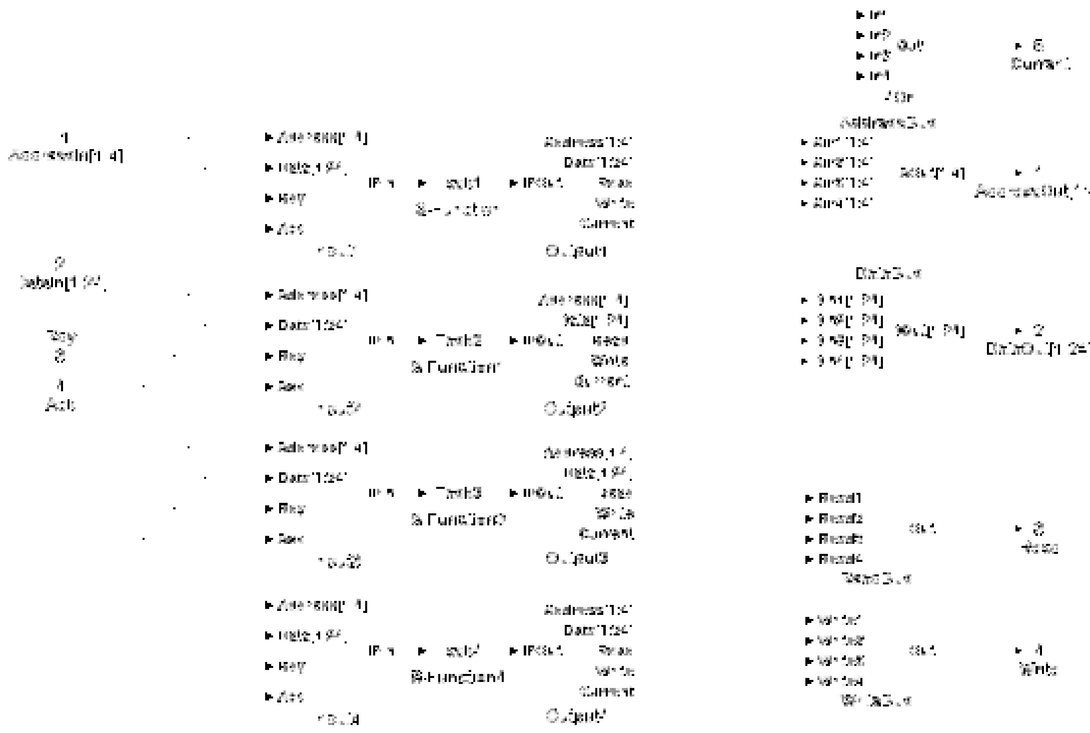


Figure 5-17: The Design of Task Subsystem Blocks in MATLAB

5.6.1 The Flow chart of the S-Function of OS

The S-function code for the OS program for VSB I is given in Appendix IX (The S-function code for other VSBs is similar). Figure 5-18 provides the flow chart of the program.

Initially, we assume that the IP core is sleeping. Therefore, variables Sleep and Read in the output vector are set to 1 and 0 respectively. When the Wakeup signal from the input vector becomes 1, the OS starts the wakeup processing. When the wakeup processing is completed, the Sleep signal is set to 0 so as to indicate the STEP that the IP core is ready for task processing. The Read signal is then set to 1 so as to load a new task from the STEP. If the IP core begins executing the new task, the Current signal in the input vector becomes 1, and it sets the Read signal in the output vector to 0.

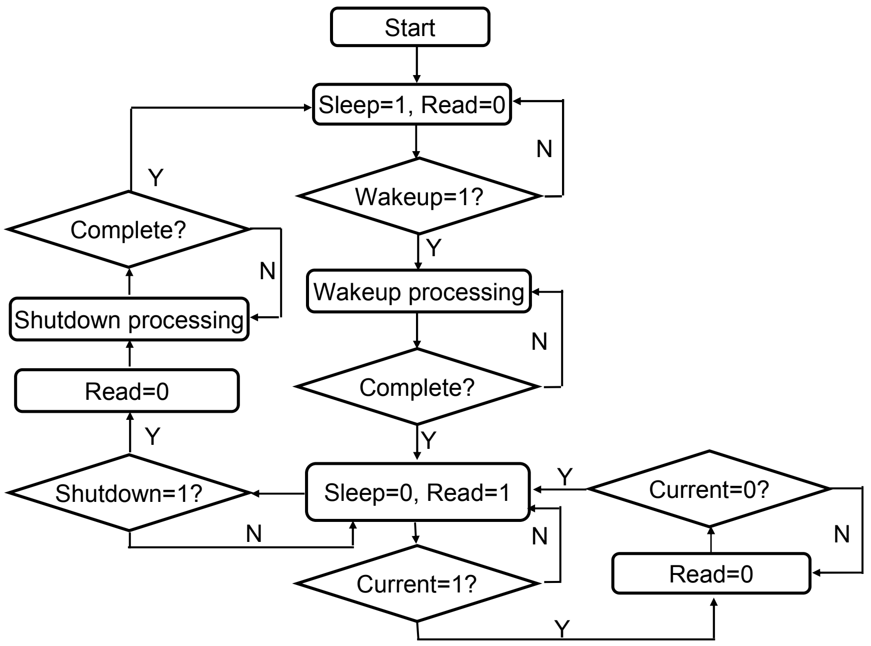


Figure 5-18: The Flow Chart of the OS program

When the Current signal becomes 0, it means that the current task is completed, and the Read signal is set to 1 again to read another new task from the STEP. This loop may continue several times before the shutdown signal from the input vector is captured. In this case, the Read signal is first set to 0 since no new task is read from the STEP, and the shutdown processing begins. When the shutdown processing is completed, the Sleep signal is reset to 1 and the IP core starts sleeping until it is activated again.

5.6.2 The Flow chart of the S-Function of Task4

The S-function code for task4 (since all task codes are similar) is given in Appendix X. Figure 5-19 shows the flow chart of the code. The program starts when its ID number (for task4, [0 0 0 1]) is loaded onto the address bus. The signal **Current** in the output vector is set to 1 so as to tell the OS to withdraw the Read signal. The first step of the task execution is to load the parameter data of all four balls from the ACM, which is carried out by the function **DataLoad** in Appendix X. Using the parameters of the previous position of Ball4, task4 can calculate the next position of the ball by the function **NextPosition**. This function lets the ball to move one step (the size of the step is determined by the speed parameter) in the direction specified by the History parameter.

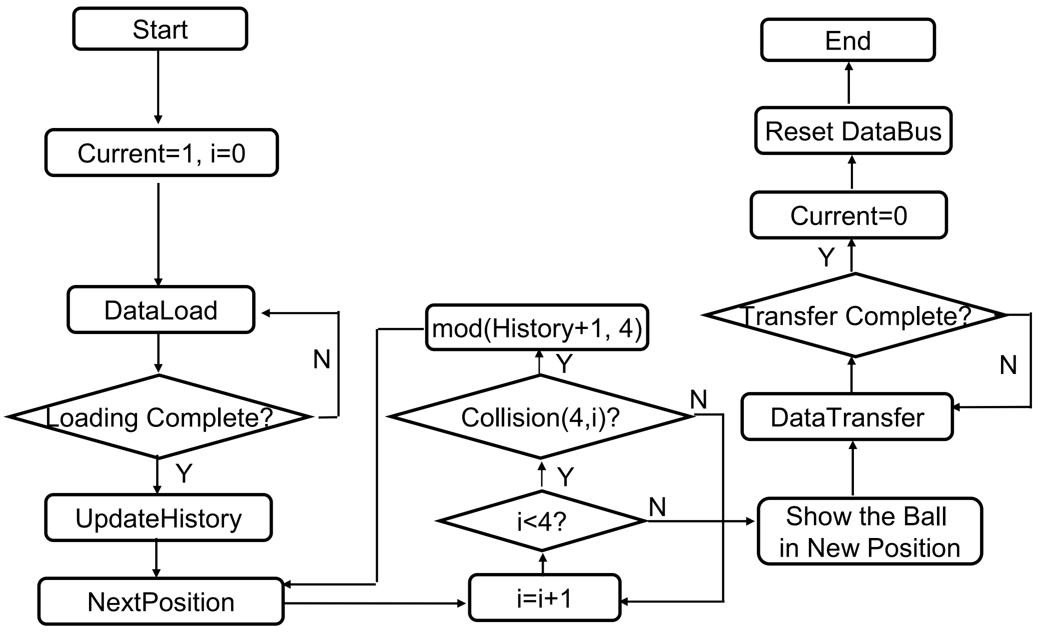


Figure 5-19: The Flow Chart of the Task4 program

Therefore, if the history parameter loaded from the ACM is used directly in the NextPosition function, the new position calculated is totally determined by the current position (unless Ball4 is knocked back by one wall of the playground or collides with other balls which will be discussed later). The **UpdateHistory** function (Figure 5-20) is used to introduce some degree of nondeterministic to the ball movement.

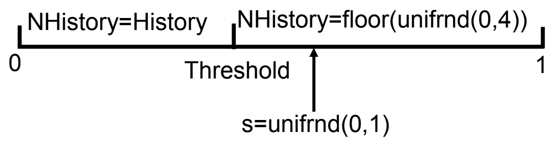


Figure 5-20: The UpdateHistory Function

This function uses the MATLAB command **unifrnd** to generate a random number from 0 to 1 which follows uniform distribution. If the random number is less than some Threshold (0≤ Threshold ≤1), the history parameter loaded from the ACM is used to calculate a new position for the ball. Otherwise, a random integer is used to generate the new ball position.

A different Threshold value gives different move mode to balls in the game. If only one ball contains in the game, the larger the Threshold value is, the more deterministic its movement becomes. However, when several balls contain in the game, the probability of collisions between different balls (which is introduced later) also increases with the rise of the Threshold value. Since a ball collision changes a ball's next movement to a random direction, what direction is chosen by the ball's movement and when the next collision happens are nondeterministic. Therefore when multiple balls contain in the game, their movements are nondeterministic no matter what Threshold value is given to each ball's movement.

The loading of the parameters for Ball4 from the POOL typed memory is used for the new position calculation. The loading of parameters of other three balls are used to check if the new position calculated by the NextPosition function can have any collision with the others. Function **Collision** in Appendix X takes charge of the collision detection and Figure 5-21 indicates the mechanism used by this function. The variable **Dis_Centres** calculate the distance between the two balls' centre. If **Width1** and **Width2** represent the width of the two balls separately, ball collision happens when $Dis\_Centres^2 < 2(\frac{Width1 + Width2}{2})^2$.

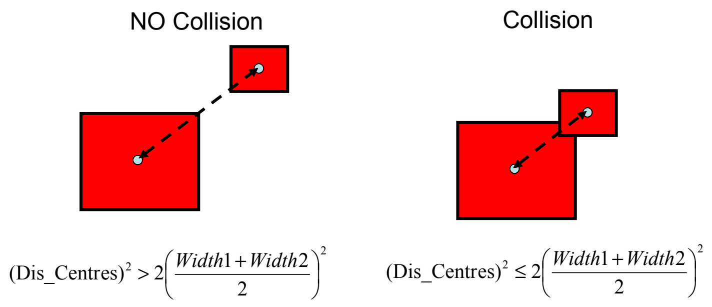

NO Collision
Collision
$(\text{Dis\_Centres})^2 > 2\left(\frac{Width1 + Width2}{2}\right)^2$
$(\text{Dis\_Centres})^2 \le 2\left(\frac{Width1 + Width2}{2}\right)^2$

Figure 5-21: The Calculation of Collision

Therefore in Figure 5-19, when the new position of ball4 is calculated, the program checks if it has collision with other three balls in sequence (*i* in Figure 5-19 represents the ball's ID number). If any collision happens, the program changes the history parameter and re-calculates the new position of Ball4 until no collision is found. After collision detection, the program can safely show the ball in the new position, and then transfer the parameters of Ball4 to the ACM. This data transfer is carried out by the **DataTransfer** function in Appendix X. When the data transfer is completed, the task4 program resets the Current signal to 0 so as to tell the STEP to do output control, and the execution is stopped after the data bus is reset.

5.7. A Test Bench of Ball Game

In our simulation, the time spent for one step movement of a ball without collision is set as one time unit. Both wakeup and shutdown executions have been adjusted so that their latency cost is one time unit as well. To simplify the analysis, we assume that the power dissipation for task processing is one unit and that for wakeup and shutdown is 1.5 units. Because wire latency cannot be reflected by MATLAB Simulink, the latency cost of a STEP model cannot be compared with that of its IP core model. Therefore, the benchmark achieved in this section regards the STEP as cost free in both power and latency.

With only four VSBs in the example SOC, events incoming to every VSB cannot be taken as an ideal exponential distribution. With only four tasks embedded in every IP core, the execution in every IP core cannot be taken as an ideal exponential distribution either. Therefore the example SOC test bench is used to analyze the power efficiency achieved by the A&F policy in a weak Markovian environment.

Four different DPM policies were used to control the four VSBs in different tests. The first one is the greedy policy which means that the threshold in the A&F part in Figure 5-7 is set to 1, therefore any ball incoming to a black playground activates the corresponding IP core.

The A&F policy is used in our second simulation. As the priorities of the four balls have been set as 1, 2, 3, 5 respectively, we set 5 as the threshold in every A&F part of STEP. This means that the incoming of ball4 only or several other balls to a black playground can activate a sleeping IP core.

The timeout policy serves as the third DPM policy in our test where we set the timeout threshold to 5 and 10 time units in two different simulations.

The prediction policy is the fourth DPM policy that is implemented. In Section 2.1.1, a T_{BE} time is defined as the minimum time spent in sleeping to compensate for the wakeup and shutdown overhead. In our case, the T_{BE} is 3 time units according to Equation 2-3. Furthermore, the implementation of the prediction policy needs to predict the length of next idle period according to Equation 2-4. A linear regression method is used in our test for idle period prediction (Equation 5-1).

$$T_{idle\_pred}^{n} = 0.5 * T_{idle}^{n-1} + 0.3 * T_{idle}^{n-2} + 0.2 * T_{idle}^{n-3} \qquad \text{Equation 5-1}$$

The prediction of the next (n^{th}) idle period depends on the last three idle periods with reliability of 0.5, 0.3 and 0.2 respectively. For the implementation of the Timeout or Prediction policy, a subsystem block of state flow is used instead of the A&F part in a STEP.

To make our test have a wide representation of real systems, we vary the threshold in Figure 5-20 from 0 to 1 in 11 independent simulations for every DPM policy's implementation. The change of this threshold from 0 to 1 indicates the variation of ball movements from pure random to mostly history based.

Figure 5-22 presents the average power dissipation of one IP core for various ball movements when different DPM policies are used. The Timeout1 in the legend indicates the case when Timeout parameter τ is set to 5, and Timeout2 is the case when τ is set to 10. From this figure, it is clear that the A&F policy is more power efficiency than the other three policies, no matter what movement the balls take.

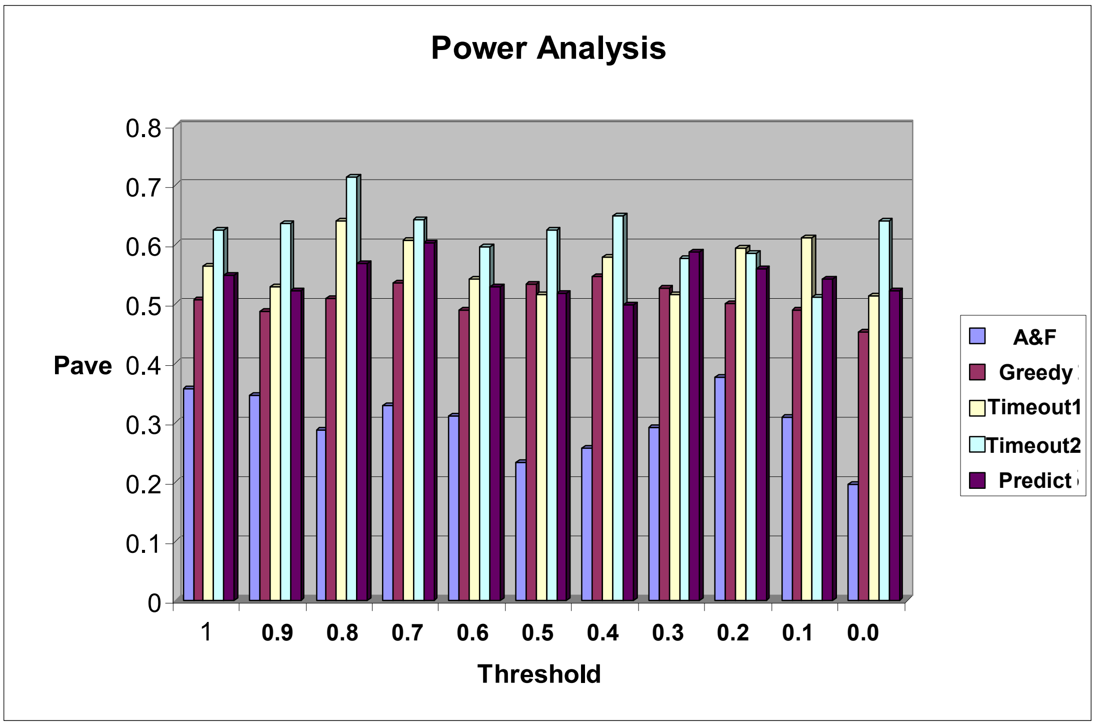


Figure 5-22: Power Analysis of Test Bench

Figure 5-23 shows the latency analysis of the test bench when the A&F policy is used. If four tasks are ready in one IP core, 2 time units are needed by one task for scheduling before execution in average, and 1 time unit is needed for execution at least (suppose no collision happens). Therefore we set the deadline (DL) for every task's execution as 6 time units in our first simulation. It can be seen front the figure that the A&F policy causes no more than 2.5% deadline violations on average. In most cases, this latency is acceptable. In our second simulation, the deadline is set to 8 time units to present the case when the deadline requirement is less strict. It can be seen that deadline violations become less accordingly.

As different priorities have been given to the four tasks, these tasks have a different latency performance. Figure 5-24 presents the different latency performance of the four tasks when the deadline is set to 6. It can be seen that task4, who has the highest priority, has extremely low deadline violation cases. It is because no latency cost is paid in task accumulation.

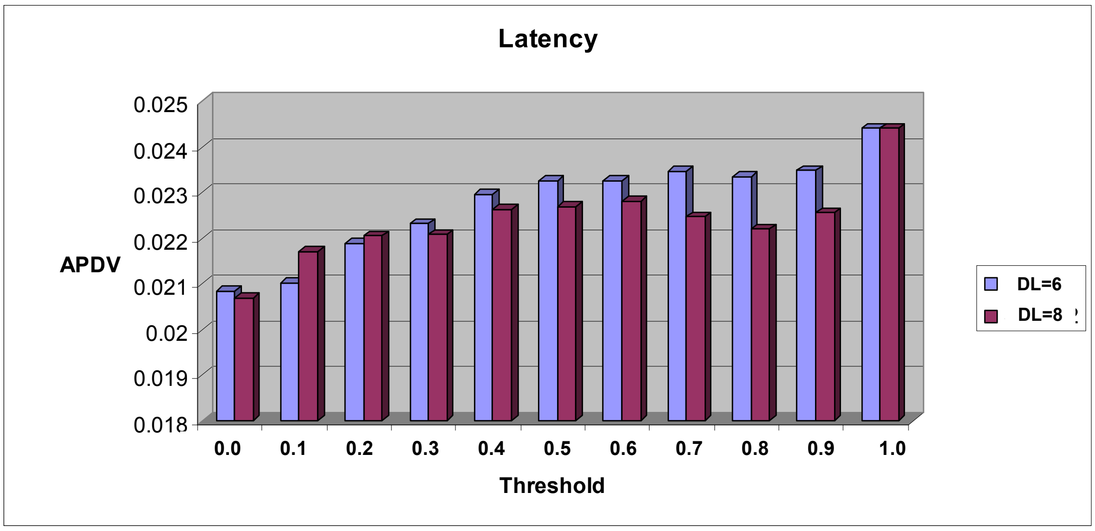


Figure 5-23: Latency Analysis of Test Bench

According to Figure 5-24, task2 and task3 have more frequent deadline violations than task1, although they have a higher priority than the latter. It is mainly caused by the parameter settings of these balls. According to Table 5-1, ball2 and ball3 have a higher speed than ball1, which means ball2 or ball3 moves more frequently across different playgrounds than ball1. When ball2 or ball3 moves to a new playground whose corresponding IP core is sleeping, it needs another balls' arrival to activate the IP core and much latency is involved during task accumulation. On the other hand, ball1, which has small size and low speed, always moves within one playground. Therefore this ball pays less cost in accumulation latency than ball2 or ball3.

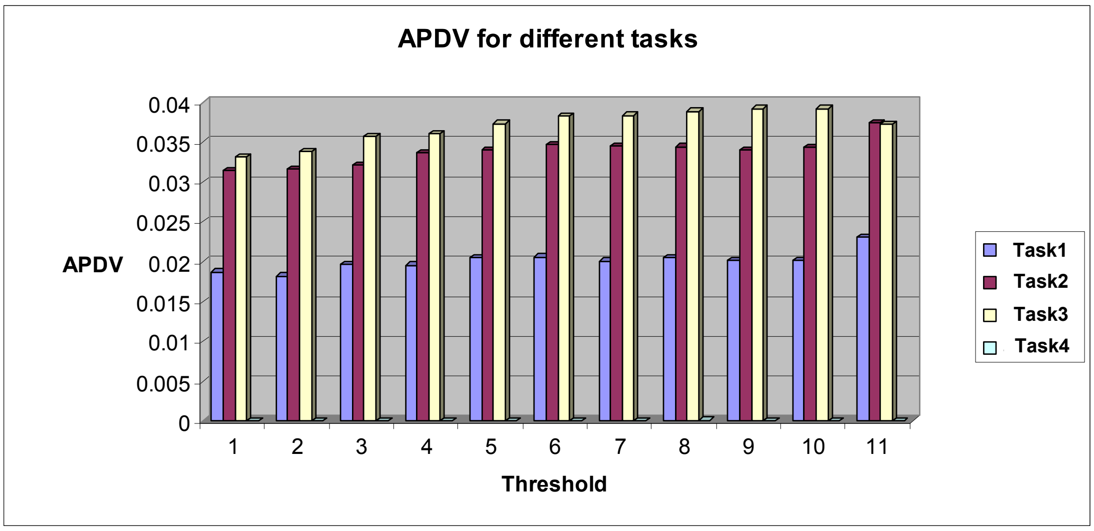


Figure 5-24: Latency Analysis for Test Bench (Continue)

5.8. Conclusions

This chapter presents an example SOC which is constructed from four VSBs in MATLAB Simulink. All parts of a STEP that are modelled by CPN models in Chapter 4 have been built by basic components in the Simulink Library. An example IP core with four embedded tasks is designed in the Simulink S-function. The example SOC is used to carry out a test bench named as a ball game in MATLAB simulation, and the simulation result achieved from the test bench not only proves the correctness of executions in the VSB based SOC, but also indicates the high energy efficiency of the A&F policy even in a weak Markovian environment.

Chapter 6

Conclusions and Future Work

6. Conclusions and Future Work

In recent years, IP cores have been widely used in SOC design under the GALS architecture. Asynchronous wrappers and ACMs are used to provide asynchronous communication for IP cores belonging to different time domains. When power instead of throughput becomes the main bottleneck of the system performance, various low power technologies in gate and transistor levels in hardware design and event driven programmes in software design help IP cores to reduce their power dissipation. However, no low power consideration so far has been provided in the GALS architecture so as to optimize system performance (mainly in power dissipation) in a SOC scheme. In this thesis, an asynchronous coprocessor named as Self-Timed Event Processor (STEP) is modelled and designed, which provides event handling, power control, task scheduling as well as asynchronous communication for each IP core in an SOC with low overheads. The combination of one IP core and its STEP forms a virtual self-timed block (VSB) since it works as a self-timed domain in the asynchronous environment.

As the demonstration of the motivation validity for such design, stochastic models in Chapter 3 were used for the power-latency analysis of a virtual self-timed block when different DPM policies have been implemented. Not only various modes provided by an IP core, but also mode switching transitions have been modelled in our stochastic (mainly Markov) models. A stochastic DPM policy named as Accumulation & Fire (A&F) is verified to be promising in trading latency for power, and this policy has relatively easy hardware implementation.

In Chapter 4, hierarchical CPN models were built to demonstrate and analyze a general architecture of a STEP, following some basic functional specifications. Concurrent processing between different components of a STEP as well as that between a STEP and its IP core were highlighted in all CPN models, and simulation and state space checking were used to correct possible design errors in our design.

Chapter 5 introduced an example SOC which is built by four VSBs in MATLAB Simulink. The construction of all these VSBs follows the specification given in the previous CPN models. A test bench named as ball game is running in the example SOC and simulation results show the energy efficiency of our design.

Further study can be done in different aspects in the future. Since all executions in a portable device rely on the energy stored in an on board battery, we prefer to combine our stochastic models presented in this thesis with that for battery given in [luci08], so as to analyze the optimized performance of an IP core (as well as a SOC) when the nonlinear power consumption in the battery is taken into consideration.

Secondly, current STEP model only provides on-off power control to its IP core. In the future, new CPN models of a STEP will be built to provide more delicate control to use various operation modes provided by advanced IP cores to further optimize system performance.

Finally, current STEP model needs to be implemented into VLSI design. In this case, we can first test the power and latency of the STEP in real case, so as to prove the A&F policy has very low overheads. Furthermore, several real IP cores can be used to cooperate with our STEP to analyze the power efficiency of the A&F policy. The construction of SOC with VSBs will be carried afterwards.

Appendix

Appendix I: Analytical Solution Derivation for the Greedy Policy in an On-Off DPM System

In this section, we use the probability of state 1 in Figure 3-1 (Q_1) to represent the probability of other states.

The probabilities of shutdown states

If Q_{sd0} is the probability of shutdown state sd0, the relationship between Q_1 and Q_{sd0} is shown in Equation a-1.

$$\mu Q_1 = (\lambda + \gamma) Q_{sd0} \qquad \text{Equation a-1}$$

Therefore, we can use Q_1 to represent Q_{sd0} in Equation a-2.

$$Q_{sd0} = \frac{\mu}{\lambda + \gamma} Q_1 \qquad \text{Equation a-2}$$

Q_{sd1} can also be represented by Q_1 with the help of Q_{sd0} (Equation a-3).

$$Q_{sd1} = \frac{\lambda}{\lambda + \gamma} Q_{sd0} = \frac{\mu}{\lambda + \gamma} \frac{\lambda}{\lambda + \gamma} Q_1 \qquad \text{Equation a-3}$$

We can carry on the derivation for Q_{sd2}, Q_{sd3} and so on. The general expression for the probability of shutdown states $Q_{sd(n)}$ (n=0,1,2,…) is given in Equation 3-2 or Equation a-4.

$$Q_{sd(n)} = \frac{\mu}{\lambda + \gamma} \left(\frac{\lambda}{\lambda + \gamma}\right)^n Q_1 \qquad \text{Equation 3-2 or Equation a-4}$$

If S_{sd} is the sum of the probabilities of all shutdown states, we can derive

$$S_{sd} = \sum_{n=0}^{\infty} Q_{sd(n)} = \frac{\mu Q_1}{\lambda+\gamma}(1+\frac{\lambda}{\lambda+\gamma}+(\frac{\lambda}{\lambda+\gamma})^2+...)$$ Equation 3-16 or Equation a-5

$$= \frac{\mu Q_1}{\lambda+\gamma}\frac{1}{1-\frac{\lambda}{\lambda+\gamma}} = \frac{\mu Q_1}{\lambda+\gamma}\frac{\lambda+\gamma}{\gamma} = \frac{\mu}{\gamma}Q_1$$

The probabilities of inactive states

According to Figure 3-1, only one state 0* belongs to inactive state. With the help of Q_{sd0}, we can represent Q_{0*} by Q_1 in Equation 3-13 or Equation a-6.

$$Q_{0*} = \frac{\gamma}{\lambda}Q_{sd0} = \frac{\gamma}{\lambda}\frac{\mu}{\lambda+\gamma}Q_1$$ Equation 3-13 or Equation a-6

The probabilities of wakeup states

If Q_{wu1} is the probability of wakeup state wu1, the relationship among Q_{wu1}, Q_{sd1} and Q_{0*} is shown in Equation a-7.

$$\gamma Q_{sd1} + \lambda Q_{0*} = (\lambda+\delta)Q_{wu1}$$ Equation a-7

Therefore, we can derive

$$Q_{wu1} = \frac{\gamma}{\lambda+\delta}Q_{sd1} + \frac{\lambda}{\lambda+\delta}Q_{0*}$$ Equation a-8

Similarly, we have

$$Q_{wu2} = \frac{\gamma}{\lambda+\delta}Q_{sd2} + \frac{\lambda}{\lambda+\delta}Q_{wu1}$$ Equation a-9

$$= (\frac{\gamma}{\lambda+\delta}\frac{\lambda}{\lambda+\gamma} + \frac{\lambda}{\lambda+\delta}\frac{\gamma}{\lambda+\delta})Q_{sd2} + (\frac{\lambda}{\lambda+\delta})^2 Q_{0*}$$

$$Q_{wu3} = \frac{\gamma}{\lambda+\delta}Q_{sd3} + \frac{\lambda}{\lambda+\delta}Q_{wu2}$$ Equation a-10

$$= \frac{\gamma}{\lambda+\delta}[(\frac{\lambda}{\lambda+\gamma})^2 + \frac{\lambda}{\lambda+\delta}(\frac{\lambda}{\lambda+\gamma}) + (\frac{\lambda}{\lambda+\delta})^2]Q_{sd1} + (\frac{\lambda}{\lambda+\delta})^3 Q_{0*}$$

Therefore, the general expression for the probability of wakeup state $Q_{wu(n)}$ (n>0) is given in Equation 3-14 or Equation a-11.

$$Q_{wu(n)} = \frac{\gamma Q_{sd1}}{\lambda+\delta}\sum_{k=0}^{n-1}[(\frac{\lambda}{\lambda+\gamma})^k(\frac{\lambda}{\lambda+\delta})^{n-1-k}]+(\frac{\lambda}{\lambda+\delta})^n Q_{0*}$$

Equation 3-14 or Equation a-11

If S_{wu} is the sum of the probabilities of all wakeup states, the calculation of S_{wu} can be carried out in the following steps:

First of all, we define

$$X_1 = \sum_{n=1}^{\infty}\sum_{k=0}^{n-1}[(\frac{\lambda}{\lambda+\gamma})^k(\frac{\lambda}{\lambda+\delta})^{n-1-k}]$$ Equation a-12

And

$$X_2 = \sum_{n=1}^{\infty}(\frac{\lambda}{\lambda+\delta})^n$$ Equation a-13

Therefore

$$S_{wu} = \sum_{n=1}^{\infty}Q_{wu(n)} = \frac{\gamma Q_{sd1}}{\lambda+\delta}X_1 + X_2 Q_{0*}$$ Equation a-14

X_2 can be easily calculated in Equation a-15.

$$X_2 = \sum_{n=1}^{\infty}(\frac{\lambda}{\lambda+\delta})^n = \frac{\lambda}{\lambda+\delta}\frac{1}{1-\frac{\lambda}{\lambda+\delta}} = \frac{\lambda}{\delta}$$ Equation a-15

In Equation a-12, it can be seen that all components in X_1 are the functions of n and k. For example, when n=1 and k=0, we have our first component 1. When n=2, k=0, the second component is $\frac{\lambda}{\lambda+\gamma}$, and when n=2, k=1, we have $\frac{\lambda}{\lambda+\delta}$.

For better understanding the relationship of all components in X_1 so as to calculate the sum of the probability of all wakeup states, we list all components in X_1 in a matrix way. The value for the (n, k) component of X_1 is given in the (n, k+1) unit of the matrix.

X_1=

	k=0	k=1	k=2	k=3	…
n=1	1				
n=2	$\frac{\lambda}{\lambda+\gamma}$	$\frac{\lambda}{\lambda+\delta}$			
n=3	$(\frac{\lambda}{\lambda+\gamma})^2$	$\frac{\lambda}{\lambda+\gamma}\frac{\lambda}{\lambda+\delta}$	$(\frac{\lambda}{\lambda+\delta})^2$		
n=4	$(\frac{\lambda}{\lambda+\gamma})^3$	$(\frac{\lambda}{\lambda+\gamma})^2\frac{\lambda}{\lambda+\delta}$	$\frac{\lambda}{\lambda+\gamma}(\frac{\lambda}{\lambda+\delta})^2$	$(\frac{\lambda}{\lambda+\delta})^3$	…
…	…	…	…	…	…

If Y_1 is the sum of the first column of the X_1 matrix (which is highlighted by the grey shade), the result is:

$$Y_1 = 1 + \frac{\lambda}{\lambda+\gamma} + (\frac{\lambda}{\lambda+\gamma})^2 + ... = \sum_{n=1}^{\infty}(\frac{\lambda}{\lambda+\gamma})^n = \frac{\lambda+\gamma}{\gamma} \qquad \text{Equation a-16}$$

Similarly, Y_2 as the sum of the second column of the matrix, the result is:

$$Y_2 = \frac{\lambda}{\lambda+\delta}[1 + \frac{\lambda}{\lambda+\gamma} + (\frac{\lambda}{\lambda+\gamma})^2 + ...] = \frac{\lambda}{\lambda+\delta}\frac{\lambda+\gamma}{\gamma} \qquad \text{Equation a-17}$$

The sums of the rest columns like Y_3, Y_4 and so on share the similar form as Equation a-17. Now, we can derive X_1 by adding all the column value together (Equation a-18).

$$X_1 = Y_1 + Y_2 + Y_3 + ... = \frac{\lambda+\gamma}{\gamma}[1 + \frac{\lambda}{\lambda+\delta} + (\frac{\lambda}{\lambda+\delta})^2 + ...] = \frac{\lambda+\gamma}{\gamma}\frac{\lambda+\delta}{\delta}$$

Equation a-18

S_{wu} can be calculated by integrating Equation a-15 and Equation a-18 into Equation a-14, and the final result is given below.

$$S_{wu} = \frac{\mu}{\delta}Q_1 \qquad \text{Equation 3-18 or Equation a-19}$$

The probabilities of active states

If Q_n (n>0) is the probability of active state n, Q_n can be represented by the probabilities of neighbour states Q_{n-1} and Q_{n+1} and the connected wakeup state $Q_{wu(n)}$ (Equation a-20 or Equation a-21).

$$\mu Q_2 + \delta Q_{wu1} = (\lambda + \mu) Q_1 \qquad \text{Equation a-20}$$

$$\lambda Q_{n-1} + \mu Q_{n+1} + \delta Q_{wu(n)} = (\mu + \lambda) Q_n \ (n>1) \qquad \text{Equation a-21}$$

The general expression for the probability of active state n is given in Equation a-22 accordingly.

$$Q_n = \sum_{k=1}^{n} (\frac{\lambda}{\mu})^{k-1} Q_1 - \frac{\delta}{\mu} [\sum_{k=1}^{n-1} \sum_{s=k}^{n-1} (\frac{\lambda}{\mu})^{n-s} Q_{wu(k)}] \qquad \text{Equation 3-15 or Equation a-22}$$

In order to calculate S_a, as the sum of the probabilities of all active states, we use the similar way as the derivation of S_{wu}. First we define X_1 and X_2 in the following equations.

$$X_1 = \sum_{n=1}^{\infty} \sum_{k=1}^{n} (\frac{\lambda}{\mu})^{k-1} \qquad \text{Equation a-23}$$

$$X_2 = \sum_{n=2}^{\infty} \sum_{k=1}^{n-1} \sum_{s=k}^{n-1} (\frac{\lambda}{\mu})^{n-s} Q_{wu(k)} \qquad \text{Equation a-24}$$

Therefore, we have

$$S_a = \sum_{i=1}^{\infty} Q_i = X_1 Q_1 - \frac{\delta}{\mu} X_2 \qquad \text{Equation a-25}$$

We rewrite X_1 as a matrix according to n, k:

X_1=

1					
1	$\frac{\lambda}{\mu}$				
1	$\frac{\lambda}{\mu}$	$(\frac{\lambda}{\mu})^2$			
1	$\frac{\lambda}{\mu}$	$(\frac{\lambda}{\mu})^2$	$(\frac{\lambda}{\mu})^3$		
1	$\frac{\lambda}{\mu}$	$(\frac{\lambda}{\mu})^2$	$(\frac{\lambda}{\mu})^3$	$(\frac{\lambda}{\mu})^4$	$\cdots$

…	…	…	…	…	…

Suppose T is the total number of active states, therefore the sum of the first column is T and the sum of the second column is $(T-1)\frac{\lambda}{\mu}$. X_1, as the sum of the matrix becomes:

$$X_1 = T + (T-1)\frac{\lambda}{\mu} + (T-2)(\frac{\lambda}{\mu})^2 + \ldots \qquad \text{Equation a-26}$$

When X_1 is multiplied by $\frac{\lambda}{\mu}$, we have

$$\frac{\lambda}{\mu}X_1 = T\frac{\lambda}{\mu} + (T-1)(\frac{\lambda}{\mu})^2 + (T-2)(\frac{\lambda}{\mu})^3 + \ldots \qquad \text{Equation a-27}$$

When we subtract Equation a-27 by Equation a-26, we have

$$(\frac{\lambda}{\mu}-1)X_1 = -T + \frac{\lambda}{\mu} + (\frac{\lambda}{\mu})^2 + (\frac{\lambda}{\mu})^3 + \ldots \qquad \text{Equation a-28}$$

Therefore

$$X_1 = \frac{\mu T}{\mu-\lambda} - \frac{\lambda\mu}{(\mu-\lambda)^2} \qquad \text{Equation a-29}$$

Next, we rewrite X_2 as a matrix according to n, k:

X_2=

$$\begin{array}{|llll|}
\hline
Q_{wu1} & & & \\
(1+\frac{\lambda}{\mu})Q_{wu1} & Q_{wu2} & & \\
(1+\frac{\lambda}{\mu}+(\frac{\lambda}{\mu})^2)Q_{wu1} & (1+\frac{\lambda}{\mu})Q_{wu2} & Q_{wu3} & \\
\ldots & \ldots & \ldots & \ldots \\
\hline
\end{array}$$

If the total number of all active states is T, then the rank of X_2 should be T-1 because the expression of Q_1 does not have any $Q_{wu(n)}$ component. Now, we first calculate the sum of the first column of the X_2 matrix.

$$Y_1 = Q_{wu1} + (1+\frac{\lambda}{\mu})Q_{wu1} + (1+\frac{\lambda}{\mu}+(\frac{\lambda}{\mu})^2)Q_{wu1} + \dots$$
$$= [(T-1)+(T-2)\frac{\lambda}{\mu}+(T-3)(\frac{\lambda}{\mu})^2+\dots]Q_{wu1}$$ Equation a-30

Similarly as the calculation of X_1, we multiply Y_1 by $\frac{\lambda}{\mu}$:

$$\frac{\lambda}{\mu}Y_1 = [(T-1)\frac{\lambda}{\mu}+(T-2)(\frac{\lambda}{\mu})^2+(T-3)(\frac{\lambda}{\mu})^3+\dots]Q_{wu1}$$ Equation a-31

The value of Y_1 can be achieved by subtracting Equation a-30 by Equation a-31.

$$Y_1 = [\frac{\mu(T-1)}{\mu-\lambda} - \frac{\lambda\mu}{(\mu-\lambda)^2}]Q_{wu1}$$ Equation a-32

The sum of the following columns can be computed using the similar steps as above. For example, the expression for Y_2 and Y_3 are given in Equation a-33 and Equation a-34 respectively.

$$Y_2 = [\frac{\mu(T-2)}{\mu-\lambda} - \frac{\lambda\mu}{(\mu-\lambda)^2}]Q_{wu2}$$ Equation a-33

$$Y_3 = [\frac{\mu(T-3)}{\mu-\lambda} - \frac{\lambda\mu}{(\mu-\lambda)^2}]Q_{wu3}$$ Equation a-34

Now, we can use Y_j (j=1,2,3,…) to represent X_2 in Equation a-35.

$$X_2 = [\frac{\mu(T-1)}{\mu-\lambda} - \frac{\lambda\mu}{(\mu-\lambda)^2}]Q_{wu1} + [\frac{\mu(T-2)}{\mu-\lambda} - \frac{\lambda\mu}{(\mu-\lambda)^2}]Q_{wu2} + \dots$$
$$= \frac{\mu}{\mu-\lambda}[(T-1)Q_{wu1}+(T-2)Q_{wu2}+\dots] - \frac{\lambda\mu}{(\mu-\lambda)^2}(Q_{wu1}+Q_{wu2}+\dots)$$ Equation a-35

If we define

$$Z_1 = (T-1)Q_{wu1} + (T-2)Q_{wu2} + \dots$$ Equation a-36

And

$$Z_2 = Q_{wu1} + Q_{wu2} + Q_{wu3} + \dots$$ Equation a-37

X_2 in Equation a-35 can be represented by the combination of Z_1 and Z_2 (Equation a-38).

$$X_2 = \frac{\mu}{\mu - \lambda} Z_1 - \frac{\lambda\mu}{(\mu - \lambda)^2} Z_2 \qquad \text{Equation a-38}$$

Z_2, as the sum of the probabilities of all wakeup states, has been calculated in Equation a-19. Equation a-11gives the general expression of all wakeup states, and we integrate this equation to Equation a-36, so we have

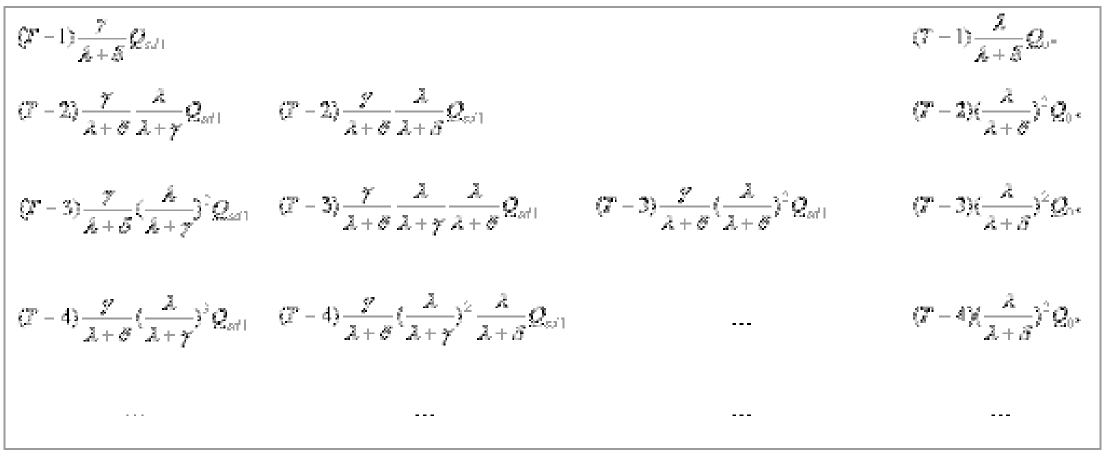
$$X_2 = (T-1)\frac{\gamma}{\lambda+\delta}Q_{sd1} + \qquad (T-1)\frac{\gamma}{\lambda+\delta}Q_{0*}$$

$$(T-2)\frac{\gamma}{\lambda+\delta}\frac{\lambda}{\lambda+\gamma}Q_{sd1} + (T-2)\frac{\gamma}{\lambda+\delta}\frac{\lambda}{\lambda+\delta}Q_{sd1} + \qquad (T-2)\frac{\gamma}{\lambda+\delta}Q_{0*}$$

$$(T-3)\frac{\gamma}{\lambda+\delta}(\frac{\lambda}{\lambda+\gamma})^2 Q_{sd1} + (T-3)\frac{\gamma}{\lambda+\delta}\frac{\lambda}{\lambda+\gamma}\frac{\lambda}{\lambda+\delta}Q_{sd1} + \ldots..$$

It can be seen that components $(T-1)\frac{\gamma}{\lambda+\delta}Q_{sd1}$, $(T-2)\frac{\gamma}{\lambda+\delta}\frac{\lambda}{\lambda+\gamma}Q_{sd1}$ and so on behave like a number serial, and components $(T-2)\frac{\gamma}{\lambda+\delta}\frac{\lambda}{\lambda+\delta}Q_{sd1}$, $(T-3)\frac{\gamma}{\lambda+\delta}\frac{\lambda}{\lambda+\gamma}\frac{\lambda}{\lambda+\delta}Q_{sd1}$ behave like another serial. In order to emphasize all number serials in the calculation, we rewrite X_2 in a matrix way (all plus (+) symbols are omitted):

Z_2=



$$\begin{array}{llll}
(T-1)\frac{\gamma}{\lambda+\delta}Q_{sd1} & & & (T-1)\frac{\lambda}{\lambda+\delta}Q_{0*} \\
(T-2)\frac{\gamma}{\lambda+\delta}\frac{\lambda}{\lambda+\gamma}Q_{sd1} & (T-2)\frac{\gamma}{\lambda+\delta}\frac{\lambda}{\lambda+\delta}Q_{sd1} & & (T-2)(\frac{\lambda}{\lambda+\delta})^2 Q_{0*} \\
(T-3)\frac{\gamma}{\lambda+\delta}(\frac{\lambda}{\lambda+\gamma})^2 Q_{sd1} & (T-3)\frac{\gamma}{\lambda+\delta}\frac{\lambda}{\lambda+\gamma}\frac{\lambda}{\lambda+\delta}Q_{sd1} & (T-3)\frac{\gamma}{\lambda+\delta}(\frac{\lambda}{\lambda+\delta})^2 Q_{sd1} & (T-3)(\frac{\lambda}{\lambda+\delta})^2 Q_{0*} \\
(T-4)\frac{\gamma}{\lambda+\delta}(\frac{\lambda}{\lambda+\gamma})^3 Q_{sd1} & (T-4)\frac{\gamma}{\lambda+\delta}(\frac{\lambda}{\lambda+\gamma})^2\frac{\lambda}{\lambda+\delta}Q_{sd1} & \cdots & (T-4)(\frac{\lambda}{\lambda+\delta})^2 Q_{0*} \\
\cdots & \cdots & \cdots & \cdots
\end{array}$$

Similarly as the calculation of Z_1 matrix, we do the calculation according to the columns. If Z_{c1} is the sum of the first column of Z_1, we have:

$$Z_{c1} = \frac{\gamma}{\lambda+\delta}[(T-1)+(T-2)\frac{\lambda}{\lambda+\gamma}+(T-3)(\frac{\lambda}{\lambda+\gamma})^2+...]Q_{sd1} \qquad \text{Equation a-39}$$

The serial of Z_{c1} is similar to the serial of X_1 in Equation a-26, and the serial of Y_1 in Equation a-31, and we can use the similar method to get the solution.

$$Z_{c1} = [\frac{(T-1)(\lambda+\gamma)}{\gamma}-\frac{\lambda(\lambda+\gamma)}{\gamma^2}]\frac{\gamma}{\lambda+\delta}Q_{sd1} \qquad \text{Equation a-40}$$

The sum of the following columns ($Z_{c2,}$ Z_{c3} and so on) can be calculated in the similar steps. Therefore, we have:

$$Z_{c(n)}=\begin{cases} [\frac{(T-n)(\lambda+\gamma)}{\gamma}-\frac{\lambda(\lambda+\gamma)}{\gamma^2}](\frac{\lambda}{\lambda+\delta})^{n-1}\frac{\gamma}{\lambda+\delta}Q_{sd1} & (n<T) \\ [(T-1)\frac{\lambda}{\delta}-(\frac{\lambda}{\delta})^2]Q_{0*} & (n=T) \end{cases} \qquad \text{Equation a-41}$$

If we use the $Z_{c(i)}$ in Equation a-41 to represent Z_2, we have:

$$Z_2 = Z_{c1}+Z_{c2}+...+Z_{c(n)} \qquad \text{Equation a-42}$$

$$= \frac{\lambda+\gamma}{\gamma}\frac{\gamma}{\lambda+\delta}[(T-1)+(T-2)(\frac{\lambda}{\lambda+\delta})+(T-3)(\frac{\lambda}{\lambda+\delta})^2+...]Q_{sd1}$$

$$-\frac{\lambda(\lambda+\gamma)}{\gamma^2}\frac{\gamma}{\lambda+\delta}[1+(\frac{\lambda}{\lambda+\delta})+(\frac{\lambda}{\lambda+\delta})^2+...]Q_{sd1}+[(T-1)\frac{\lambda}{\delta}-(\frac{\lambda}{\delta})^2]Q_{0*}$$

$$=\frac{\lambda+\gamma}{\lambda+\delta}[\frac{(T-1)(\lambda+\delta)}{\delta}-\frac{\lambda(\lambda+\gamma)}{\delta^2}]Q_{sd1}-\frac{\lambda(\lambda+\gamma)}{\gamma(\lambda+\delta)}\frac{\lambda+\delta}{\delta}Q_{sd1}+[(T-1)\frac{\lambda}{\delta}-(\frac{\lambda}{\delta})^2]Q_{0*}$$

Using Equation a-3 and Equation a-6, we can use Q_1 instead of Q_{sd1} and Q_{0*} to represent Z_2.

$$Z_2 = \frac{\mu}{\delta}[(T-1)-\frac{\lambda}{\delta}-\frac{\lambda^2}{(\lambda+\gamma)\gamma}]Q_1 \qquad \text{Equation a-43}$$

When we integrate Equation a-19 and Equation a-43 into Equation a-38, we have

$$X_2 = \frac{\mu}{\mu-\lambda} Z_1 - \frac{\lambda\mu}{(\mu-\lambda)^2} Z_2$$ Equation a-44

$$= \frac{\mu}{\mu-\lambda}[-(T-1) + \frac{\lambda}{\mu-\lambda} + \frac{\lambda}{\delta} + \frac{\lambda^2}{(\lambda+\gamma)\gamma}]Q_1$$

When we integrate Equation a-29 and Equation a-44 into Equation a-25, we finally achieve the equation for the S_a.

$$S_a = \sum_{n=1}^{\infty} Q_n = \frac{\mu}{\mu-\lambda}[1 + \frac{\lambda}{\delta} + \frac{\lambda^2}{(\lambda+\gamma)\gamma}]Q_1$$ Equation 3-19 or Equation a-45

Now, we have used Q_1 to represent the probabilities of all other states. If Q_1 is known, we can derive the probabilities of other states using the equations before. When we set the total probability of all states to 1 ($S_i + S_a + S_{wu} + S_{sd} = 1$), we can derive the value of Q_1.

$$Q_1 = \frac{1}{\frac{\mu}{\gamma} + \frac{\mu\gamma}{\lambda(\lambda+\gamma)} + \frac{\mu}{\delta} + \frac{\mu}{\mu-\lambda}[1 + \frac{\lambda}{\delta} + \frac{\lambda^2}{(\lambda+\gamma)\gamma}]}$$ Equation 3-20 or Equation a-46

Appendix II: Analytical Solution Derivation for the A&F Policy in an On-Off DPM System

In this section, we use Q_1 (the probability of (Active) state 1 in Figure 3-8) to represent the probability of other states.

The probabilities of shutdown states

The relationship between the shutdown states $Q_{sd(n)}$ and Q_1 in the A&F policy keeps the same as its greedy counterpart. Therefore, we have:

$$Q_{sd(n)} = \frac{\mu}{\lambda+\gamma}\left(\frac{\lambda}{\lambda+\gamma}\right)^n Q_1$$ Equation 3-2 or Equation a-4

$$S_{sd} = \frac{\mu}{\gamma} Q_1$$ Equation 3-16 or Equation a-5

The probabilities of inactive states

The relationship between Q_1 and Q_0 is given in Equation 3-13 or Equation a-6.

$$Q_{0*} = \frac{\gamma}{\lambda}\frac{\mu}{\lambda+\gamma} Q_1$$ Equation 3-13 or Equation a-6

The probabilities of the rest inactive states Q_{n*} can be expressed by the probability of its left neighbour $Q_{(n-1)*}$ and that of the corresponding shutdown state $Q_{sd(n)}$ in.

$$\lambda Q_{n*} = \lambda Q_{(n-1)*} + \gamma Q_{sd(n)} \quad (1<n<N)$$ Equation a-47

When we integrate Equation 3-2 or Equation a-4 and Equation a-6 into Equation a-47, we can derive the general expression for the inactive states.

$$Q_{n*} = \frac{\gamma Q_{sd0}}{\lambda}\sum_{k=0}^{n}\left(\frac{\lambda}{\lambda+\gamma}\right)^k = \frac{\mu}{\lambda}\left[1-\left(\frac{\lambda}{\lambda+\gamma}\right)^{n+1}\right]Q_1 \quad (n<N)$$

Equation 3-21 or Equation a-48

The sum of all inactive states (S_i) is given in the following equation.

$$S_i = \sum_{n=0}^{N-1} Q_{n*} = \frac{\mu}{\lambda}[N - \frac{\lambda}{\gamma}(1-(\frac{\lambda}{\lambda+\gamma})^N)]Q_1$$ Equation 3-25 or Equation a-49

When N=1,Equation a-48 and Equation a-49 become their counterpart Equation a-6 (Because there is only one inactive state, S_i=Q_{0*}) in the greedy policy. This characteristic can also be found in the following state groups.

The probabilities of wakeup states

As explained before, the first wakeup state changes from wu1 in the greedy policy to wu(N) in the A&F policy. However, the basic relationship among wakeup states does not change. We have:

$$\gamma Q_{sd(N)} + \lambda Q_{(N-1)*} = (\lambda+\delta)Q_{wu(N)} \quad (n=N)$$ Equation a-50

$$\gamma Q_{sd(n)} + \lambda Q_{wu(n-1)} = (\lambda+\delta)Q_{wu(n)} \quad (n>N-1)$$ Equation a-51

Using the general expressions of shutdown and inactive state groups, we can derive the general expression of the wakeup states in Equation a-52.

$$Q_{wu(n)} = \frac{\gamma Q_{sd(N)}}{\lambda+\delta}\sum_{k=0}^{n-N}[(\frac{\lambda}{\lambda+\gamma})^k(\frac{\lambda}{\lambda+\delta})^{n-N-k}] + (\frac{\lambda}{\lambda+\delta})^{n-N+1}Q_{(N-1)*} \quad (n>N-1)$$

Equation 3-22 or Equation a-52

The sum of the probabilities of the wakeup state S_{wu} can be derived accordingly. For more detailed steps, one can seek the derivation from Equation a-12 to Equation a-19 in the greedy policy. We simply give the result here.

$$S_{wu} = \frac{\mu}{\delta}Q_1$$ Equation 3-18 or Equation a-19

The probabilities of active states

The active state group can be divided into two sub-groups. The probabilities of active states from 1 to (N-1) are only determined by the probability of their neighbours. It is because the SP is not activated when the accumulation in the TQ is not enough. Therefore, we have:

$$Q_n = \sum_{k=1}^{n} (\frac{\lambda}{\mu})^{(k-1)} Q_1 \qquad (n \le N)$$ Equation 3-23 or Equation a-53

For active states from N, their probabilities are not only influenced by the probabilities of their neighbour states, but also affected by the probability of the connected wakeup state. The relationship between these states is given in Equation a-54.

$$\lambda Q_{(n-1)} + \mu Q_{(n+1)} + \delta Q_{wu(n)} = (\mu + \lambda) Q_n$$ Equation a-54

The general expression for these states is given in Equation a-55.

$$Q_n = \sum_{k=1}^{n} (\frac{\lambda}{\mu})^{k-1} Q_1 - \frac{\delta}{\mu} [\sum_{k=N}^{n-1} \sum_{s=k}^{n-1} (\frac{\lambda}{\mu})^{n-s} Q_{wu(k)}] \quad (n > N)$$ Equation 3-24 or Equation a-55

In order to derive the sum of the probabilities of all active states (S_a), we first define

$$X_1 = \sum_{n=1}^{\infty} \sum_{k=1}^{n} (\frac{\lambda}{\mu})^{k-1} Q_1$$ Equation a-56

$$X_2 = -\frac{\delta}{\mu} [\sum_{n=N}^{\infty} \sum_{k=N}^{n-1} \sum_{s=k}^{n-1} (\frac{\lambda}{\mu})^{n-s} Q_{wu(k)}]$$ Equation a-57

So we have

$S_a = X_1 + X_2$ Equation a-58

The calculation of X_1 is given in detail in its counterpart of the greedy policy. One can seek the derivation from Equation a-25 to Equation a-29 for detail. Here we only give the final result (T is the number of all active states under calculation).

$$X_1 = [\frac{\mu T}{\mu - \lambda} - \frac{\lambda \mu}{(\mu - \lambda)^2}] Q_1$$ Equation a-29

If we rewrite the X_2 serial as a matrix by n and k, we have

$X_2 =$

$Q_{wu(N)}$

$$\boxed{\begin{array}{llll}
(1+\frac{\lambda}{\mu})Q_{wu(N)} & Q_{wu(N+1)} & & \\
(1+\frac{\lambda}{\mu}+(\frac{\lambda}{\mu})^2)Q_{wu(N)} & (1+\frac{\lambda}{\mu})Q_{wu(N+1)} & Q_{wu(N+2)} & \\
\dots & \dots & \dots & \dots
\end{array}}$$

This matrix is quite similar to its counterpart in the greedy policy, but has quite different rank. Starting from wu(*N*), the rank of X_2 matrix shrinks to *T*-*N* because the calculation of the first *N* active states do not involve wakeup states. The value of the matrix changes accordingly. For example, the sum of the first column becomes

$$Y_1 = [\frac{\mu(T-N)}{\mu-\lambda} - \frac{\lambda\mu}{(\mu-\lambda)^2}]Q_{wu(N)} \qquad \text{Equation a-59}$$

The result of X_2 is given in Equation a-60. One can seek the derivation from Equation a-32 to Equation a-44 in the greedy policy for detail.

$$X_2 = \frac{\mu}{\mu-\lambda}[-(T-N) + \frac{\lambda}{\mu-\lambda} + \frac{\lambda}{\delta} + \frac{\lambda^{(N+1)}}{\gamma(\lambda+\delta)^N}]Q_1 \qquad \text{Equation a-60}$$

The value of S_a is given in Equation a-61.

$$S_a = X_1 + X_2 = \frac{\mu}{\mu-\lambda}[N + \frac{\lambda}{\delta} + \frac{\lambda^{N+1}}{(\lambda+\gamma)^N\gamma}]Q_1 \qquad \text{Equation 3-26 or Equation a-61}$$

Appendix III: Analytical Solution Derivation for a DPM System with multiple inactive modes

In this section, the probability of the idle state (Q_{idle}) in Figure 3-15 is used to represent the probability of the remaining states.

The probabilities of shutdown states

Q_{sd0}, as the probability of state sd0 is only determined by Q_{idle}. Therefore, we have:

$$Q_{sd0} = \frac{\varepsilon}{\lambda + \gamma} Q_{idle}$$ Equation a-62

The probability of the other shutdown states sd(n) is determined by the probability of its left neighbour $Q_{sd(n\text{-}1)}$. Using the iteration method, we can derive the general expression of the shutdown states and the sum of the probabilities (S_{sd}) in Equation a-63 and Equation a-64 respectively.

$$Q_{sd(n)} = \frac{\varepsilon}{\lambda + \gamma} \left(\frac{\lambda}{\lambda + \gamma}\right)^n Q_{idle}$$ Equation 3-27 or Equation a-63

$$S_{sd} = \sum_{n=0}^{\infty} P_{sd(n)} = \frac{\varepsilon}{\gamma} Q_{idle}$$ Equation 3-33 or Equation a-64

The two equations are similar to their counterparts in the greedy or A&F polices of the on-off DPM system, with the change of μ to ε.

The probability of sleep states

With more than one inactive modes involved in the DPM system, states 0* to (N-1)* cannot be called as inactive states as in the on-off DPM system. We rename these states as sleep states because the SP is in the sleep mode when it is in any of these states.

In the current case, the relationship between Q_{sd0} and the probability of the sleep states are the same as that in the on-off DPM system when either the greedy or the A&F policy is implemented. Therefore, the general expressions for the probability of sleep states and the sum of the probabilities (S_i) are given inEquation 3-28 or Equation a-65 and Equation 3-34 or Equation a-66 respectively.

$$Q_{n*} = \frac{\gamma}{\lambda}\sum_{k=0}^{n} Q_{sd(k)} = \frac{\varepsilon}{\lambda}[1-(\frac{\lambda}{\lambda+\gamma})^{n+1}]Q_{idle} \quad (n \le N)$$ Equation 3-28 or Equation a-65

$$S_i = \sum_{n=0}^{N} Q_{n*} = \{\frac{N\varepsilon}{\lambda} - \frac{\varepsilon}{\gamma}[1-(\frac{\lambda}{\lambda+\gamma})^{N}]\}Q_{idle}$$ Equation 3-34 or Equation a-66

The probability of wakeup states

Similar as the sleep states, the relationship between the probability of wakeup states and shutdown/sleep states are the same as that in the on-off DPM system when either the greedy or the A&F policy is implemented. Therefore, the general expressions for the probability of wakeup states and the sum of the probabilities (S_{wu}) are given in Equation 3-24 or Equation a-55 andEquation 3-35 or Equation a-67 respectively. The detail of S_{wu} calculation can be found in the derivation from Equation a-12 toEquation 3-18 or Equation a-19 in Section 3.2.1.

$$Q_{wu(n)} = \frac{\gamma Q_{sd(N)}}{\lambda+\delta}\sum_{k=0}^{n-N}[(\frac{\lambda}{\lambda+\gamma})^{k}(\frac{\lambda}{\lambda+\delta})^{n-N-k}] + (\frac{\lambda}{\lambda+\delta})^{n-N+1}Q_{(N-1)*}$$ Equation 3-24 or Equation a-55

$$S_{wu} = \frac{\varepsilon}{\delta}Q_{idle}$$ Equation 3-35 or Equation a-67

The probability of turn-off states

The derivation of the analytical solution of the turn-off states is similar to the derivation of the solution of shutdown states. The probability of state toff0 (Q_{toff0}) is only determined by Q_{idle} (Equation a-68).

$$Q_{toff0} = \frac{\lambda+\varepsilon}{\alpha} Q_{idle}$$ Equation a-68

The probability of turn-off state toff(*n*) is determined by its left neighbor toff(*n*-1). Using the iteration method, we can derive the general expression of the turn-off states and the sum of the probabilities (S_{toff}) inEquation 3-29 or Equation a-69 and Equation 3-23 or Equation a-70 respectively.

$$Q_{toff(n)} = \frac{\lambda+\varepsilon}{\alpha}\left(\frac{\lambda}{\lambda+\alpha}\right)^n Q_{idle}$$ Equation 3-29 or Equation a-69

$$S_{toff} = \frac{(\lambda+\varepsilon)(\lambda+\alpha)}{\alpha^2} Q_{idle}$$ Equation 3-23 or Equation a-70

The probabilities of turn-on states

The probability of state ton(*n*) is determined by its left neighbor ton(*n*-1) and the corresponding turn-off state toff(*n*). Therefore, we have:

$$Q_{ton1} = \frac{\lambda}{\lambda+\beta} Q_{idle} + \frac{\alpha}{\lambda+\beta} Q_{toff1}$$ Equation a-71

$$Q_{ton(n)} = \frac{\lambda}{\lambda+\beta} Q_{ton(n-1)} + \frac{\alpha}{\lambda+\beta} Q_{toff(n)}$$ Equation a-72

Using the iteration method, we can derive the general expression of the turn-on states and the sum of the probabilities (S_{ton}) in Equationa-73 and Equation a-74 respectively. The calculation of S_{ton} can be referred to the calculation of S_{wu} in the greedy policy of the on-off DPM system for detail.

$$Q_{ton(n)} = \frac{\alpha Q_{toff1}}{\lambda+\beta}\sum_{k=0}^{n-1}\left[\left(\frac{\lambda}{\lambda+\beta}\right)^k\left(\frac{\lambda}{\lambda+\alpha}\right)^{n-k-1}\right] + \left(\frac{\lambda}{\lambda+\beta}\right)^n Q_{idle}$$ Equation 3-30 or Equation a-73

$$S_{ton} = \frac{\lambda}{\beta}\frac{\lambda+\alpha+\varepsilon}{\alpha} Q_{idle}$$ Equation 3-37 or Equation a-74

The probabilities of active states

As shown in Figure 3-15, the probabilities of active state 1 to *N*-1 are influenced by the probability of their neighbour active states and that of the corresponding turn-on states. Therefore, we have

$$Q_1 = \frac{\lambda+\alpha}{\mu} Q_{toff0}$$ Equation a-75

$$Q_n = (1+\frac{\lambda}{\mu})Q_{(n-1)} - \frac{\beta}{\mu} Q_{ton(n-1)}$$ Equation a-76

Using the iteration, the general expression of Q_n ($n \le N$) is given in Equation a-77.

$$Q_n = \sum_{k=0}^{n-1} (\frac{\lambda}{\mu})^k Q_1 - \frac{\beta}{\mu} \sum_{k=1}^{n-1} \sum_{s=0}^{k} (\frac{\lambda}{\mu})^{k-s-1} Q_{ton(k)} \quad (n \le N)$$ Equation 3-31 or Equation a-77

For $n>N$, Q_n is also affected by $Q_{wu(n)}$, therefore we have:

$$Q_n = \sum_{k=0}^{n-1} (\frac{\lambda}{\mu})^k Q_1 - \frac{\beta}{\mu} \sum_{k=1}^{n-1} \sum_{s=0}^{k} (\frac{\lambda}{\mu})^{k-s-1} Q_{ton(k)} - \frac{\delta}{\mu} \sum_{k=N}^{n-1} \sum_{s=0}^{k} (\frac{\lambda}{\mu})^{k-s-1} Q_{wu(k)} \quad (\text{n}>N)$$
Equation 3-32 or Equation a-78

In order to calculate S_a, we first regroup all components according to Q_1, $Q_{ton(n)}$ and $Q_{wu(n)}$. Therefore, we have:

$$X_1 = \sum_{n=1}^{\infty} \sum_{k=0}^{n-1} (\frac{\lambda}{\mu})^k Q_1$$ Equation a-79

$$X_2 = -\frac{\beta}{\mu} \sum_{n=2}^{\infty} \sum_{k=1}^{n-1} \sum_{s=0}^{k} (\frac{\lambda}{\mu})^{k-s-1} Q_{ton(k)}$$ Equation a-80

$$X_3 = -\frac{\delta}{\mu} \sum_{n=N+1}^{\infty} \sum_{k=N}^{n-1} \sum_{s=0}^{k} (\frac{\lambda}{\mu})^{k-s-1} Q_{wu(k)}$$ Equation a-81

$$S_a = X_1 + X_2 + X_3$$ Equation a-82

Suppose the number of the active states under calculation in Equation a-82 is T, the value of X_1 can be easily calculated in Equation a-83.

$$X_1 = [\frac{\mu T}{\mu-\lambda} - \frac{\lambda\mu}{(\mu-\lambda)^2}]Q_1$$ Equation a-83

The calculation of X_2 and X_3 can refer to the calculation of X_2 in Equation a-30 to Equation a-44 in Section 3.2.1. Now we simply give the results here.

$$X_2 = -\frac{Q_{idle}}{\mu-\lambda}\{\frac{\lambda(\lambda+\alpha+\varepsilon)}{\alpha}(T-1) - \lambda^2[\frac{1}{\beta} + \frac{\lambda+\alpha+\varepsilon}{\alpha(\mu-\lambda)} + \frac{\lambda+\varepsilon}{\alpha^2} + \frac{\lambda+\varepsilon}{\alpha\beta}]\}$$ Equation a-84

$$X_3 = -\frac{\varepsilon}{\mu-\lambda}[(T-1) - \frac{\lambda}{\delta} - \frac{\lambda}{\gamma}(\frac{\lambda}{\lambda+\gamma})^N - \frac{\lambda}{\mu-\lambda}]Q_{idle}$$ Equation a-85

When we integrate Equation a-83, Equation a-84 and Equation a-85 into Equation a-82, we can achieve the value of S_a in Equation 3-38 or Equation a-86.

$$S_a = \{\frac{\lambda^2}{\mu-\lambda}[\frac{1}{\beta}+\frac{\lambda+\alpha+\varepsilon}{\alpha(\mu-\lambda)}+\frac{\lambda+\varepsilon}{\alpha^2}+\frac{\lambda+\varepsilon}{\alpha\beta}]+\frac{\lambda(\lambda+\alpha+\varepsilon)}{\alpha(\mu-\lambda)}-\frac{\lambda(\lambda+\alpha)(\lambda+\varepsilon)}{\alpha(\mu-\lambda)^2}$$
$$+\frac{\varepsilon}{\mu-\lambda}[\frac{\lambda}{\delta}+\frac{\mu}{\mu-\lambda}+\frac{\lambda}{\gamma}(\frac{\lambda}{\lambda+\gamma})^N]\}Q_{idle}$$

Equation 3-38 or Equation a-86

Appendix IV: Analytical Solution Derivation for an DPM System with Multiple Active Modes

When (Active) state 1 is chosen as the delegate state, we use Q_1 to represent the probability of the rest states. The derivation of the analytical solution for the shutdown, wakeup and inactive states are similar to the derivation in Appendix I and the only change is to use μ_L to replace μ.

The probabilities of active states

In this section, we try to derive the probabilities of active states in Figure 3-20.

For active states from 1 to H-1, Equation 3-42 or Equation a-87 gives the general expression for the probabilities of these states.

$$Q_n = \sum_{n=1}^{H-1} (\frac{\lambda}{\mu_L})^{n-1} Q_1 \ (n<H) \qquad \text{Equation 3-42 or Equation a-87}$$

The sum of these states (X_1) can be achieved in Equation a-88.

$$X_1 = \{\frac{\mu_L (H-1)}{\mu_L - \lambda} - \frac{\lambda \mu_L}{(\mu_L - \lambda)^2}[1-(\frac{\lambda}{\mu_L})^{H-1}]\} Q_1 \qquad \text{Equation a-88}$$

Q_H, as the probability of the state H, can be represented by $Q_{H\text{-}1}$ and $Q_{H\text{-}2}$ (Equation 3-43 or Equation a-89).

$$Q_H = (\frac{\lambda + \mu_L}{\mu_H}) Q_{H-1} - \frac{\lambda}{\mu_H} Q_{H-2} \qquad \text{Equation 3-43 or Equation a-89}$$

For active states from H+1 to N, we can use the combination of Q_H and $Q_{H\text{-}1}$ to represent their probabilities.

$$Q_n = \sum_{k=0}^{n-H} (\frac{\lambda}{\mu_H})^k Q_H - \frac{\lambda}{\mu_H} \sum_{k=0}^{n-H-1} (\frac{\lambda}{\mu_H})^k Q_{H-1} \qquad (H<n\leq N)$$

Equation 3-44 or Equation a-90

For the rest active states, their probabilities can be expressed by the combination of Q_H, $Q_{H\text{-}1}$ and the corresponding wakeup states.

$$Q_n = \sum_{k=0}^{n-H}(\frac{\lambda}{\mu_H})^k Q_H - \frac{\lambda}{\mu_H}\sum_{k=0}^{n-H-1}(\frac{\lambda}{\mu_H})^k Q_{H-1} - \frac{\delta}{\mu_H}\sum_{k=N}^{n-1}\sum_{s=k}^{n-1}(\frac{\lambda}{\mu_H})^{n-s} Q_{wu(k)} \quad (n>N)$$

Equation 3-45 or Equation a-91

In order to derive the analytical solution of S_a, we reconstruct the calculation components in the following equations.

$$X_2 = \sum_{n=H}^{\infty}\sum_{k=0}^{n-H}(\frac{\lambda}{\mu_H})^k Q_H$$ Equation a-92

$$X_3 = -\frac{\lambda}{\mu_H}\sum_{n=H+1}^{\infty}\sum_{k=0}^{n-H-1}(\frac{\lambda}{\mu_H})^k Q_{H-1}$$ Equation a-93

$$X_4 = -\frac{\delta}{\mu_H}\sum_{n=N}^{\infty}\sum_{k=0}^{n-N}\sum_{s=k}^{n-N}(\frac{\lambda}{\mu_H})^s Q_{wu(k)}$$ Equation a-94

$$S_a = X_1 + X_2 + X_3 + X_4$$ Equation a-95

Suppose the number of active states whose index is no smaller than H is T and the value of X_2 and X_3 are easily calculated in Equation a-96 and Equation a-97 respectively.

$$X_2 = \{\frac{\mu_H(T+N-H)}{\mu_H-\lambda} - \frac{\lambda\mu_H}{(\mu_H-\lambda)^2}\}Q_H$$ Equation a-96

$$X_3 = -\frac{\lambda}{\mu_H}\{\frac{\mu_H(T+N-H-1)}{\mu_H-\lambda} - \frac{\lambda\mu_H}{(\mu_H-\lambda)^2}\}Q_{H-1}$$ Equation a-97

If we integrate Equation a-87 and Equation a-89 into Equation a-96 and Equation a-97, we can derive:

$$X_2 + X_3 = [\frac{\lambda\mu_L(1-(\frac{\lambda}{\mu_L})^{H-1})}{(\mu_H-\lambda)(\mu_L-\lambda)} + \frac{\mu_L(T+N-H)}{(\mu_H-\lambda)} - \frac{\lambda\mu_L}{(\mu_H-\lambda)^2}]Q_1$$ Equation a-98

The calculation of X_4 can follow the steps shown in Equation a-30 to Equation a-43. One thing needs to be paid attention to is the number of states under calculation of X_4 is T-N. The result is given in Equation a-99.

$$X_4 = -\frac{\mu_L}{\mu_H - \lambda}[T - 1 - \frac{\lambda}{\delta} - \frac{\lambda^{(N+1)}}{\gamma(\lambda+\gamma)^N} + \frac{\lambda\mu_L}{(\mu_H - \lambda)^2}]Q_1$$ Equation a-99

When we integrate Equation a-88, Equation a-98 and Equation a-99 into Equation a-95, the value of S_a is given in Equation a-100.

$$S_a = \{\frac{\mu_L(H-1)}{\mu_L - \lambda} - \frac{\lambda\mu_L}{(\mu_L-\lambda)^2}[1-(\frac{\lambda}{\mu_L})^{H-1}] + \frac{\lambda\mu_L[1-(\frac{\lambda}{\mu_L})^{H-1}]}{(\mu_H-\lambda)(\mu_L-\lambda)} + \frac{\mu_L}{\mu_H-\lambda}[N-H+1+\frac{\lambda}{\sigma}+\frac{\lambda^{(N+1)}}{\gamma(\lambda+\gamma)^N}]\}Q_1$$

($\lambda \neq \mu_L$) Equation 3-49 or Equation a-100

Given H=1 and $\mu_L = \mu_H = \mu$, the Markov model in Figure 3-20 becomes Figure 3-8, which stands for an simple on-off DPM system, and the equations for the analytical solutions in this section such as Equation 3-49becomes their counterparts like Equation 3-26.

In previous sections, $\lambda < \mu$ serves as the basic requirement about the rate of incoming events. Otherwise the SP may never finish the execution of tasks. When we deal with DPM systems with multiple active modes, the system may provide a serial execution rate μ_1, μ_2, … μ_r (suppose r is the number of all active modes, and $\mu_1 \leq \mu_2 \leq \ldots \leq \mu_r$). In this case, the requirement of the rate of incoming events becomes $\lambda < \mu_r$, and for the model in Figure 3-21, we have $\lambda < \mu_H$. Given $\lambda \rightarrow \mu_L$, the expression of the analytical solution is provided by or Equation a-101.

$$S_a = \{\frac{(H-1)H}{2} + \frac{\lambda(H-1)}{\mu_H - \lambda} + \frac{\mu_L}{\mu_H-\lambda}[N-H+1+\frac{\lambda}{\sigma}+\frac{\lambda^{(N+1)}}{\gamma(\lambda+\gamma)^N}\}Q_1$$

($\lambda = \mu_L$) Equation 3-50 or Equation a-101

Appendix V: State Space Report for the Event Handler

This state space report is about the CPN model of Event Handler in Figure 4-6:

```
Statistics
----------------------------------------------------------------------
 Occurrence Graph                     Scc Graph
     Nodes:  31                         Nodes:  6
     Arcs:   44                         Arcs:   5
     Secs:   0                          Secs:   0
     Status: Full

 Boundedness Properties
----------------------------------------------------------------------
  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
----------------------------------------------------------------------
  Home Markings:  None

 Liveness Properties
----------------------------------------------------------------------
  Dead Markings:  None
  Dead Transitions Instances: None
  Live Transitions Instances: Matrix'env1
```

Appendix VI: State Space Report for the Power Manager

This state space report is about the CPN model of the Power Manager.

```
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'Me1 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'Me1 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'Me1 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 has no confliction with that of later validated ready signals.

The Best Upper Multi-set Bound of the acc place indicates the token value in this place is no more than *N* (*N*=2 is the accumulation limit) which means the activation signal is 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 the Task Manager

This state space report is about the CPN model of the Task Manager

```
Statistics
---------------------------------------------------------------------
 Occurrence Graph                  Scc 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
```

Appendix VIII: State Space Report for Output and Interface

This state space report is about the CPN model of Output Control and Interface

```
Statistics
----------------------------------------------------------------------
 Occurrence Graph                   Scc Graph
     Nodes:  46                         Nodes:  1
     Arcs:   82                         Arcs:   0
     Secs:   0                          Secs:   0
     Status: Full

Boundedness Properties
----------------------------------------------------------------------
  Best Integers Bounds    Upper      Lower
  OutCt'Ch3 1             1          0
  OutCt'Complete 1        1          0
  OutCt'Current 1         1          1
  OutCt'DIN1 1            1          0
  OutCt'DIN2 1            1          0
  OutCt'DOUT 1            1          0
  OutCt'LTask 1           1          0
  OutCt'Mtask1 1          1          0
  OutCt'Mtask2 1          1          0
  OutCt'OCh3 1            1          0
  OutCt'RQ 1              1          0
  OutCt'Rdy 1             1          1
  OutCt'Read 1            1          1
  OutCt'STEPSleep 1       1          1
  OutCt'Sd 1              1          0
  OutCt'SearchEn 1        1          0
  OutCt'Sleep 1           1          1
  OutCt'Wu 1              1          0
  OutCt'activation 1      1          1

  Best Upper Multi-set Bounds
 OutCt'Activation   1 1`1++1`0     OutCt'Ch3 1        1`(2,"DATA1")
 OutCt'Current 1    1`0++1`1       OutCt'DIN1 1       1`"DATA2"
 OutCt'DIN2 1       1`"DATA1"      OutCt'LTask 1      1`0++1`1++1`2
 OutCt'DOUT 1       1`"DATA1"++1`"DATA2"
 OutCt'LoadEn 1     1`1            OutCt'Mtask1 1     1`1
 OutCt'Mtask2 1     1`2            OutCt'OCh3 1       1`(3,"DATA1")
 OutCt'RQ 1         1`1++1`2       OutCt'Rdy 1        1`0++1`1++1`2
 OutCt'Sleep 1      1`0++1`1       OutCt'Complete 1   1`1
 OutCt'Read 1       1`0++1`1       OutCt'STEPSleep 1  1`0++1`1
 OutCt'Sd 1         1`1            OutCt'Wu 1         1`1
 OutCt'Sleep 1      1`0++1`1

  Best Lower Multi-set Bounds
 OutCt'Activation 1 empty          OutCt'Ch3 1        empty
 OutCt'Current 1    empty          OutCt'DIN1 1       empty
 OutCt'DIN2 1       empty          OutCt'LTask 1      empty
 OutCt'DOUT 1       empty          OutCt'Sleep 1      empty
 OutCt'LoadEn 1     empty          OutCt'Mtask1 1     empty
 OutCt'Mtask2 1     empty          OutCt'OCh3 1       empty
 OutCt'RQ 1         empty          OutCt'Rdy 1        empty
 OutCt'Sleep 1      empty           OutCt'Complete 1   empty
 OutCt'Read 1       empty           OutCt'STEPSleep 1  empty
 OutCt'Sd 1         empty           OutCt'Wu 1         empty

 Home Properties
---------------------------------------------------------------------
  Home Markings:  All

 Liveness Properties
---------------------------------------------------------------------
  Dead Markings:  None
  Dead Transitions Instances: None
  Live Transitions Instances: All
```

Appendix IX: The S-function Code of OS Subsystem

```
function [sys,x0,str,ts] = sfundsc1(t,x,u,flag)

switch flag,

  %%%%%%%%%%
  % Initialization  %
  %%%%%%%%%%
  case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;

  %%%%%%%
  % Update %
  %%%%%%%
  case 2,
    sys = mdlUpdate(t,x,u);

  %%%%%%%
  % Output %
  %%%%%%%
  case 3,
    sys = mdlOutputs(t,x,u);

  %%%%%%%%
  % Terminate%
  %%%%%%%%
  case 9,
    sys = [];

  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end

%end sfundsc1

%
%=============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes

sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 3; % 3 states Sleep, Read and Now to be kept;
sizes.NumOutputs      = 2; % 2 output named as Sleep, Read;
sizes.NumInputs        = 3; % 3 input Wakeup Shutdown and Current;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

%initialization
x0(1) = 1;    % the initial value of Sleep
x0(2) = 0;    % the initial value of Read
x0(3) = -1;   % the initial value of Now

str = [];
ts  = [0, 0]
% end mdlInitializeSizes

%
%=============================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step requirements.
%=============================================================
%
%system status

function sys = mdlUpdate(t,x,u)

%name the three inputs from vector u
Wakeup    = u(1);
Shutdown = u(2);
Current    = u(3);

%name the five states from vector x
Sleep  = x(1);
Read   = x(2);
Now    = x(3);

%the size of playboard
Board  = 100;

%Initial patching the playboard I (50*50) which is  controlled by VSB I in black
%color

if Now==-1
   H=[0   0           Board/2   Board/2];
   V=[0   Board/2  Board/2   0      ];
   patch(H,V,'k');
   drawnow;
end
```

```
%If the IP core is sleeping and the Wakeup signal is captured, start wakeup
if (Sleep==1 & Wakeup==1)
   Now = 0;
end

%The wakeup processing is simulated by patching one stripe of playboard I
%everytime by white color. State Now is used to record %the processing degree
if (Now<=(Board/2)-1 & Now>=0 & Sleep==1)
    H=[0        0          Board/2   Board/2];
    V=[Now    Now+2   Now+2    Now   ];
    patch(H,V,'w');
    drawnow;
    Now = Now + 2;
end

%The wakeup processing completes when all playboard I is patched by
%white color, the Sleep signal is set to 0 and a Read signal is
%sent out to load new task ID number to the IP core
if Now==(Board/2) & (Sleep==1)
  Sleep = 0;
  Read  = 1;
  Now = Now + 1;
end

%If some task start processing in the IP core, the Current signal
%becomes 1 and the OS will withdraw the Read signal
if(Current==1)
  Read = 0;
end

%When Current becomes 0, it means the current task is completed, and
%the OS needs to load another task to the IP core
if(Current==0 && Sleep==0)
   Read = 1;
end

%When a Shutdown signal is captured, the OS starts the shutdown
%execution state Shutting becomes 1 to mark the shutdown is in %processing
if(Sleep==0 && Shutdown==1 && Now>Board/2)
   Now = (Board/2);
   Read = 0;
end

%The shutdown processing is simulated by patching one stripe of
%playboard I everytime by black color
if (Now<=(Board/2) && Now>=2 && Sleeping==1)
    H=[0        0          Board/2   Board/2];
    V=[Now    Now-2   Now-2     Now    ];
    patch(H,V,'k');
    drawnow;

    Now = Now - 2;
end

%When all playboard I have patched in black, the shutdown processing is
%completed. The Sleep signal becomes 1
if Now == 0 && Sleep==0
   Sleep = 1;
   Now = Board;
end

%Updating the system states
sys = [Sleep R Now];

%end mdlUpdate

%=============================================================
% mdlOutputs
% Return the output vector for the S-function
%=============================================================
%
function sys = mdlOutputs(t,x,u)
%Output Sleep and Read signals
sys = [x(1) x(2)]';
%end mdlOutputs
```

Appendix X: The S-Function Code of Task4 in the Ball Game

```
function [sys,x0,str,ts] = sfundsc1(t,x,u,flag)

Task    = [0 0 0 1];    % the one hot code for task4
Width  = [2, 4, 6, 8]; % the width of all four balls
Speed  = [2, 4, 6, 8]; % the moving speed of all four balls

switch flag,

  %%%%%%%%%%%
  % Initialization  %
  %%%%%%%%%%%
  case 0,
   [sys,x0,str,ts]=mdlInitializeSizes;

  %%%%%%%
  % Update %
  %%%%%%%
  case 2,
    sys = mdlUpdate(t,x,u,Task, Width, Speed);

  %%%%%%%
  % Output %
  %%%%%%%
  case 3,
    sys = mdlOutputs(t,x,u);

  %%%%%%%%%%
  % Terminate  %
  %%%%%%%%%%
  case 9,
    sys = [];

  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end

%end sfundsc1

%
%=============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes
```

```
sizes = simsizes;

sizes.NumContStates    = 0;
sizes.NumDiscStates    = 46;
sizes.NumOutputs       = 31;
sizes.NumInputs        = 30;
sizes.DirFeedthrough   = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

x0  = zeros(1,46);

for i=43:45
   x0(i) = -1;
end

str = [];
%ts  = [-1 0]; % Inherited sample time
ts  = [0, 0]
% end mdlInitializeSizes

%
%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
%
%system status

function sys = mdlUpdate(t,x,u,Task, Wide, Speed)

%specialize the input vector u
   AddressIn = u(1:4);
   DataIn    = u(5:28);
   Rdy       = u(29);
   Ack       = u(30);

%specialize the state vector x
   Address = [ x(1),   x(2), x(3), x(4)];
   Data1   = [ x(5),   x(6),  Wide(1), Speed(1), x(7)];
   Data2   = [ x(8),   x(9),  Wide(2), Speed(2), x(10)];
   Data3   = [ x(11), x(12), Wide(3), Speed(3), x(13)];
   Data4   = [ x(14), x(15), Wide(4), Speed(4), x(16)];

%DataOut[1:8] represents PosX, DataOut[9:16] represents PosY,
%DataOut[17:24] represents History
```

```
  DataOut    = x(17:40);
  Read       = x(41);
  Write      = x(42);
  NextData   = x(43);
  Stage      = x(44);
  NextACM = x(45);
  Current    = x(46)

%check if the ID number is matching. If so, set the Current signal to 1,and let the OS
%to withdraw the Read signal
  if(Task*AddressIn'==1)
       Current = 1;
       Stage = 1;
  end

%Load the parameters of four balls from ACM
  if(Stage==1)
     [Current, Data1, Data2, Data3, Data4, R, NextData, NextACM]
    = DataLoad(Data1, Data2, Data3, Data4,DataIn, Rdy, NextData,
     NextACM, Current, R);

     if (NextACM==4)
             Stage  = 2;
        NextData  = -1;
        NextACM = -1;
     end
  end

 %Calculate the new position of the corresponding ball
  if (Stage==2)
     %Load the old position
     PosX    = Data4(1);
     PosY    = Data4(2);
     Width   = Data4(3);
     Speed   = Data4(4);
     History = Data4(5);

     %erase the old mark of the ball
     H=[PosX    PosX             PosX+Width    PosX+Width];
     V=[PosY    PosY+Width    PosY+Width    PosY     ];
     patch(H,V,'w');
     drawnow;

     %update the history
     [History] = UpdateHistory(History);

     %find the new position candidate
     [NPosX, NPosY, NHistory]
```

```
= NextPosition(PosX, PosY, Width, Speed, History);

%check if the new position has collision with Ball 1
OtherX = Data1(1);  OtherY = Data1(2);  OtherW = Data1(3);

[Collapse]= Collision(OtherX, OtherY, OtherW, NPosX, NPosY, Width);

while(Collapse==1)
     History=mod(History+1,4)
     [NPosX, NPosY, NHistory]
     = NextPosition(PosX, PosY, Wide, Speed, History);

     [Collapse] = Collision(OtherX, OtherY, OtherW, NPosX, NPosY, Width);
end

%check if the new position has collision with Ball2
OtherX = Data2(1); OtherY = Data2(2); OtherW = Data2(3);

[Collapse] = Collision(OtherX, OtherY, OtherW, NPosX, NPosY, Width);

while(Collapse==1)
     History=mod(History+1,4)
    [NPosX, NPosY, NHistory]
    = NextPosition(PosX, PosY, Width, Speed, History);

    [Collapse] = Collision(OtherX, OtherY, OtherW, NPosX, NPosY, Width);
end

%check if the new position has collison with Ball3
OtherX = Data3(1); OtherY = Data3(2); OtherW = Data3(3);

[Collapse] = Collision(OtherX, OtherY, OtherW, NPosX, NPosY, Width);

while(Collapse==1)
     History=mod(History+1,4)
    [NPosX, NPosY, NHistory]
    = NextPosition(PosX, PosY, Wide, Speed, History);

    [Collapse] = Collision(OtherX, OtherY, OtherW, NPosX, NPosY, Width);
end

%show the new mark of the ball
H=[NPosX    NPosX           NPosX+Width   NPosX+Width];
V=[NPosY    NPosY+Width    NPosY+Width   NPosY     ];
patch(H,V,'r');
drawnow;

%update the ball parameter
```

```
        Data4(1)= NPosX;
        Data4(2)= NPosY;
        Data4(5)= History;
        Stage = 3;

    end

%Transfer the parameter of the new position of the ball to the ACM
    if (Stage==3)
        %Address[1:2] indicates the VSB's ID
        %and Address[3:4] indicates the ball's ID
        Address = [0 1 0 0];
        [Address, DataOut, W, NextData]
        = DataTransfer(Data4, Ack, NextData, Address, DataOut, W);

        if (NextData== -2)
            Stage = 4;
            Write = 0;
            NextData = -1;
        end
    end

    %Release the usage of the IP Core
    if (Stage==4 && Ack==0)
        Current = 0;
        Write = 1;
        Stage =5;
    end

%When the ID number in the AddressIn is changed, the Write
%signal is withdrawn
    if(Task*AddressIn'~=1)
            Address = [0 0 0 0];
            DataOut = zeros(1,16);
            Write = 0;
            Stage = -1;
            NextData = -1;

    end

Data = [Data1(1),Data1(2),Data1(5), Data2(1),Data2(2),Data2(5),
         Data3(1),Data3(2),Data3(5),  Data4(1),Data4(2),Data4(5),DataOut];

sys = [Address, Data, Read, Write, NextData, Stage, NextACM];

%end mdlUpdate

%
```

```
%==============================================================
% mdlOutputs
% Return the output vector for the S-function
%==============================================================
%
function sys = mdlOutputs(t,x,u)

sys = [x(1:4), x(17:42)]

%end mdlOutputs

%the function is about loading data to the ACM
function [Current, DataOut, W, NextData]
        = DataTransfer(Data4, Ack, NextData, Current, DataOut, W)

  PosX    = Data4(1);
  PosY    = Data4(2);
  Width   = Data4(3);
  Speed   = Data4(4);
  History = Data4(5);

  % turn the position parameter into binary
  DataOut1 = Binary (PosX);
  DataOut2 = Binary (PosY);

% if Current is 1, then output the History Parameter to
% DataOut[17:24]
  if (Current==1)
        DataOut3 = Binary (History);
  % else DataOut[17:24] is composed of the width of the ball as
  %well as the ID number of the ball. This data is used for output
  %control in the STEP
  else
        DataOut4 = Binary (Width);
        DataOut3 = DataOut4(5:8);
        DataOut3 = [DataOut3 0 0 0 1];
  end

  % Load the data to the databus, then enable the Write signal
  if (NextData==-1 && Ack==0)
     DataOut  = [DataOut1, DataOut2, DataOut3];
     Write = 1;
     NextData=0;
  end

  % When Ack from the ACM is recognized, withdraw the write signal
  if (Ack==1 && NextData == 0)
     W = 0;
```

```
      NextData = -2;
   end

    %this function is about conveying integer number to binary
function [Data] = Binary(Original)
    Data = zeros(1, 8);
    i = 7;
    x = Original;

    Data(8) = mod(x,2);
    x = floor(x/2);
    while (x>0)
        Data(i) = mod(x,2);
        x = floor(x/2);
        i = i-1;
    end

%this function is about load data of four balls in sequence from the ACM
function [Address, Data1, Data2, Data3, Data4, R, NextData, NextACM]
         = DataLoad(Data1, Data2, Data3, Data4, DataIn, Rdy,
           NextData, NextACM, Address, R)

   if NextACM ==-1
%Address[3:4]=[0 1] indicates the ACM that data for ball1 is needed
      Address3 = 0;
      Address4 = 1;
      [Address,Data1,R, NextData] = DataCome(DataIn, Data1,
       Address3, Address4, Rdy, NextData, Address, R);
      if NextData == -2
         NextACM = 1;
         NextData = -1;
      end
   end

   if NextACM == 1
 %Address[3:4]=[1 0] indicates the ACM that data for ball2 is needed
      Address3 = 1;
      Address4 = 0;

      [Address,Data2,R, NextData] = DataCome(DataIn, Data2,
       Address3, Address4, Rdy, NextData, Address, R);
      if NextData == -2
         NextACM = 2;
         NextData = -1;
      end
   end

   if NextACM == 2
 %Address[3:4]=[1 1] indicates the ACM that data for ball3 is needed
      Address3 = 1;
      Address4 = 1;
      [Address,Data3,R, NextData] = DataCome(DataIn, Data3,
      Address3, Address4, Rdy, NextData, Address, R);
      if NextData == -2
         NextACM = 3;
         NextData = -1;
      end
  end

  if NextACM == 3
%Address[3:4]=[0 0] indicates the ACM that data for ball4 is needed
      Address3 = 0;
      Address4 = 0;
     [Address,Data4,R, NextData] = DataCome(DataIn, Data4,
     Address3, Address4, Rdy, NextData, Address, R);
      if NextData == -2
         NextACM = 4;
         NextData = -1;
      end
  end

%this function specify the parameters loading of a ball in sequence
function [Address, Data, R, NextData] = DataCome(DataIn, Data,
        Address3, Address4, Rdy, NextData, Address, R)

%When the previous loading is complete (Rdy=0), start the current
      if(NextData == -1 && Rdy == 0)
         Address    = [0 1 Address3, Address4];
         Read       = 1;
         NextData  = 0;
      end

%When Rdy=1, it means the ACM is loading the data into the databus
      if(NextData == 0 && Rdy == 1)
         % Turn the binary information into integer
         PosX = 128*DataIn(1) + 64*DataIn(2) + 32*DataIn(3)
                     + 16*DataIn(4) + 8*DataIn(5) + 4*DataIn(6)
                     + 2*DataIn(7) + DataIn(8);
         PosY = 128*DataIn(9) + 64*DataIn(10) + 32*DataIn(11)
                    + 16*DataIn(12) + 8*DataIn(13) + 4*DataIn(14)
                    + 2*DataIn(15) + DataIn(16);
       History = 128*DataIn(17) + 64*DataIn(18) + 32*DataIn(19)
                    + 16*DataIn(20) + 8*DataIn(21) + 4*DataIn(22)
                    + 2*DataIn(23) + DataIn(24);
    NextData = -2;
      Address = [0, 0, 0, 0];
```

```
        Read = 0;
    end

    Data = [PosX, PosY, Width, Speed, History];

%the function is about updating history
function [NHistory] = UpdateHistory(History)
     s=unifrnd(0, 1);
     Threshold = 0.5;
     if s>=Threshold
        History = floor(unifrnd(0,4));
     else
        NHistory = History;
     end

%the function is to calculate whether two balls have collision
function [Collapse]
    = Collision(OtherX, OtherY, OtherW, NPosX, NPosY, Wide)

     Centre1_X = NPosX+0.5*Wide;
     Centre1_Y = NPosY+0.5*Wide;
     Centre2_X = OtherX+0.5*OtherW;
     Centre2_Y = OtherY+0.5*OtherW;

     Dis_Centres = power((Centre1_X - Centre2_X), 2)
                + power((Centre1_Y - Centre2_Y), 2);

     Dis_Length  = power((Wide+OtherW)*0.5, 2);

     if(Dis_Centres <2*Dis_Length)
         Collapse = 1;
     else
         Collapse = 0;
     end

%the function is to give a new position of the ball
function [NPosX, NPosY, NHistory]
         = NextPosition(PosX, PosY, Width, Speed, History)

  Board = 100;

  switch (History)
     case 0 %move left
        if PosX-Speed>0
           PosX = PosX - Speed;
           NHistory = 0;
        else
           PosX = PosX + Speed;
```

```
            NHistory = 1;
        end

    case 1 %move right
        if PosX+Width+Speed < Board
            PosX = PosX + Speed;
            NHistory = 1;
        else
            PosX = PosX - Speed;
            NHistory = 0;
        end
    case 2 %move up
        if PosY+Width+Speed < Board
            PosY = PosY + Speed;
            NHistory = 2;
        else
            PosY = PosY - Speed;
            NHistory = 3;
        end
    case 3 %move down
        if PosY-Speed > 0
            PosY = PosY - Speed;
            NHistory = 3;
        else
            PosY = PosY + Speed;
            NHistory = 2;
        end

end%end switch
NPosX = PosX;
NPosY = PosY;
```

Bibiliography

[aydi01] H.Aydin, R.Melhem, D.Mosse, P.Mejia-Alvarez, "Determining Optimal Processor Speeds for Periodic Real-time Tasks with Different Power Characteristics", 13th Euromicro Conference on Real-Time Systems, 2001

[alex98] A. Yakovlev and A.M. Koelmans "Petri nets and Digital Hardware Design" Lectures on Petri Nets II: Applications Advances in Petri Nets, Lecture Notes in Computer Science, vol. 1492. Springer-Verlag, pp. 154-236, 1998

[beig08] E.Beigne, F.Clermidy, S.Miermont, P.Vivet, "Dynamic Voltage and Frequency Scaling Architecture for Units Integration within a GALS NoC" ", *NOCS* 2008.

[beni98] L Benini, G de Micheli, "Dynamic Power Management: Design Techniques and CAD Tools" , *Kluwer Academic Publishers Norwell*, USA, 1998.

[beni99] L.Benini, A.Bogliolo, G.A.Paleologo, "Policy Optimization for Dynamic Power Management", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1999

[beni00] L.Benini, A.Bogliolo, G. De Micheli, "A survey of Design Techniques for System-level Dynamic Power Management" *IEEE Transactions on VLSI* June 2000.

[bork99] S.Borkar, "Design Challenges of Technology Scaling", *IEEE Micro* 1999

[broc03] B. Brock, K.Rajamani, "Dynamic Power management for Embedded Systems", *IEEE SOC Conference*, 2003

[buzs04] G.Buzsaki, A.Draguhn, "Neuronal Oscillations in Cortical Networks", *Science*, June 2004.

[byst00] A.Bystrov, D.Kinniment, A.Yakovlev, "Priority Arbiters", *Async* 2000.

[calh03] B.H.Calhoun, F.A.Honore, A.Chandrakasan, "Design Methodology for Fine-Grained Leakage Control in MTCMOS", *International Symposium on Low Power Electronics and Design*, 2003.

[camp04] E.Campbell, Hugo Simpson, "Butler Chip – Concept and Design", *IEE*

Talk, 2004

[chia01] C.-F. Chiasserini, R. Ramesh, “Energy Efficient Battery Management”, *IEEE Journal on Selected Areas in Communications*, Jul 2001.

[chou04] G Choudhury, M Paul, “A batch arrival queue with an additional service channel under N-policy”, *Applied mathematics and Computation*, 2004.

[chun99] E.Chung, L.Benini, A.Bogliolo, and G.De Micheli, “Dynamic Power Management for Nonstationary Service Requests”, *Design and Test in Europe Conf.*, 1999.

[cpnt08] CPN Tools, http://wiki.daimi.au.dk/cpntools/cpntools.wiki

[dasg06] S. Dasgupta, A. Yakovlev, “Modelling and Performance Analysis of GALS architectures”, *SOC* 2006

[gorg08] 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

[henz07] S.Henzler, “Power Management of Digital Circuits in Deep Sub-Micron CMOS Technologies”, Springer 2007.

[hwan97] C.-H. Hwang and A. Wu, “A predictive system shutdown method for energy saving of event-driven computation”, *Proc. Of the ICCAD*, 1997.

[ibmt97] IBM Travelstar 5GS 2.5-Inch 4.4 and 5.1 GB Full Height Disk Drives, http://www.hitachigst.com/tech/techlib.nsf/techdocs/

EBB131F7767F58CE86256E2F007F051D/$file/trav5gs.pdf

[iran03] S.Irani, S.Shukla, R.Gupta, “Online Strategies for Dynamic power Management in Systems with Multiple Power-Saving States”, *ACM Transactions on Embedded Computing Systems*, 2003.

[jaco04] H. M. Jacobson, “Improved Clock-Gating through Transparent. Pipelining”, *ISLPED04*, 2004

[jeju03] R.Jejurikar, C.Pereira, R.K.Gupta, “Leakage Aware Dynamic Voltage Scaling for Real Time Embedded Systems”, *DAC*, 2004

[jens97] Kurt Jensen, “Coloured Petri nets, Basic Concepts, Analysis methods and Practical Use Volume 1”, *Springer* 1997

[jens07] 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.

[kans03] A. Kansal, M. B. Srivastava, “An environmental energy harvesting framework for sensor networks”, *ISLPED’03*, Seoul, Korea.

[karg05] M. Kargahi, A. Movaghar, “A Stochastic DVS-Based Dynamic Power Management for Soft Real-time Systems”, *International Conference on Wireless Networks, Communication and Mobile computing*, 2005.

[kinn07a] D J. Kinniment, C E. Dike, K. Heron, G. Russell and A.Yakovlev. “Measuring Deep Metastability and Its Effect on Synchronizer

Performance", *IEEE Transactions on Very Large Scale Integration Systems* 2007.

[kinn07b] D.J.Kinniment, "Synchronization and Arbitration in Digital Systems", *John Wiley & Sons*, Ltd, 2007.

[klei75] L. Kleinrock "Queuing Systems Volume I: Theory", *John Wiley & Sons*, 1975.

[lu00] Y.Lu, E.Chung, T. Simunic, L. Benini, G. De Micheli "Quantitative Comparison of Power Management Algorithms", *DATE* 2000.

[lu06] H. Lu, Y. Lu, Z. Tang, S. Wang, "SOC Dynamic Power Management using Artificial Neural Network", *ISDA* 2006.

[luci08] C.Lucia, H.Boudewijn, "Quantitative Evaluation in Embedded System Design: Predicting Battery Lifetime in Mobile Devices", *DATE* 2008

[masc87] Joint, IECCA and MUF Committee, The Official Handbook of MASCOT, 1987

[mats08] H.Matsutani, M.Koibuchi, D.Wang, H. Amano, "Adding Slow-Silent Virtual Channels for Low-Power On-chip Networks", *NOCS* 2008

[mihi04] K. Mihic, T. Simunic, G. D. Micheli, "Reliability and Power management of Integrated Systems", *DSD* 2004.

[miln90] R.Milner, M. Tofte, R.Harper, "The definition of Standard ML", *MIT Press*, 1990

[moor02] S.Moore, G.Taylor, R.Mullins, P.Robinson, "Point to Point GALS Interconnect", *ASYNC* 2002.

[nowk02] KJ Nowka, etc, "A 32-bit PowerPC System-on-a-Chip with Support for Dynamic Voltage Scaling and Dynamic Frequency Scaling", *IEEE Journal of Solid-State Circuits*, 2002.

[ren05] Z. Ren, B.H. Krogh, R. Marculescu, "Hierarchical Adaptive Dynamic Power Management", *IEEE Transactions on Computers*, 2005.

[pear04] W.L. Pearn, Y.C.Chang, "Optimal management of the N-policy M/Ek/1 queuing system with a removable service station: a sensitivity investigation", *Computers & Operations* 2004.

[pete81] J.L. Peterson, "Petri Net Theory and the Modeling of Systems", *Prentice Hall PTR Upper Saddle River*, NJ, USA, 1981

[pill01] P. Pillai, K.G.Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems", *18th Symposium on Operating Systems Principles*, 2001.

[qiu99] Q. Qiu, M. Pedram, "Dynamic Power Management Based on Continuous-Time Markov Decision Processes", *DAC* 1999.

[sa11] SA1110 Datasheet, http://www.chipdocs.com/datasheets/datasheet-pdf/Intel/SA1110.html

[shan02] D.Shang, D.Sokolov, N.A.Starodoubtsev, "Asynchronous Circuit

Synthesis by Direct Mapping: Interfacing to Environment", *ASYNC* 2002

[simp90] H.R.Simpson, "Four-slot fully asynchronous communication mechanism", *IEE Proceedings, Computers and Digital Techniques*, 1990.

[simu00] T.Simunic, L.Benini, P.Glynn. G.de Micheli, "Dynamic Power Management for Portable Systems", *the 6th annual international conference on Mobile computing and networking*, Boston, 2000.

[simu01] T.Simunic, L.Benini, P.Glynn. G.de Micheli, "Event Driven Power Management", *IEEE Transactions of CAD*, 2001

[sing06] G.Singh, S.K.Shukla, "Low Power Hardware Synthesis from TRS-based Specifications", *MEMOCODE* 2006.

[sriv96] M. Srivastava, A, Chandrakasan, R. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation", *IEEE Transactions on VLSI Systems*, 1996.

[tan08] Y.Tan, Q.Qiu, "A Framework of Stochastic Power Management using Hidden Markov Model", *DATE* 2008

[thom08] Nigel Thomas, "A PEPA Model of a Threshold Policy Sleeping Server", *UKPEW* 2008

[wang95] K. Wang, H. Huang, "Optimal Control of a Removable Server in an M/Ek/1 Queuing System with Finite Capacity", *Microelectronics Reliability*,1995

[xia02] F. Xia, and I. Clark, "Algorithms for Signal and message Asynchronous Communication mechanisms and their Analysis," in *Fundamenta Informaticae,* Volume 50, Issue 2, 2002

[yadi63] M. Yadin and P. Noar, "Queueing systems with a removable service station", *Operational Research Quarterly*, 4, pp. 393-405, 1963

[yseb07] L. Yseboodt, M. De Nil, J. Huisken, M. Berekovic, Q. Zhao, F. Bouwens, and J. Van Meerbergen, "Ultra-low-power DSP design", http://www.mwee.com/printableArticle/?articleID=201803413

[yuan05] L. Yuan, G. Qu. "Analysis of energy reduction on dynamic voltage scaling-enabled systems", *IEEE Transactions on CAD*, Vol.24, No.12, 2005

[zhua02] S.Zhuang, W.Li, J.Carlsson, K.Palmkyist, L.Wanhammar, "An asynchronous wrapper with novel handshake circuits for GALS systems", *Communications, Circuits and Systems and West Sino Expositions Conference*, 2002.