UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Micro et Nano Electronique**

Arrêté ministériel : 7 août 2006

Présentée par

Hatem Mohamed Zakaria Radwan

Thèse dirigée par **Laurent Fesquet** et
codirigée par **Marc Renaudin**

préparée au sein du **Laboratoire TIMA**
dans **l'École Doctorale « Electronique, Electrotechnique, Automatique et Traitement du Signal »**

Architecture Asynchrone pour L'Efficacité Energétique et L'Amélioration du Rendement en Fabrication dans les Technologies Décananométriques: Application à un Système sur Puce Multi-Cœurs

Thèse soutenue publiquement le **« 24 Février 2011 »**,
devant le jury composé de :

M. Michel ROBERT
Professeur des Universités, LIRMM, Président

M. Olivier SENTIEYS
Professeur des Universités, ENSSAT, Rapporteur

M. Habib MEHREZ
Professeur des Universités, Laboratoire LIP6, Rapporteur

M. Laurent Fesquet
Maitre de Conférences, TIMA, Directeur de thèse

M. Marc Renaudin
Directeur Technique, TIEMPO, Co-encadrant

M. Carlos Canudas-de-Wit
Directeur de Recherche, GIPSA-Lab, CNRS, Membre

M. Mario Diaz Nava
Directeur Technique, STMicroelectronics, Invité

Asynchronous Architecture for Power Efficiency and Yield Enhancement in the Decananometric Technologies: Application to a Multi-Core System-on-Chip

By

Hatem Zakaria

France, 2011

To …

My dear Wife and lovely child, Ali

My Parents, Sisters and Brother

My lovely country Egypt

ACKNOWLEDGMENT

This thesis arose in part out of years of research that has been done within the Concurrent Integrated Systems group (CIS) of the TIMA laboratory in Grenoble. By that time, I have worked with a great number of people whose contribution in assorted ways to the research and the making of the thesis deserved special mention. It is a pleasure to convey my gratitude to them all in my humble acknowledgment.

In the first place I would like to record my gratitude to Dr. Laurent Fesquet for his supervision, advice, and guidance from the very early stage of this research as well as giving me extraordinary experiences throughout the work. Above all and the most needed, he provided me unflinching encouragement and support in various ways. His truly scientist intuition has exceptionally enriched my growth as a scientist want to be.

I greatly indebted to my co-supervisor Dr. Marc Renaudin for all his precious support, consistent encouragement and valuable guidance thought the start of my thesis. I owe an enormous debt of gratitude to him, for the confidence that he accorded to me by accepting to participate in the supervision of my thesis.

I am grateful to the thesis jury members for their precious time which they have scarified for me. Thanks to Prof. Michel Robert, for the interest he gave to this work by agreeing to be the president of my thesis jury. I am also thankful to the thesis reviewers, Prof. Olivier Sentieys and Prof. Habib Mehrez, the interest they have shown towards my work has brought an outside perspective on enriching the subject. Their constructive suggestions on the thesis are really appreciated for me. I am grateful that in the midst of their activities, they accepted these tasks. My thanks also go to Dr. Carlos Canudas-de-Wit and Dr. Mario Diaz Nava, for their commitment to take part in my jury committee.

My deep regards for Dr. Gilles Sicard, for his precious friendly attitude and his immediate help for many administrative issues. I am also grateful to Mr. Alexandre Chagoya, the responsible of service design and test at CIME Nanotech. He always provided me with the needed help and software tools for my work. I also thank all the

administrative staff of the TIMA laboratory and the EEATS, who have always been gentle with me and helped me to get things done very smoothly.

Collective and individual acknowledgments are also owed to my colleagues in CIS group whose present somehow perpetually refreshed, helpful, and memorable. Many thanks go in particular to Eslam and Oussama for their friendship, brotherhood and continual support in work and life. Thanks to Taha, Hakim, Franck and Gregory for the great discussions and open minded attitude. I cannot finish without thanking also Khaled, Saeed, Jeremie, Florant, Alexandre, Hawraa, Olivier, Mathieu, David and Rodrigo. Special thanks also go to all my colleagues in the ARAVIS Minalogic project, those from INRIA, CEA-Leti and STMicroelectronics for their appreciated discussions and valuable remarks all over my PhD work.

My warm special thanks to my family, for their consideration of my research abroad. Without their encouragement, inseparable support and prayers it would have been impossible for me to finish this work. My Father, in the first place is the person who put the fundament my learning character, showing me the joy of intellectual pursuit ever since I was a child. My Mother is the one who sincerely raised me with her caring and gently love. My sisters and brother, who inspired my life with the real meaning of the family. My dearest and lovely kid, Ali who is the real source of happiness in my life, he actually suffered a lot because of my busyness.

Words fail me to express my appreciation to my wonderful and beautiful wife, whose prayers, dedication, love, continual support and persistent confidence in me, has taken the load off my shoulder. I owe her for being unselfishly let her intelligence, passions, and ambitions collide with mine.

All gratitude to my dearest teachers and professors those did great efforts to drive me up to the scientific level which I have now. Special thanks for Dr. Abdel Aziz El-Bassiouni who always keep encouraging and supporting me to do the best.

Hatem ZAKARIA **Grenoble, France**

February 2011

Abstract

Continuous scaling of CMOS technology push circuit designs towards multi-core complex SoCs. Unfortunately with the nanometric technologies, the integrated system performances after fabrication will not be fully predictable. Indeed, the process variations really become huge at the chip scale. Therefore the design of such complex SoCs in the nanoscale technologies is now constrained by many parameters such as the energy consumption and the robustness to process variability. This implies the need of efficient algorithms and built-in circuitry able to adapt the system behavior to the workload variations and, at the same time, to cope with the parameter variations which cannot be predicted or accurately modeled at design time. In this context, this thesis work addresses the design of Globally Asynchronous Locally Synchronous “GALS” based Network-on-Chip “NoC” architectures in the upcoming CMOS technologies. A novel methodology to dynamically control the speed of different voltage-frequency NoC islands according to the process variability impact on each domain is proposed. This control technique can improve the performances, the energy consumption, and the yield of future SoC architectures in a synergistic manner. The control methodology is based on the design of an asynchronous programmable self-timed ring where the controller takes into account the dynamic workload and the process variability effects. The controller especially considers the operating frequency limit which does not exceed the maximum locally allowed value for a given clock domain. With such an approach, it is no more required to separately guaranty the timing performances for each node at design time. This drastically relaxes the fabrication constraints and helps the yield enhancement.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Context and Motivations

With increased levels of integration in scaled technologies, complex systems containing (large number of processors, on-chip memories, IPs, complex clock trees, I/O control units, etc.) have become a reality. As recognized by the International Technology Roadmap for Semiconductors (ITRS), dealing with on-chip communications and power management problems require a drastic departure from the classic design methodologies [SEM 06]. Therefore novel on-chip communication architectures that use a Network-on-Chip (NoC) approach have emerged as a scalable alternative to traditional bus-based or point-to-point communication solutions [DAL 01] and [LEE 07].

By eliminating global wires, the NoC approach provides the needed modularity and performance, while facilitating design reuse. Moreover, the NoC approach offers a matchless platform for implementing the globally asynchronous locally synchronous (GALS) design paradigm [CHA 84] and [MUT 00]. This makes the clock distribution and timing closure problems more manageable. In addition, a GALS design style fits nicely with the concept of different voltage-frequency domains, which provides better power-performance tradeoffs than its single voltage, single clock frequency counterpart [DIE 03] and [BER 05], while taking advantage of the natural partitioning and mapping of applications onto the NoC platform. However, the implementation of GALS systems makes mandatory the synchronization of the communicating clock domains frequencies and phases in order to guarantee that data is reliably transferred among them [BEI 06].

In addition to this huge increase in system complexity, the systems designed in nanoscale technologies suffer from systematic and random variations in process, voltage, and temperature (PVT), particularly from the 45nm CMOS technologies and beyond. As

a result, the integrated system performances after fabrication will not be fully predictable. Indeed, within die variations play an increasingly important role in system power consumption and performance as the technology scales down [BOR 03].

Since designers cannot rely on the accuracy of the nominal parameter values, there is a tremendous need for on-line techniques that can cope with such dynamic variations [REB 09] and [MIE 08]. More precisely, there is a need for efficient algorithms and built-in circuitry able to adapt the system behavior to workload variations and, at the same time, cope with the parameter variations which cannot be predicted or accurately modeled at design time.

Not surprisingly, designing appropriate dynamic voltage and frequency scaling (DVFS) control algorithms for run-time control of different voltage-frequency domains in a GALS system is a matter of great importance. While this problem has been addressed before by a number of authors [OGR 08], [WU 04] and [NIY 05], no attention has been given to add the feature of controlling the impact of manufacturing process variations to the capabilities of the DVFS controllers.

Starting from these overarching ideas, ARAVIS project (Architecture avancée Reconfigurable et Asynchrone pour la Video et la radio logicielle Intégrée Sur puce) sponsored by Minalogic, looks for architecture and design solutions that allow the production of embedded computational platforms in its scalability limit. It proposes a generalization of certain techniques in order to obtain a solution to the technology variability problem in 32nm, which will represent an input toward the development of a new paradigm. The ARAVIS project is focused on three technology keys:

1- Reconfigurable structure with respect to applicability requirements. It can be accomplished by programming the flexible interconnections between the clustered nodes of the SoC computational unit [PIN 01].

2- GALS paradigm in order to release the communication constraints between remote points.

3- Dynamic management of the power consumption and activity with respect to constraints are achieved by control theory application [HEL 04].

This thesis is included as a part of the ARAVIS project context. It specifically focuses on the design of GALS-based NoC architectures in the upcoming CMOS technologies. Activity monitors are disseminated into each voltage-frequency island to locally evaluate the process quality in terms of its relative speed with respect to the other processing nodes. A novel methodology to dynamically control the speed of different voltage-frequency GALS-NoC islands according to the process variability impact on each domain is proposed. This control technique can improve the performances, the energy consumption, and the yield of future SoC architectures in a synergistic manner.

The proposed control methodology is based on the design of an asynchronous programmable self-timed ring where the controller takes into account the dynamic workload and the process variability effects. The controller especially considers the operating frequency limit which does not exceed the maximum locally allowed value for a given clock domain. With such an approach, it is no more required to separately guaranty the performance for each node. This drastically relaxes the fabrication constraints and helps the yield enhancement.

1.2 Thesis Outline

This thesis dissertation is mainly split into two main parts: GALS Paradigm (Part-I) and CMOS Power Reduction and Design for Yield (Part-II).

This first part comprises the chapters 2 to 4. In Chapter 2 an introduction of the asynchronous circuits is introduced. Asynchronous circuit classes are discussed stating the different handshaking protocols. In addition to this, token and bubble rules were also presented to facilitate the discussion of data flow in asynchronous circuits. This helps to understand the basic principle of operation of self-timed rings.

In Chapter 3, we propose a design for a Programmable/Stoppable Oscillator (PSO) which is based on self-timed ring to exploit its interesting characteristics (programmability, accuracy and robustness against process variability). Through a handshaking protocol, the oscillator is communicating with the synchronous processor to ensure a proper switching from one frequency to another. The oscillator is designed in order to avoid the presence of glitches and truncated clock periods.

As we consider GALS-NoC architectures, low latency high throughput asynchronous communication mechanisms between synchronous blocks will be obligatory. Therefore in Chapter 4, we make use of the previously designed programmable self-timed ring to propose a new scheme based on the use of asynchronous handshake circuits to safely synchronize these clocks. This circuit design gathers the small area and the low power consumption advantages of the pausible clocking GALS with the high throughput advantage of the FIFO-based GALS.

The second part of the thesis contains the chapters 5 to 7. The necessary concepts needed to understand and well characterize the main sources of CMOS power dissipation in the nanometric era are declared in Chapter 5. Moreover, in this chapter we investigate different low power solutions that are needed to reduce the impact of each contributing part to the total CMOS power consumption. As DVFS is recognized as one of the most effective power reduction techniques, therefore its different algorithms and its architecture are detailed.

In Chapter 6, a novel DFVS architecture that dynamically adapts the speed of each voltage-frequency island in a GALS system is proposed. The DVFS control principal not only considers the dynamic workload variations, but also ensures that the operating frequency does not exceed the maximum allowed value for a given process variability effect. The control methodology is based on the use of the asynchronous PSO with a fast predictive feedback controller.

Chapter 7 addresses the problem of designing the proposed process variability robust DVFS control methodology. MIPS R2000 presents our processing load case study. The PSO was programmed and configured to compensate for MIPS R2000 variations on STMicroelectronics 45nm CMOS technology. Moreover, the digital controller part of the DVFS was completely designed and tested on different process variability corners.

Finally the conclusion of the thesis and the prospects are discussed in Chapter 8.

Part – I

GALS Paradigm

Chapter 2

Asynchronous Circuits

2.1 Introduction

In recent years there has been tremendous growth in the silicon integration capacity. Starting from the 45 nm technology and beyond, designers face various problems in power consumption, process variability, environment-parameters (Process, Voltage, and Temperature) “PVT” variations and Electromagnetic Interference “EMI”. As synchronous design style is based on global timing assumptions determined by the clock. Coping with this assumption, especially with the recent technologies, is challenging from two points of views. First, the increase of process variability implies inefficient increase in timing doubt while designing. Second, clock trees are gradually consuming more power and needing more effort for managing. Therefore, different solutions are presented for these challenges as multi-clock systems and clock gating. However, Asynchronous circuits show an efficient alternative solution for these problems [REN 03] and [BEE 02].

Asynchronous circuits use handshaking between their components in order to perform the necessary synchronization, communication, and sequencing of operations. This results in a behavior that is similar to systematic fine-grain clock gating and local clocks that are not in phase and whose period is determined by actual circuit delays; i.e. registers are only clocked where and when needed. This behavior gives asynchronous circuits the following interesting features [SPA 01], [BER 99], and [MAR 97]:

(1) Low power consumption

Because asynchronous circuits do not need any clock signals, the power spent on clock switching in a synchronous chip is avoided. Additionally, the signal transitions in asynchronous circuits will automatically stop when there is no data. Therefore, asynchronous circuit design has zero standby dynamic power consumption), [BER 94].

(2) No global-signal distribution and clock skew problems

Since asynchronous circuit designs have simple handshake interfaces and local timing, the difficulties of clock distribution and clock skew faced by synchronous designs are removed from asynchronous designs.

(3) Low emitted EMI noise

In a synchronous design, flip-flop transitions follow a certain clock frequency so that the energy spent on signal transitions concentrates within the very narrow bands around the clock frequency. Thus, the synchronized signal switching activities will produce substantial electrical noise. Whereas, the switching activities in an asynchronous circuit are correlated loosely because there is no universal timing pace, hence, they produce a more distributed noise spectrum and a lower peak noise value, [PAV 98].

(4) Average-case performance

In a synchronous design, the operating speed is limited by the worst-case, called critical path in the circuits. However, in asynchronous circuits, the operating speed is determined by actual local latencies in the circuits rather than global worst-case latency. In most of cases, the average-case of latencies are smaller than the worst-case latency, hence, asynchronous designs can achieve better operating speed performance, [WIL 91] and [WIL 95].

(5) Robust and adaptive

A synchronous circuit is sensitive to the delay variations caused by the process variability effects and environment-parameters variations (i.e. clock signal, supply voltage, and operating temperature) related with the manufacturing process and application surrounding. Whereas, the loose timing requirement of asynchronous circuits allow them to operate correctly under large variations caused by different manufacturing processes and application environment**,** [NIE 94] and [SPA 01].

(6) Better modularity

The modularity of asynchronous circuits is almost perfect. It is due to the locality of control and the use of a well-specified communication protocol by all operators. It is indeed easy to construct a complex system involving pre-existing blocks, [SPA 01].

According to the previously mentioned asynchronous circuit advantages, research and industry are progressively more motivated to the introduction of asynchronous technology into products and design flows. On the other hand there are also some drawbacks of the asynchronous circuit implementation. The asynchronous control logic that implements the handshaking normally causes an overhead in terms of silicon area, circuit speed, and power consumption. It is therefore important to ask whether or not the investment pays off, i.e. whether the use of asynchronous technique results in a considerable improvement over the synchronous one or not [SPA 01].

2.2 Asynchronous Circuits Principles

In synchronous design style, depicted in Figure 2.1, synchronous circuits are composed of combinational function blocks and registers. The circuit activity is controlled by a global clock which triggers at the same time the memorization of the complete state of the circuit. As a new state is sampled and placed in the registers, the combinational circuits start the computation of the next state to be sampled at the next clock edge. The clock signal is fixed so that all functional blocks correctly complete their operations and their data outputs are stable and ready to be sampled. That implies a global timing assumption which is applied to the whole circuit: the longest combinational path (critical path) must not exceed the clock period. Synchronization in asynchronous circuits is done by replacing the clock signal with some form of handshaking between adjacent registers; for example the request-acknowledge based handshake protocol shown in Figure 2.2 [MYE 01] and [SPA 01].

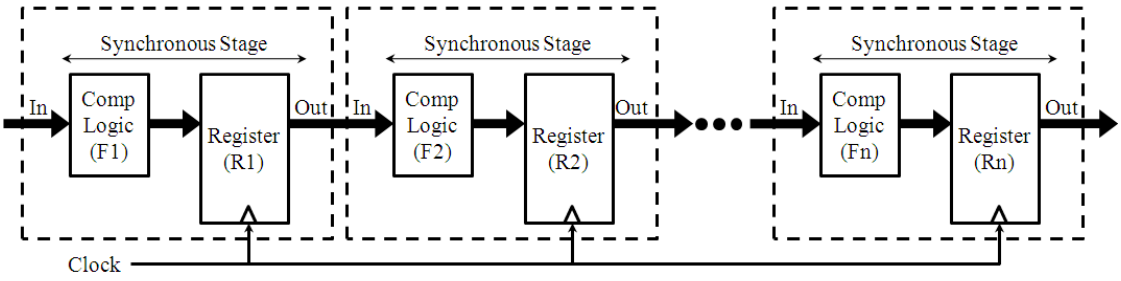


Figure 2.1: Basic Structure of a Synchronous Circuit.

Asynchronous circuits are composed of communicating stages which computation is controlled by the presence of data at their inputs and outputs. Their behavior is much similar to the data-flow model, [REN 03]. Let's consider one asynchronous stage described in Figure 2.2. It receives data from its input port, the functional block computes it, and sends the result through a register to its output port. Data communication over its ports are not controlled by an external signal, like a clock, but by a communication protocol implemented within the control part of the stage itself. Such protocols require a bi-directional exchange of information between senders and receivers called handshake protocols.

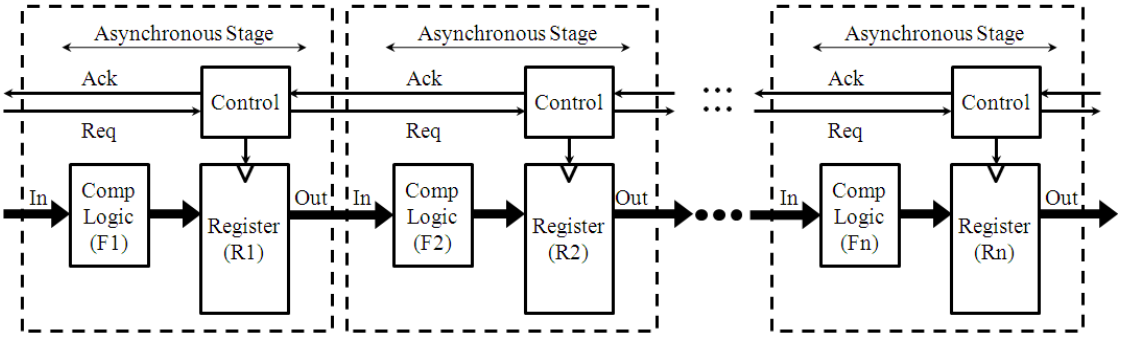


Figure 2.2: Basic Structure of an Asynchronous Circuit.

The communication protocol is the basis of the sequencing rules of asynchronous circuits. The first rule is that an asynchronous stage starts the computation if and only if all the data required for the computation are available. Once the result can be stored in the register, the asynchronous stage releases the input ports. It outputs the result through the output port if and only if this port is available, i.e. released by the next stage connected to it, at the end of the previous communication. The implementation of the communication protocol in each stage is the price to pay to get rid of the clock and be able to control the sequencing locally, [REN 03].

Each asynchronous stage can implement functions with very different granularity: bit-level functions, word-level arithmetic functions or even complex algorithms. However, they always have the following three main characteristics, [YUN 96], [REN 96], [CUM 94], and [WIL 94]:

1. All their channels respect a unified communication protocol.
2. They all compute the input data set at the highest speed and place the result at its output as soon as possible (minimum forward latency), while, the synchronous circuits forward latency may not vary with the data computed. The forward latency is constrained by the clock period.
3. They also all have a maximum throughput which corresponds to the maximum frequency at which the module can process the incoming data (maximum cycle time). This characteristic is not the inverse of the forward latency because the asynchronous stage has to release the communication channels before accepting the next data set. With asynchronous circuits, we have to consider the forward and reverse phases to compute the throughput (Notice that is not a trivial relation because this depends on the employed handshaking protocol).

2.3 Handshaking Protocols

To implement a bidirectional signaling in asynchronous circuits, two communication protocols are commonly used: the two-phase protocol, also called NRZ (Non Return to Zero) or “half-handshake” and four-phase protocol, also known as RZ (Return to Zero) or “full-handshake”. Both protocols are respectively shown in Figure 2.3 and Figure 2.4. In both cases, it should be noted that any event on a signal from the transmitter stage is reset by an event on a signal from the receiver stage, and vice versa. This mechanism ensures the insensitivity to the processing time of the operator. In these protocols, the only importance is in the occurrence of events locally between the transmitter and the receiver, and not in their relative time, or their respective orders in relation to the transmitter input or the receiver output. The choice of communication protocol affects the characteristics of the board layout (Area, speed, power consumption, robustness, etc ...), [REN 03] and [SPA 01].

The two phase protocol has the minimum required sequence of information exchange per communication. Signaling in the 2-phase protocol is event-based since data

detection and acknowledgment are done by means of rising and falling edges of signals. The two phases are as follows:

1st Phase: The transmitter generates a transition on the request signal. The receiver detects the incoming data and once the reception process is completed, it generates a transition on the acknowledgement signal.

2nd Phase: The transition on the acknowledgement signal is detected by the transmitter and depending upon the availability, the new data is transmitted.

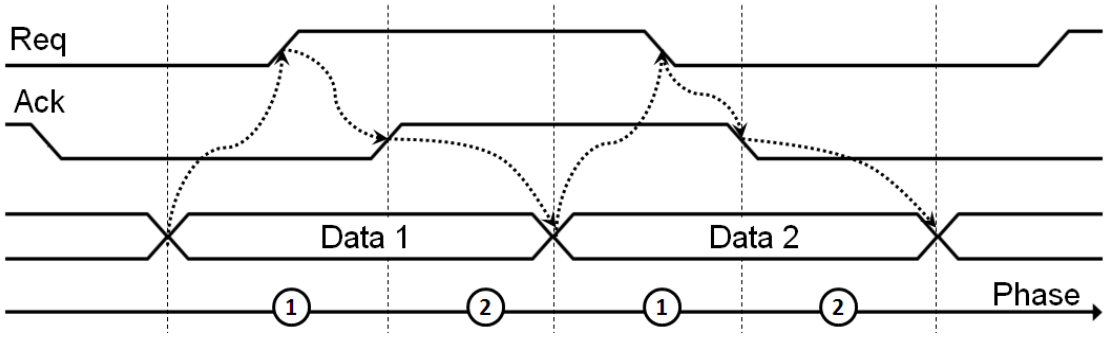


Figure 2.3: A Two-Phase Protocol.

Four phase protocols are level-based. These protocols require a return to zero phase for both the data and the acknowledgement as shown in Figure 2.4. The sequencing between two asynchronous stages in the four phase protocol is done in the following manner:

1st Phase: The transmitter issues data and sets the request signal high.

2nd Phase: The receiver absorbs the data and sets the acknowledgement high.

3rd Phase: The transmitter responds by taking request low (at which point data is no longer valid).

4th Phase: The receiver acknowledges this by taking acknowledge low. At this point the transmitter may initiate the next communication cycle.

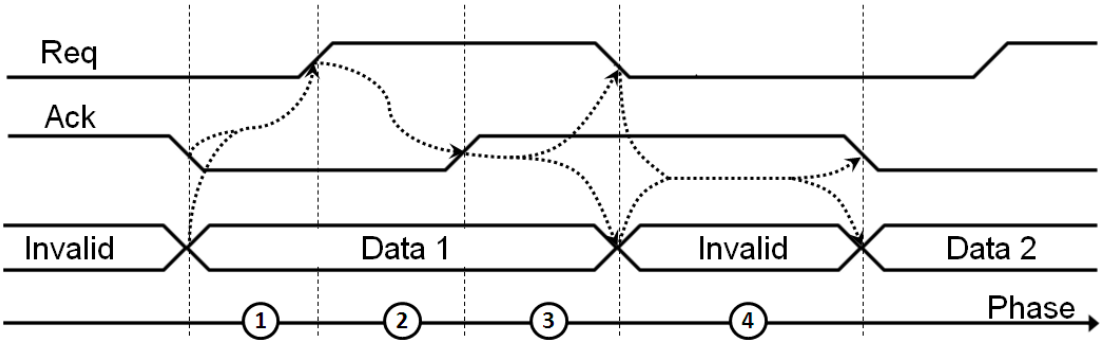


Figure 2.4: A Four-Phase Protocol.

Despite of the apparent effectiveness of the two-phase protocol (i.e. faster and less power consuming), but actually it is not always the case because event-based logic is more costly than level-based in terms of CMOS implementation. Moreover, very effective optimizations can be done at the logic and architectural levels when using four-phase protocols. Generally speaking two-phase protocols are preferred when slow components (with large forward latencies) are involved in the cycle time. On the contrary, four-phase protocols are mostly used within the circuit due to the symmetry of its phases, where the designer is free of balancing component latencies to optimize the circuit performances [REN 03], [SPA 01], and [REN 00].

2.4 The Muller's C-element

In a synchronous circuit the checking role is to define points in time where signals are stable and valid. In between the clock ticks, signals may exhibit hazards and may have multiple transitions till the combinational circuits stabilize. This does not matter from the functional point of view. Conversely, asynchronous control circuits shown in Figure 2.2 have a different behavior. The absence of the clock means that, in many situations signals are required to be valid all the time, that every signal transition has a meaning. As a result, hazards and races must be avoided. A circuit that is better in this respect is the Muller's C-element shown in Figure 2.5, [RIG 02], and [MAR 86]. Actually, it is a state holding element works in the following principle: When both inputs are 0 the output is set to 0, and when both inputs are 1 the output is set to 1. For other input combinations the output does not change. Consequently, an observer seeing the

output change from 0 to 1 may conclude that both inputs are now at 1; and similarly, an observer seeing the output change from 1 to 0 may conclude that both inputs are now 0. Combining this with the observation that all asynchronous circuits rely on handshaking that involves cyclic transitions between 0 and 1, it should be clear that the C-element is indeed a fundamental component that is extensively used in asynchronous circuits.

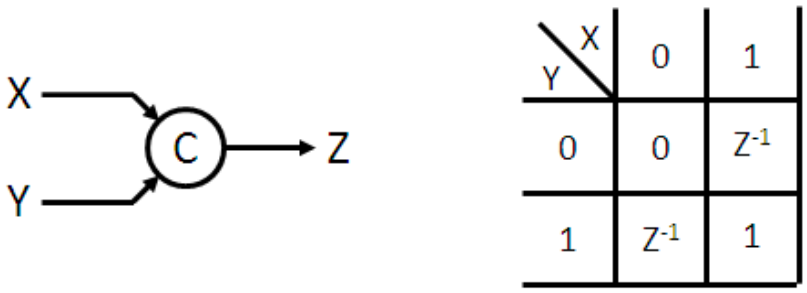


Figure 2.5: The Muller's C-element: Symbol and Truth Table.

2.5 The Muller's Pipeline

Figure 2.6 shows a circuit that is built from C-elements and inverters. The circuit is known as a Muller's pipeline, [GUN 93] and [MUL 59]. Variations and extensions of this circuit form the control backbone of almost all asynchronous circuits. This Muller's pipeline is a way to transmit the handshakes. After all of the C-elements have been initialized to 0, the left environment may start handshaking. To understand what happens let's consider the *i*th C-element, C[*i*]: It will propagate a 1 from its predecessor, C[*i*-1], only if its successor, C[*i*+1], is 0. In a similar way it will propagate a 0 from its predecessor if its successor is 1. It is often useful to think of the signals propagating in an asynchronous circuit as a sequence of waves, as illustrated at the bottom of figure 2.6. Viewed this way, the role of a C-element stage in the pipeline is to propagate these waves in a carefully controlled way that maintains the integrity of each wave.

On any interface between C-element pipeline stages an observer will see correct handshaking, but the timing may differ from the timing of the handshaking on the left hand environment. Once a wave has been injected into the Muller pipeline, it will propagate with a speed that is determined by the actual delays in the circuit. Eventually

the first handshake (request) injected by the left hand environment will reach the right hand environment. If the right hand environment does not respond to the handshake, the pipeline will eventually fill. If this happens the pipeline will stop handshaking with the left hand environment. In addition to this elegant behavior, the pipeline has a number of beautiful symmetries. Firstly, it does not matter if 2-phase or 4-phase handshaking is used. The difference is in how the signals are interpreted and the circuit is used. Secondly, the circuit operates equally well from right to left, with reversing the role of the request and acknowledge signals. Finally, the circuit has the interesting property that it works correctly regardless of delays in gates and wires. The Muller pipeline is delay-insensitive.

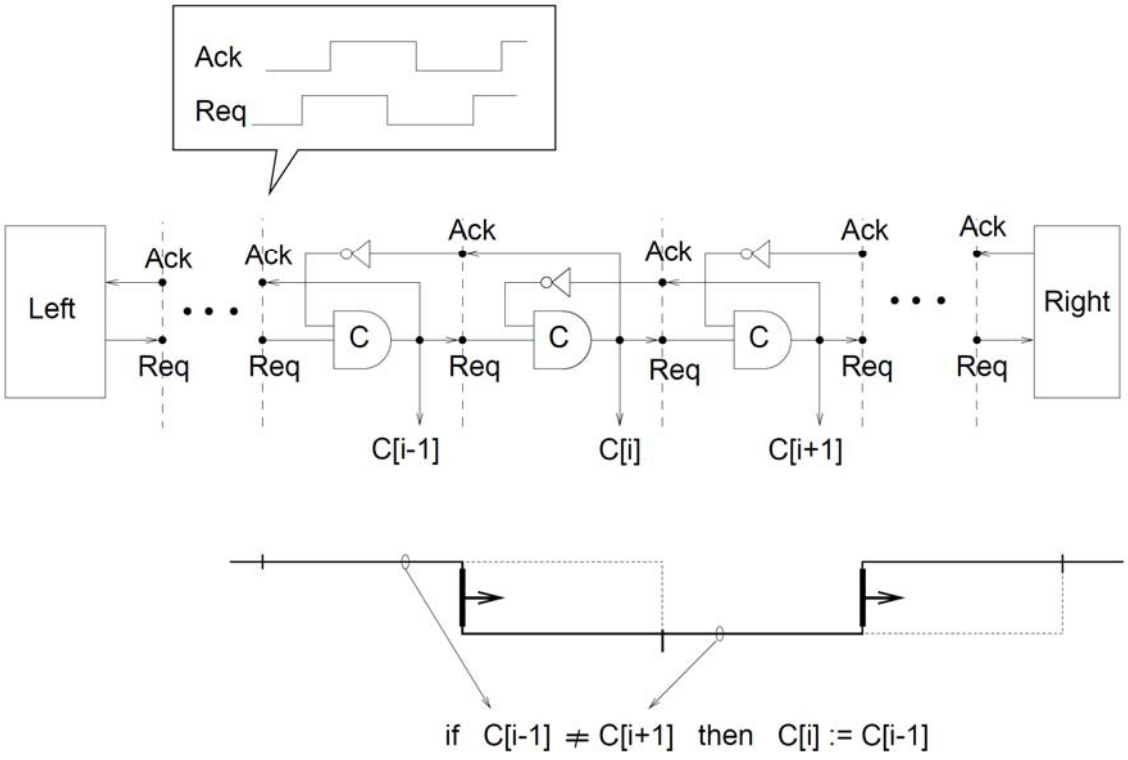


Figure 2.6: The Muller Pipeline.

2.6 Classification of Asynchronous Circuits

Asynchronous circuits can be classified as being speed-independent, delay insensitive or self-timed depending on gate and wire delays assumptions that are required to guarantee their functional correctness [PAN 02], and [EBE 91]. Figure 2.7 facilitate the illustration of this concept. The figure shows three gates: A, B, and C, where the output signal from gate A is connected to inputs on gates B and C.

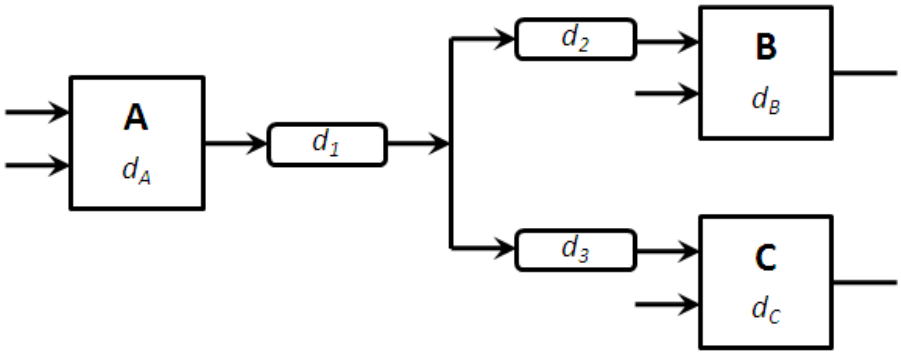


Figure 2.7: Gates and Wires Delay Model for a Circuit Fragment, the output of Gate A forks to inputs of gates B and C.

Speed-Independent (SI):

This class of circuits operates correctly assuming that positive, bounded but unknown delays in gates and ideal zero-delay wires, [EBE 91]. Referring to Figure 2.7 this means arbitrary d_A, d_B, and d_C, but $d_1 = d_2 = d_3 = 0$. Assuming ideal zero-delay wires is not very realistic in today's semiconductor processes.

Delay-Insensitive (DI):

This class of circuits operates correctly with positive, bounded but unknown delays in wires as well as in gates. Referring to Figure 2.7 this means arbitrary d_A, d_B, d_C, d_1, d_2, and d_3. Such circuits are obviously extremely robust. At the gate level, only circuits composed of C-elements and inverters can be delay-insensitive, and their functionality is limited. This is why delay-insensitive circuits are usually built out of modules which require some delay assumptions internally. Circuits that are delay-

insensitive with the exception of some carefully identified wire forks, where $d_2 = d_3$ are called *Quasi-Delay-Insensitive* (QDI). Such wire forks where signal transitions occur at the same time at all end points are called “isochronic forks”, [BER 92] and [MAR 90]. At the gate level, this assumption enables the design of circuits made of single-output gates.

Self-Timed Circuits:

Speed independency and delay-insensitivity as introduced above are mathematically well defined prosperities under the unbounded gate and wire delay model, [SPA 01]. Circuits whose correct operation relies on more elaborate and/or engineering timing assumptions are simply called self-timed. In the next chapter a full description of self-timed rings and their programmability will be presented.

The different circuit classes SI, DI, QDI and self-timed are not mutually exclusive ways to build complete systems, but useful abstractions that can be used at different levels of design, [SPA 01] and [HAU 95]. In most practical designs they are mixed. For example, in the Amulet processors SI design is used for local asynchronous controllers, and DI is used for high-level composition, [GAR 00]. Therefore, the careful choice of handshake protocol and circuit implementation style is among the factors to optimize an asynchronous digital system.

2.7 Pipelines and Rings: Token Game

At architectural level, in order to ease the token flow discussion it is assumed that an asynchronous circuit is composed of functional blocks (combinational circuits), memory elements (registers) and channels (request, acknowledgment, and data signals). Figure 2.8 shows the basic elements of an asynchronous circuit.

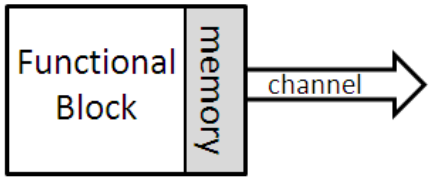


Figure 2.8: Basic Elements of an Asynchronous Circuit.

A token is carrying information and is stored in a memory element. A token is represented by a filled circle next to the memory element it is stored in. When using a 4-phase protocol, the asynchronous data-path processes a stream of alternating valid and return to zero tokens. When a 2-phase protocol is used, there are only valid tokens, but apart from that everything is the same. Data flow information is controlled by two rules, [REN 03]:

> *Token rule: a memory may receive and store a new token from its predecessor if and only if it has a bubble.*
>
> *Bubble rule: a memory becomes empty (bubble) if and only if its successor has received and stored the token that it was holding.*

Figure 2.9 shows a snapshot of a simple five stage pipeline, which is implemented using 2-phase protocol. The valid value in L1 and L3 has just been copied into L2 and L4 respectively. This means that L1 and L3 are now holding old duplicates of the values now stored in L2 and L4. Such old duplicates are called “bubbles”, and the newest valid values are called “tokens”. To distinguish tokens from bubbles, tokens are presented with a circle around the value. Bubbles can be viewed as a medium: a bubble allows a token to move forward, and in supporting this, the bubble moves backward one step.

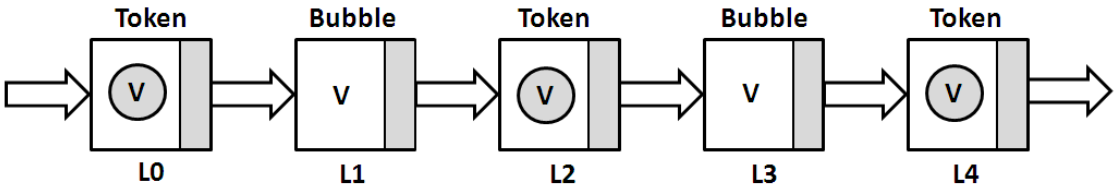


Figure 2.9: A Possible State of a Five Stage Pipeline.

Any circuit should have one or more bubbles; otherwise it will be in a deadlock state. This is a matter of initializing the circuit properly. Furthermore, as we will see in the next chapter, the number of bubbles also has a significant impact on performance.

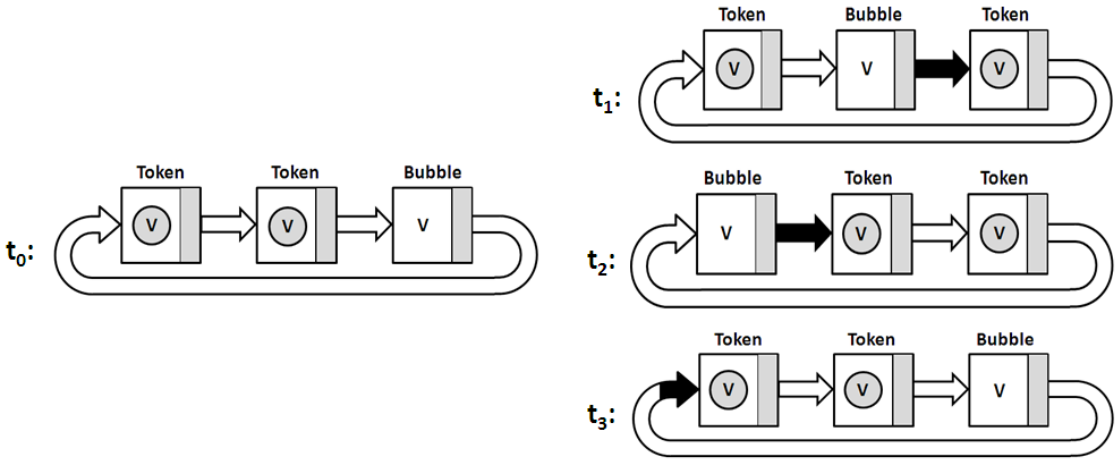


Figure 2.10: A Sequence of Data Transfers in a Ring.

In a pipeline with at least three stages, it is possible to connect the output of the last stage to the input of the first stage, forming a ring in which data tokens can circulate autonomously, [SPA 01]. Assuming the ring is initialized as shown in Figure 2.10 at time t_0 with two valid tokens and a bubble, the steps of the circulation process are shown in Figure 2.10, at time t_1, t_2 and t_3. Rings are the backbone structure of circuits that perform iterative computations. The cycle time of the ring in Figure 2.10 is 3 steps (the state at t_3 is identical to the state at t_0).

Next chapters of the thesis show how programmable versions of 2-phase self-timed rings could be used, in order to: Firstly, synchronize the data transfer between different clock domains in a GALS system, while avoiding metastability problems. Secondly, manage not only the energy consumption but also the process variability in complex integrated system (System on Chip often denoted SoC).

2.8 Conclusions

Problems with the clock in synchronous circuit design are becoming more and more complex, which open the path to the design of asynchronous circuits. Unlike the global clock mechanism used in synchronous circuits, timing in asynchronous circuits is done locally through a bidirectional signaling between all elements of the circuit. This signaling is established via communication channels using a specified handshaking

protocol (2-phase or 4-phase). Asynchronous circuits are classified into different types according to their gate and wire delays assumptions that are required to guarantee their functional correctness. Asynchronous circuits make the design of distributed finite state machines as well as data paths much easier than before, as there is no need to know the global state of the system to make things communicate and synchronize, where it is the case with synchronous circuits. Token and bubble rules were presented to facilitate the discussion of data flow in asynchronous circuits. This also helps to understand the basic principle of operation of self-timed rings, which will be presented in details within the next chapter.

Chapter 3

Self-Timed Rings

3.1 Introduction

Ring oscillators are one of the most essential building blocks in many digital and communication systems. They are used in to generate and distribute timing signals which synchronize processing unit operations. Programmable versions of these ring oscillators have a variety of applications. In Dynamic Voltage and Frequency Scaling “DVFS” systems, they are used to dynamically manage the energy consumption of a complex system integrated on a chip. This is done with the regular adaption of the processing unit clock frequency according to the computational needs, as will be shown in Chapter 5. In Globally Asynchronous Locally Synchronous “GALS” systems programmable ring oscillators are also required in order to adjust the clock frequency for each clock domain, where each domain has its own local clock frequency, as will shown in Chapter 4. With the rapid increase in the process variability effects on the system behavior due to the shrinking of the integrating technology, one must carefully choose the operational ring oscillator. This oscillator should have robust behavior under different environmental and process variability effects.

Adjustable clocks can be derived from different kinds of clock generators. For example, these clocks can be derived from analog Voltage Controlled Oscillators “VCO”, which are a part of a Phase Locked Loop “PLL”. However, VCO have a limited operating range and a required stabilization time when changing the frequency [BOY 06]. Another solution is to use a standard clock divider, but this will make the time resolution coarser, due to counting integer periods of the input frequency [STO 03]. In addition, they give regular time step which implies irregular frequency step (usually frequency step follows “1/x” curve).

Asynchronous self-timed rings are considered promising solution for generating clocks. In [FAI 04] they are efficiently used to generate high-resolution timing signals. Their robustness against process variability in comparison to inverter rings is proven in [HAM 08]. They are efficiently used as a data driven clocks in [MUL 07]. Moreover, for a given number of stages self-timed rings can be reconfigured easily by controlling their initialization (i.e. their number of tokens and bubbles; explain in the sequel), while in the contrary, inverter-ring frequency is fixed [FAI 04]. Furthermore, as events propagate between adjacent stages in self-timed rings according to a simple request/acknowledge handshake signals, we can make use of these handshakes in synchronizing the data transfer between different clock domains in a GALS system while avoiding metastability problems, as will be shown in Chapter 4.

This chapter proposes a design for a Programmable/Stoppable Oscillator which is based on asynchronous self-timed ring to exploit its interesting characteristics. Through a handshaking protocol, the oscillator is communicating with the synchronous processor to insure a proper switching from one frequency to another. The oscillator is designed in order to avoid the presence of glitches and truncated clock periods. A new methodology for calculating the ring oscillation period will also be presented.

3.2 Ring Structure

Various kinds of implementations of a self-timed ring exist and the underlying token and bubble notions depend on the communication protocol (e.g. 2-phase or 4-phase communication protocol). In this context, it is important to clearly define the structure and the token and bubble notions.

3.2.1 Ring Connectivity

Figure 3.1 shows the structure of the ring stage. It is composed of a Muller C-element and an inverter. In each stage, the input which is connected to the previous stage is marked F (Forward) and the input which is connected to the following stage is marked R (Reverse), C denotes the output of the stage. Figure 3.2 shows an example of a ring composed of self-timed handshaking N-stages, [MUL 59].

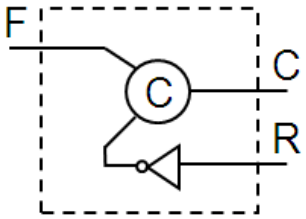


Figure 3.1: A Ring Stage

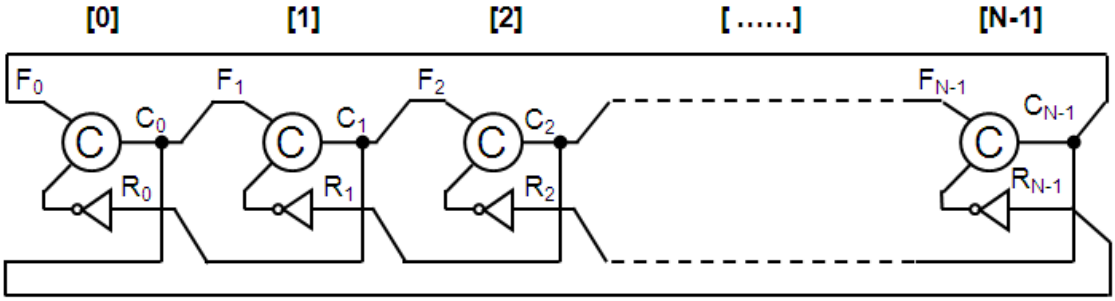


Figure 3.2: A Self-Timed Ring.

3.2.2 Definitions and Notations

We choose the 2-phase communication protocol as defined in [SUT 89], which allows us to derive the token and bubble concepts of the self-timed ring as follows:

- $Stage_i$ contains a token if its output C_i is not equal to the output C_{i+1} of $stage_{i+1}$.

$$C_i \neq C_{i+1} \Leftrightarrow Stage_i \leftarrow Token$$

- $Stage_i$ contains a bubble if its output C_i is equal to the output C_{i+1} of $stage_{i+1}$.

$$C_i = C_{i+1} \Leftrightarrow Stage_i \leftarrow Bubble$$

The numbers of tokens and bubbles are denoted by N_T and N_B respectively. Because a token is defined with respect to the value of two adjacent stages, N_T is an even number whatever the number of stages. Moreover, each stage of the ring contains either a token or a bubble, so $N_T + N_B = N$, where N is the number of the ring stages. As self-timed rings have a 2-phase pipeline structure, the data flow through its adjacent stages is the same as it was previously discussed in Chapter 2. If a token is present in $stage_i$, it will propagate to $stage_{i+1}$, if and only if $stage_{i+1}$ contains a bubble. The bubble of $stage_{i+1}$ will

move backward to $stage_i$. This implies a transition on $stage_{i+1}$. In Figure 2.10 an example was shown for these token/bubble movements in a three stage self-timed ring, which contains two tokens and one bubble. With the previous notations, the necessary and sufficient condition for a token to propagate from $stage_{i-1}$ to $stage_i$ can be expressed as:

$$C_{i-1} \neq C_i = C_{i+1}$$

From these propagation rules, we can derive the requirements to obtain an oscillating asynchronous self-timed ring. By definition, the number of tokens in a ring must be even and at least one bubble is necessary to enable tokens to move. The minimal number of stages is hence 3. Self-timed rings produce two different modes of oscillation: "Evenly spaced" or "Burst" modes, see Figure 3.3. In the evenly spaced mode, the events inside the ring are equally spaced in time. In the burst mode, the events are non-uniformly spaced in time. In our application, we only target the evenly spaced mode.

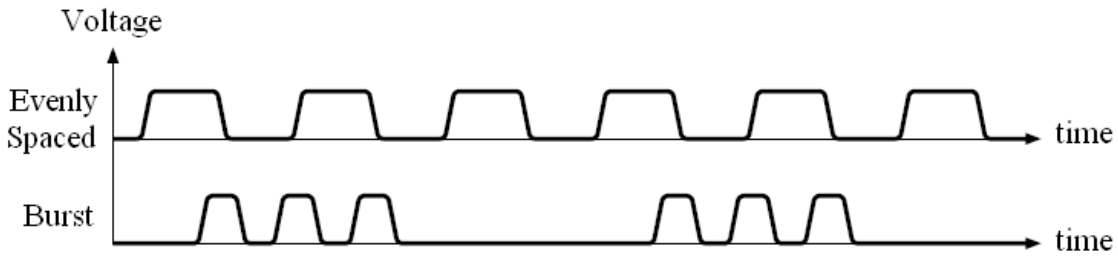


Figure 3.3: Modes of Operation in Self-Timed Rings.

3.3 Ring Modeling

The key to understand the temporal properties of self-timed rings lies in finding an appropriate model for the ring. The ideal model will be one that captures the non-linear timing dependencies of real circuits while still being simple enough to provide insight into the causes of bursting and uniformly spaced behavior. Two key effects have been identified as being responsible of the ring behavior, the Charlie effect, [EBE 98] and [ZEB 05], and the Drafting effect [WIN 01], [WIN 02], and [FAI 04].

3.3.1 The Charlie effect

The deep observation of the C-element temporal behavior shows that the separation time between input events has a direct impact on the propagation delay: "*the closer the input events; the longer the propagation time*". This phenomenon is called "Charlie effect" in reference to Charles E. Molnar. The analysis of the C-element structure can explain these variations in the propagation time. Figure 3.4 shows a conventional implementation of the C-element referenced as "week feedback", [SHA 98]. It can be divided into two stages: the input stage which calculates the output value when both inputs are identical and the output stage which stores this value in other cases.

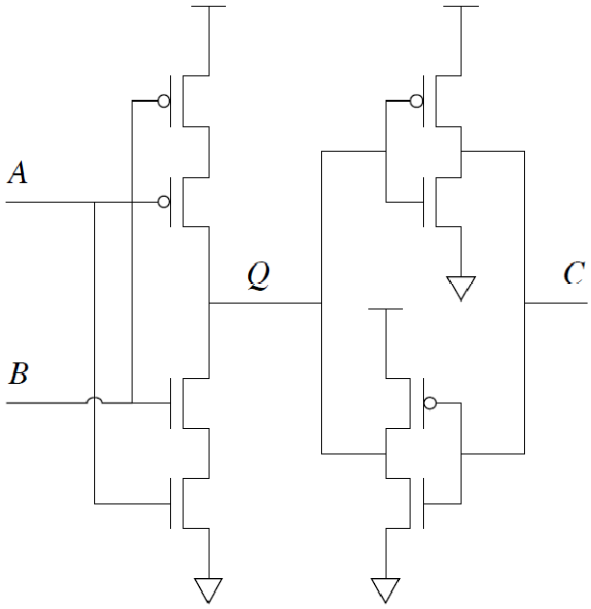


Figure 3.4: "week feedback" Implementation of a Muller C-element.

The Charlie effect appears in the input stage transistors when both N-channel and P-channel transistors must move simultaneously from the cut-off state to the saturation state, or vice versa. This occurs when the events on both inputs of the C-element are close. To understand this phenomenon, we propose an example. Consider a scenario where both *A* and *B* make a 0 to 1 transitions, and *A* changes after *B*. If *A* changes a long time after *B*, then the P-channel transistor controlled by *B* will be in its cut-off region, and the N-channel transistor controlled by *B* will be *fully conducting* as *A* changes.

Furthermore, the node between the two N-channel transistors will be close to ground potential. This allows a relatively fast transition on signal *Q* and therefore on the output *C*. on the other hand, if A changes only slightly after B, then the transistors controlled by *B* will both be *partially conducting* as *A* changes. This result in a greater delay from the transition of *A* to the transition of *C*. Similar effects occur if *A* changes before *B*.

3.3.2 The Drafting effect

Similarly, there is an impact of the time elapsed between two successive output communications on the propagation delay: *the closer the successive transitions; the shorter the propagation delay*. This phenomenon is called the "Drafting effect". This effect is opposite to the Charlie effect which has a tendency to slow the spread of tokens, while the Drafting effect tends to accelerate several tokens in sequence. This effect which appears on the output stage of the Muller C-element is due to the output capacitance of the stage. When a commutation occurs just after the previous one, the output has not enough time to reach neither V_{dd} nor GND before its new commutation which causes a faster transition (i.e. shorter propagation delay).

3.3.3 The Charlie Model

The Charlie model represents the ring stage propagation delay as a function of two variables, *s* and *y* as shown in Figure 3.5 where:

- *s* is the half separation time between input events:

$$\boldsymbol{s = \frac{t_F - t_R}{2}}$$

- *y* is the time between the previous output commutation and the mean input time.

$$\boldsymbol{y = \frac{t_F + t_R}{2} - t_{C^{-1}} = t_{mean} - t_{C^{-1}}}$$

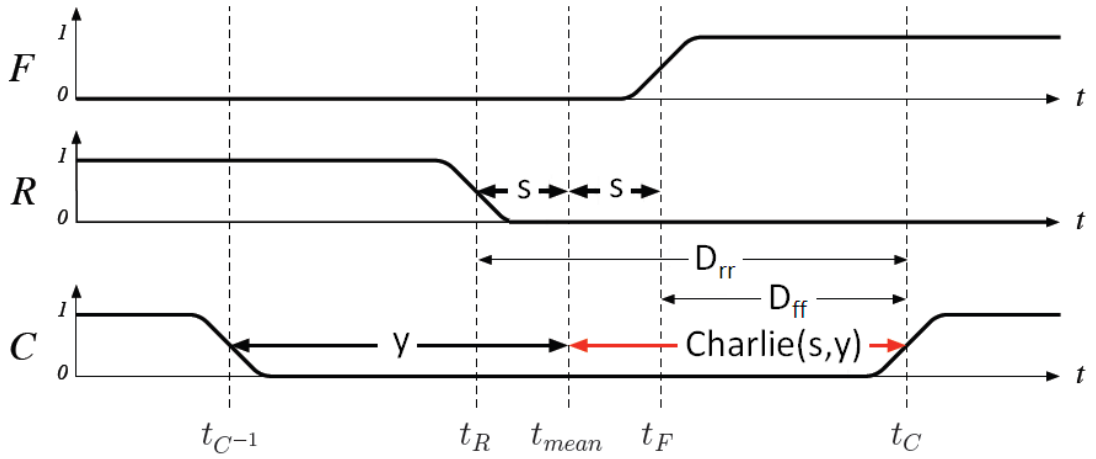


Figure 3.5: A Ring Stage Chronogram.

The Charlie model is also adjusted by a set of five parameters that correspond to the physical characteristics of the stage implementation (dimensions of the Muller C-element and inverter transistors). These parameters are noted as:

- D_{ff} the static forward propagation delay.
- D_{rr} the static reverse propagation delay.
- $D_{charlie}$ the amplitude of the Charlie effect.
- A the duration of the drafting effect.
- B the amplitude of the drafting effect.

The shape of the Charlie model at constant y corresponds to a parabola defined by the asymptote lines $D_{ff} + s$ and $D_{rr} - s$ (see Figure 3.6). Noting that the minimum value of this parabola, which is called "valley" of Charlie model, corresponds to the longest propagation delay where the influence of the Charlie effect is the maximum. Similarly, the shape of the Charlie model at constant s follows the exponential form of charging a capacitance through a resistance (see Figure 3.7). Therefore, the analytical formulation of the Charlie model can be expressed as:

$$\boldsymbol{Charlie}\ (\boldsymbol{s},\boldsymbol{y}) = \underbrace{\boldsymbol{D_{mean}} + \sqrt{\boldsymbol{D_{charlie}^2} + (\boldsymbol{s} - \boldsymbol{s_{min}})^2}}_{\boldsymbol{Charlie}} - \underbrace{\boldsymbol{B}^{-\frac{\boldsymbol{y}}{\boldsymbol{A}}}}_{\boldsymbol{Drafting}} \tag{3.1}$$

with:

$$\begin{cases} \boldsymbol{D_{mean}} = \dfrac{\boldsymbol{D_{rr}} + \boldsymbol{D_{ff}}}{\boldsymbol{2}} \\ \boldsymbol{s_{min}} = \dfrac{\boldsymbol{D_{rr}} - \boldsymbol{D_{ff}}}{\boldsymbol{2}} \end{cases}$$

Finally, the commutation instant t_C can be expressed with respect to the input commutation instants (t_F and t_R) and the Charlie model as:

$$\boldsymbol{t_C} = (\boldsymbol{t_F} + \boldsymbol{t_R})/\boldsymbol{2} + \boldsymbol{charlie}(\boldsymbol{s},\boldsymbol{y}) \tag{3.2}$$

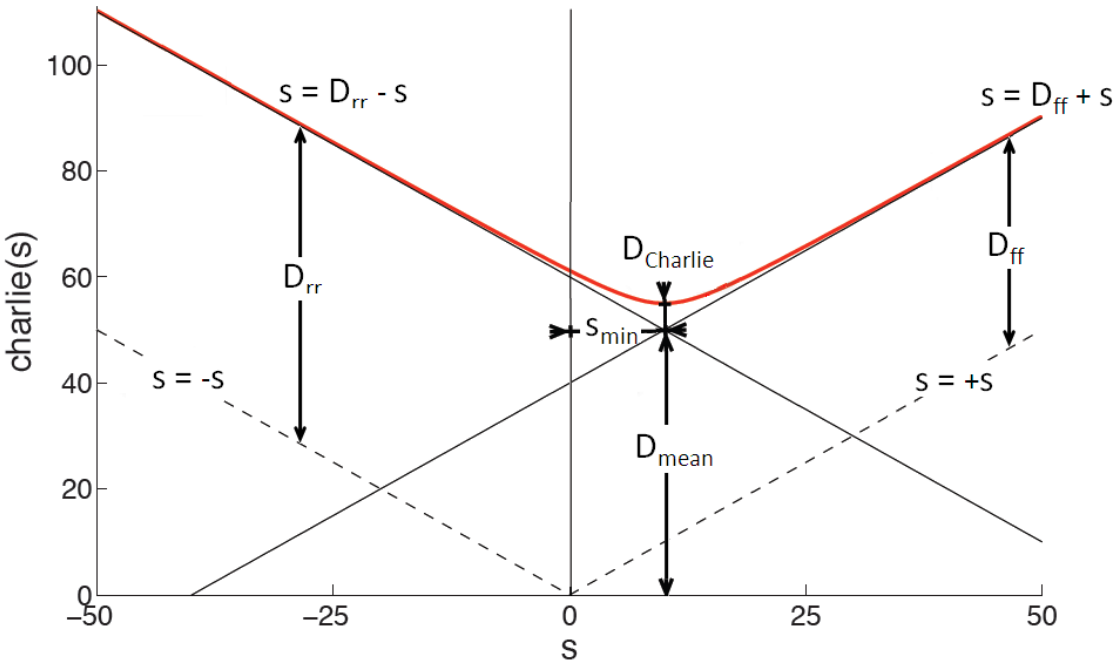


Figure 3.6: Charlie Diagram at Constant *y*.

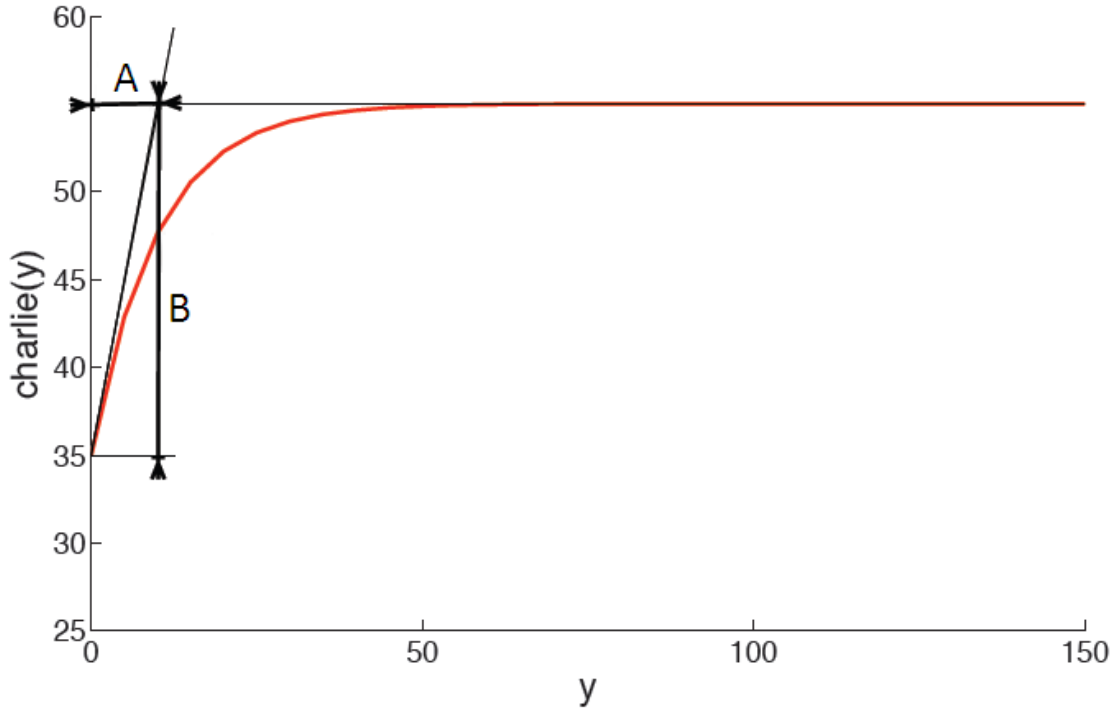


Figure 3.7: Charlie Diagram at Constant *s*.

3.3.4 Timed VHDL Models

Without modeling the Charlie effect and by simulating the same self-timed ring with the same number of tokens and bubbles, but with different spatial token distributions, digital simulation (VHDL Model) shows different steady state waveform. However, Analog simulation shows exactly the same steady state waveform. An explanation of this incorrect behavior of the digital simulation is due to the absence of Charlie effect in the model, [YAH 09]. To confirm this explanation, we simulate an 11-stage ring without modeling the Charlie effect. We first configured it by the following Token/Bubble pattern ("TTTTBBBBBBB"). The steady state output was a burst mode oscillation. When the ring is initialized with ("TBBBBTTBBBT"), the steady state output becomes evenly spaced. This wrongly concludes that the initial spatial distribution of the Tokens/Bubbles could affect the oscillation mode, which is not true. These simulation results are shown in Figure 3.8.

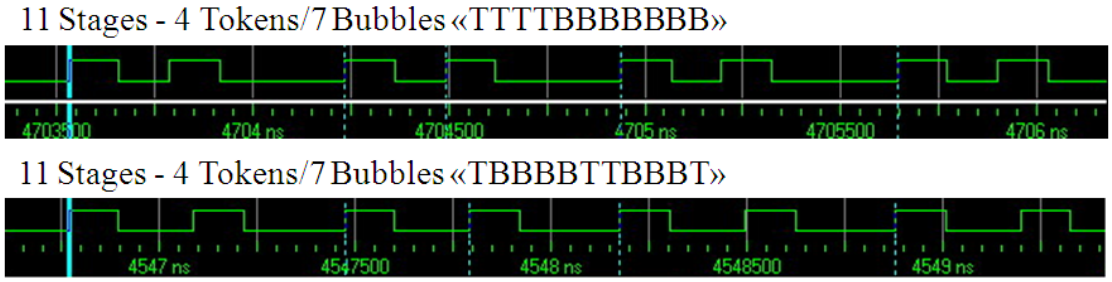


Figure 3.8: VHDL Simulation Results for two Different Configurations of 11-Stages Self-Timed Ring without Charlie Effect.

The same simulation is now performed using Muller gates models which are taking the Charlie effect into consideration. Figure 3.9 shows that they give identical steady state behavior and both are oscillating in evenly spaced mode as it was expected.

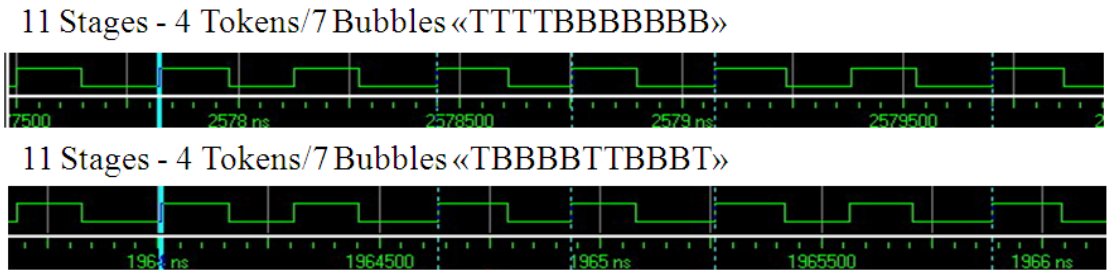


Figure 3.9: VHDL Simulation Results for two Different Configurations of 11-Stages Self-Timed Ring with Charlie Effect.

The conclusion is that including the Charlie effect inside our digital model is mandatory to have a correct behavior of the ring model. As the design and analysis phases need so many simulations, our digital model saves us a lot of time compared to standard analog simulations.

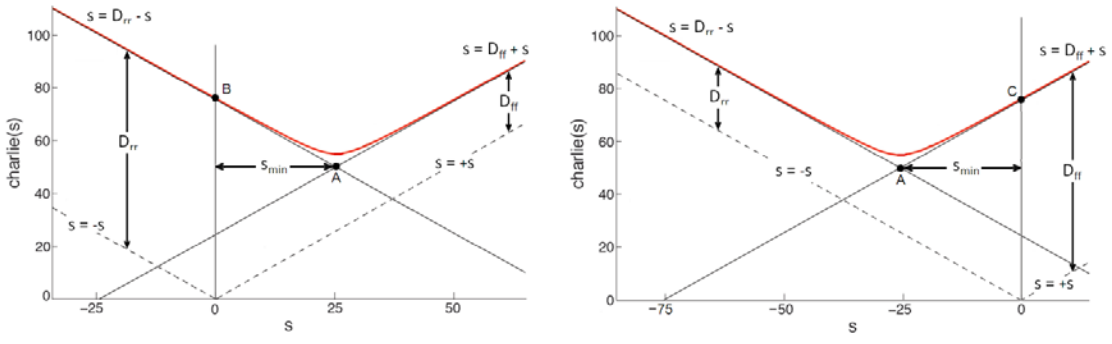
3.4 Oscillation Frequency Calculation of Self-Timed Rings

In this section, we propose a simple methodology for calculating the oscillation frequency of the self-timed ring oscillator. The new method allows us to estimate the oscillation period in function of the stage temporal parameters and the number of bubbles and tokens. In [HAM 08] a behavioral model of asynchronous ring is proposed. The proposed behavioral model offers interesting prospects regarding timing information. It needs to calculate the half separation time between inputs *s*. To do so, it uses state graph

to iteratively catch the timing evolution inside the ring. That seems a bit complex if the goal is only estimating the oscillation frequency. Consequently, it seems very useful to find a simple relation between *s* and the number of bubbles and tokens. In [FAI 04], the authors give a relation between the period, the stage delay and the separation time. In this model the Drafting effect can be neglected because its effect is very small compared to the Charlie effect in the period calculation. With our terminology we can express this relation as:

$$T = 4 \times D + 4 \times |s| = 4 \times (D + |s|) = 4 \times Charlie(s) \qquad (3.3)$$

Where $D = Charlie(s) - |s|$ is the stage delay from the last arriving input (i.e. D_{ff} in Figure 3.5). The period of the self-timed ring oscillators depends on the ratio $R = {N_T}/{N_B}$. For the same ratio, we have the same oscillation whatever the number of stages. This is why we think that there is a direct relation between *s* and *R*. To estimate *s* we can use the Charlie diagram at constant *y* shown in Figure 3.6 at two different operating conditions. First, when the operating point of the asynchronous self-timed ring is in the *token limited* region, where ${N_T}/{N_B} \geq {D_{ff}}/{D_{rr}}$ i.e. $s \leq s_{min}$. Second, when the operating point of the asynchronous self-timed ring is in the *bubble limited* region where ${N_T}/{N_B} \leq {D_{ff}}/{D_{rr}}$ i.e. $s \geq s_{min}$, the two operating conditions are shown in Figure 3.10.



(a) Token Limited (b) Bubble Limited

Figure 3.10: Charlie Diagram at Constant *y* for two Different Operating Conditions.

At point "**A"** shown in Figure 3.10 (a) the ring is oscillating at its maximum frequency, [WIN 01]:

$$R = N_T/N_B = D_{ff}/D_{rr} \quad \text{and} \quad s = s_{min} = \left(D_{rr} - D_{ff}\right)/2 \qquad \textit{Lemma 1}$$

At points B and C shown in Figures 3.10 (a) and (b) respectively:

$$s = 0 \;\rightarrow\; R = 1 \qquad \textit{Lemma 2}$$

Lines (A-B) and (A-C) are applied when ${N_T}/{N_B} \geq {D_{ff}}/{D_{rr}}$ and ${N_T}/{N_B} \leq {D_{ff}}/{D_{rr}}$ respectively. From *Lemma 1* and by dividing *s* by D_{rr}:

$$\frac{2s}{D_{rr}} = \frac{D_{rr} - D_{ff}}{D_{rr}} \qquad \rightarrow \qquad s = \frac{D_{rr}}{2}(1 - R) \tag{3.4}$$

If we prove that Equation 3.4 is correct for point B then we can assume that it is correct for the line (A-B). At point B, $s = 0$ and from Lemma 2, $R = 1$. Substituting $R = 1$ into Equation 3.4, results in $s = 0$ which is correct. This means that Equation 3.4 is correct for the line (A-B). This could be mathematically formulated as:

$$\text{If} \qquad R = \frac{N_T}{N_B} \geq \frac{D_{ff}}{D_{rr}} \qquad \text{then} \qquad s = \frac{D_{rr}}{2}(1 - R) \tag{3.5a}$$

By the same way, from *Lemma 1* and by dividing *s* by D_{ff}, we can prove that:

$$\text{If} \qquad R = \frac{N_T}{N_B} \leq \frac{D_{ff}}{D_{rr}} \qquad \text{then} \qquad s = \frac{D_{ff}}{2}\left(\frac{1}{R} - 1\right) \tag{3.5b}$$

These equations allow us to calculate *s* without going through the behavioral model. Therefore, we can introduce a new function called *Charlie(R)*, which expresses the value of the function of Charlie according to the ratio R. This equation can be obtained by replacing *s*, from Equation 3.5, into Equation 3.1. The result is shown in Equation 3.6.

If $R = \frac{N_T}{N_B} \geq \frac{D_{ff}}{D_{rr}}$ then

$$Charlie\ (R) = \frac{D_{rr} + D_{ff}}{2} + \sqrt{\left(D_{Charlie}^2 + \left(\frac{D_{rr}}{2} \left(R - \frac{D_{ff}}{D_{rr}} \right) \right)^2 \right)} \quad (3.6a)$$

If $R = \frac{N_T}{N_B} \leq \frac{D_{ff}}{D_{rr}}$ then

$$Charlie\ (R) = \frac{D_{rr} + D_{ff}}{2} + \sqrt{\left(D_{Charlie}^2 + \left(\frac{D_{ff}}{2} \left(\frac{1}{R} - \frac{D_{rr}}{D_{ff}} \right) \right)^2 \right)} \quad (3.6b)$$

To show the accuracy of Equation 3.6, we compared the Charlie values extracted from the VHDL simulation which is based on *Charlie (s)* Equation 3.1. These values are plotted against the corresponding *R* in Figure 3.11. The same curve is plotted using *Charlie(R)*, Equation 3.6. The figure shows that the two plots are identical with an error which is less than 1%.

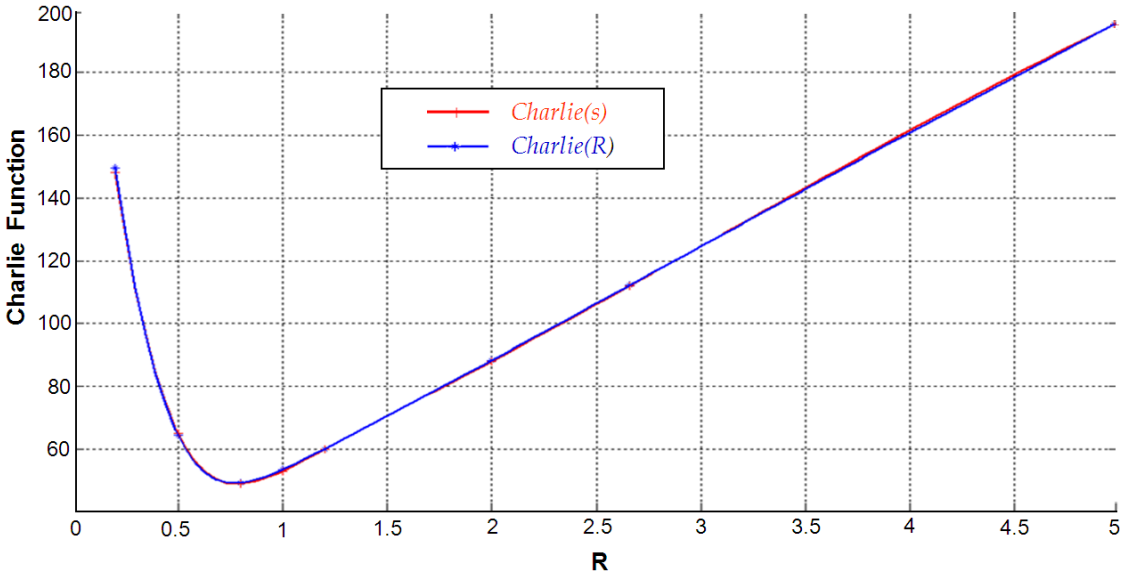


Figure 3.11: *Charlie (R)* against *Charlie (s)* Diagrams with respect to *R*.

The importance of this *Charlie(R)* diagram is that it allows us to have a clear idea about the oscillation period with respect to $R = N_T / N_B$. For designers it is easier, even obvious to interpret the Charlie diagram with the R parameter than the Charlie diagram with the s parameter. In case of the burst mode, s is not the same for all stages and s which is computed by Equation 3.5 is the average value. Equation 3.7 shows how the oscillation period can be calculated using *Charlie(R)*.

$$T = 4 \times Charlie\ (R) \tag{3.7}$$

If we neglect Charlie effect in Equation 3.6, and by substituting the result in Equation 3.7:

$$\text{If} \quad R = \frac{N_T}{N_B} \geq \frac{D_{ff}}{D_{rr}} \quad \text{then} \quad T = 2 \times D_{rr}(R+1) \tag{3.8a}$$

$$\text{If} \quad R = \frac{N_T}{N_B} \leq \frac{D_{ff}}{D_{rr}} \quad \text{then} \quad T = 2 \times D_{ff}\left(\frac{1}{R}+1\right) \tag{3.8b}$$

3.5 Programmable Self-Timed Rings (PSTR)

As explained in the previous sections, Self-Timed Rings give advantages from different points of view, generation of high-resolution timing signals, robustness against process variability, and reconfigurability. As a result, our programmable oscillator will be based on a Programmable Self-Timed Ring (PSTR). One example of a Self-Timed Ring is depicted in Figure 3.2.

3.5.1 PSTR Architecture

There are many possible ways to control the frequency of the self timed ring. In this section, two strategies using architectural solutions are described. *Strategy1* is to change the number of tokens and bubbles inside the ring. That would change the value of the term R in Equation 3.4. We designed a Muller gate which has a Set/Reset control, see Figure 3.12. Using this gate, an asynchronous self-timed ring in which we can dynamically insert tokens and bubbles is designed. This ring has the same architecture as the one in Figure 3.2 except that it is based on Set/Reset Muller gates.

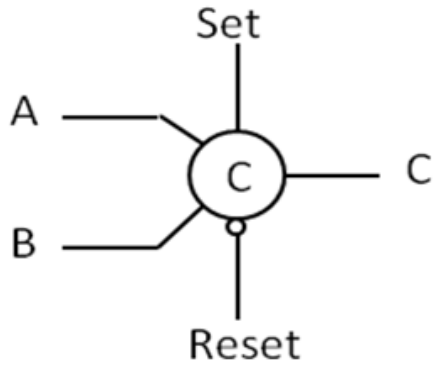


Figure 3.12: Muller Gate with Set/Reset.

As an example, for a ring with five stages (N=5), we can insert “BTTTT” by controlling the Set/Reset of the stages so they are initially loaded by “11010”. That configuration would produce the minimum frequency of this ring. If we change the Set/Reset so that the ring is initialized by “11110”, which means “BBBTT”, this ring will oscillate at a higher frequency. This example shows how the ring frequency can be controlled by changing the N_T over N_B ratio. By using Strategy1, the Ring can achieve high frequencies. Since the programmability overhead is limited to the addition of the Set/Reset controls. It is not significantly affecting the speed. However, using fixed number of stages limits the number of possible frequencies. For example, 12 stages gave us 5 different frequencies. This makes the frequency step quite coarse. If the objective is to produce finer frequency steps, we propose Strategy2.

Strategy2: in this strategy, not only the Token/Bubbles ratio is controllable but also the number of stages. Figure 3.13 shows the block diagram of our programmable self-timed ring. It is composed of stages based on the Muller gates with Set/Reset. The initialization of the ring is controlled by the Token Control Word “TCW”. This is defining the number of Tokens and Bubbles inside the ring. By means of the TCW the frequency of the ring can be programmed. To be able to change the number of stages, a multiplexer is placed after each stage. These multiplexers are controlled by the Stage Control Word “SCW”. SCW is controlling the number of stages inside the ring. The design starts with the maximum number of stages “N”. Then, by using SCW we can remove up to N-3 stages. If the stage is enabled, the equivalent bit in the SCW is set to

zero. In Figure 3.13, suppose that all stages are enabled. That means SCW bits are all zeros. That will make all the multiplexers select input “a” which is connected to the preceding C-Element (STG1 is connected to STG2, STG2 is connected to STG3 etc...). Regarding the last stage “STGn”, its acknowledgment is connected by the tri-state buffer “T1” to the output of STG1. That closes the ring and makes it functioning correctly. Now, suppose that STG1 has to be removed. Then SCW0 is set to “1” and the other bits are zeros. Two things are to be done. First, the “Req” of STG2 needs to be connected to the output of STGn. Second, the signal “Ack” of STGn needs to be connected to the STG2 output. As SCW0 is “1”, Mux1 will connect input “b” to the output. That would connect STGn out to the STG2 Request signal. In addition, SCW0 will force “T1” to high impedance. The AND gate “D1” outs “1” which enables “T2” and connects STG2 out to STGn Ack. By this scenario, the ring contains N-1 stages. By means of SCW and TCW, the self-timed ring in Figure 3.13 can dynamically be programmed to have a range of stages between N to 3, and whatever configurations of Tokens and Bubbles.

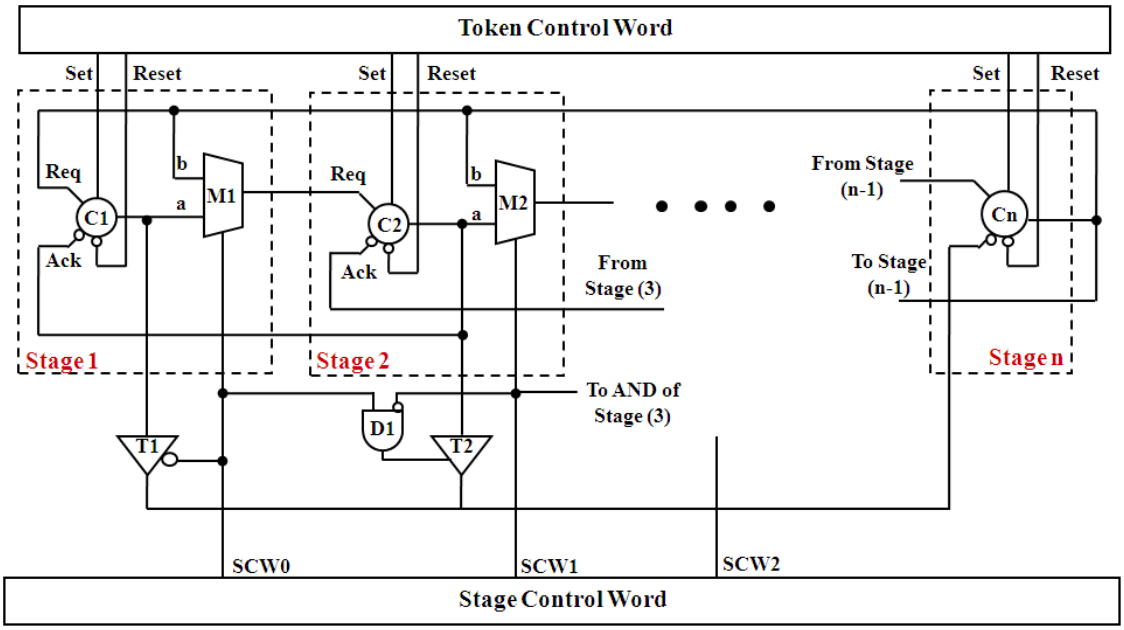


Figure 3.13: Programmable Self-Timed Ring.

Regarding the delays of the ring stages, the design is very modular and balanced. The forward delay D_{ff} has the Muller gate and the Mux. The reverse delay D_{rr} has the

Muller gate and an inverter. Only STGn has a bit longer D_{rr} as the delay of the tri-state buffer is added. This will not affect the ring behavior especially this stage is dealt differently during the physical implementation. The set of AND gates at the tri-state buffer control are not contributing to the delay as they are prepared once, at programming phase, and they never change until the ring is reprogrammed. This structure provides a Self-Timed Ring which has an interesting range of frequencies. More details about the implementation as well as performance figures are shown in Section 3.7. In Strategy2, the PSTR gives finer frequency steps, however the programmability overhead from speed and power point of views is higher than Strategy1. Adding a limited number of multiplexers gives a nice solution. For example, for a 12 stages two multiplexers can be added, one after stage 1 and one after stage 7. That gives a PSTR with possibilities of 12, 11 and 5 stages. This compromise gives nice results from all points of view. This hybrid solution is denoted as *Strategy3*.

3.5.2 PSTR Design Flow

In this section, we present a design flow for PSTR oscillators, see Figure 3.15. According to the specifications, thanks to the frequency calculation Equation 3.8 and according to D_{rr} (defined by the targeted CMOS implementation); the design flow helps the designers to choose the suitable PSTR architecture (number of stages N_{stages}, bubbles N_B and Tokens N_T) which allows approaching the target oscillation frequencies F_{tar}.

In this design flow, we assume that the self-timed ring operating point always exists in the token limited region, and that the maximum allowable N_{stages} from which we start our design is 20, and then it will be changed during the design. This number is chosen according to the circuit area and power consumption limitations; see Chapter 7 for more details on the design of PSTR for MIPS R2000 on 45nm STMicroelectronics CMOS technologies. As the thesis is targeting an optimum design in terms of area and power consumption on the 45nm CMOS technology, designing a PSTR with this limitation on N_{stages} will always give us frequencies within the GHz range (i.e. above 0.5 GHz), see Section 3.7. Consequently, if the design is targeting lower frequencies (i.e. below 0.5 GHz), we have to choose between two solutions. First solution is to increase

the allowable N_{stages}. This will accordingly increase the area and power consumption of the PSTR. For example in order to get a frequency of 60 MHz with D_{rr} = 73 ps on the 45nm CMOS, and according to Equation 3.8a we need to have a PSTR with at least 113 stages, which is impractical. This guides us to the second solution, which is the addition of external D Flip-Flops at the self-timed ring output, see Figure 3.14. This will divide the asynchronous self-timed ring output F_{osc} by $2^{No.\ of\ F.Fs}$. Subsequently, we can complete our design using F_{osc} which is now applicable with the N_{stages} limitation.

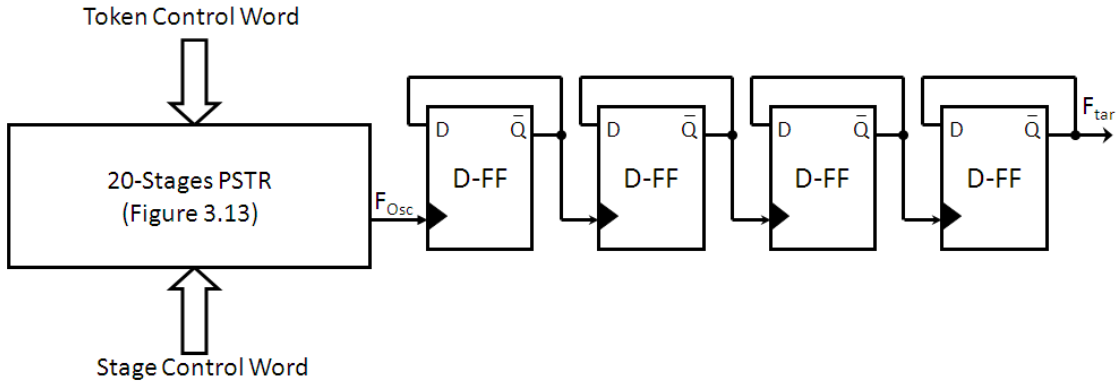


Figure 3.14: PSTR with External D-F.Fs for Targeting Low Frequencies.

The main idea of the design flow shown in Figure 3.15 is to keep looking on the most suitable values of N_T, N_B and N_{stages} that achieve the target frequency F_{tar}. This is done by comparing the ratio N_T/N_B calculated from Equation 3.8a with our assumption N_{T_d}/N_{B_d}. Noting that in the case of targeting low frequencies we have to use a set of D-F.Fs. In this case, the target PSTR frequency F_{Osc} will be equal to F_{tar} multiplied by $2^{No.\ of\ F.Fs}$. If N_{T_d}/N_{B_d} is not equal to N_T/N_B, then we enter into a loop to adapt these two ratios to be as close as possible. As long as $N_T > N_B$ (token limited region), we keep minimizing N_{T_d} by 2 while $N_{B\_d} = N_{stages}-N_{T\_d}$ and each time we compare the two ratios together. If we could not achieve that $N_{T\_d}/N_{B\_d} \approx N_{T\_d}/N_{B\_d,}$ then we resort to adapting N_{stages}. This is done by reducing N_{stages} by 1, and selecting the maximum allowable even number for N_{T_d}, while $N_{B\_d} = N_{stages} - N_{T\_d}$. Once N_{T_d}/N_{B_d} is approximately equal to N_{T_d}/N_{B_d} then $N_T = N_{T\_d}$, $N_B = N_{B\_d}$ and $N_{stages} = N_T + N_B$, so the design is complete.

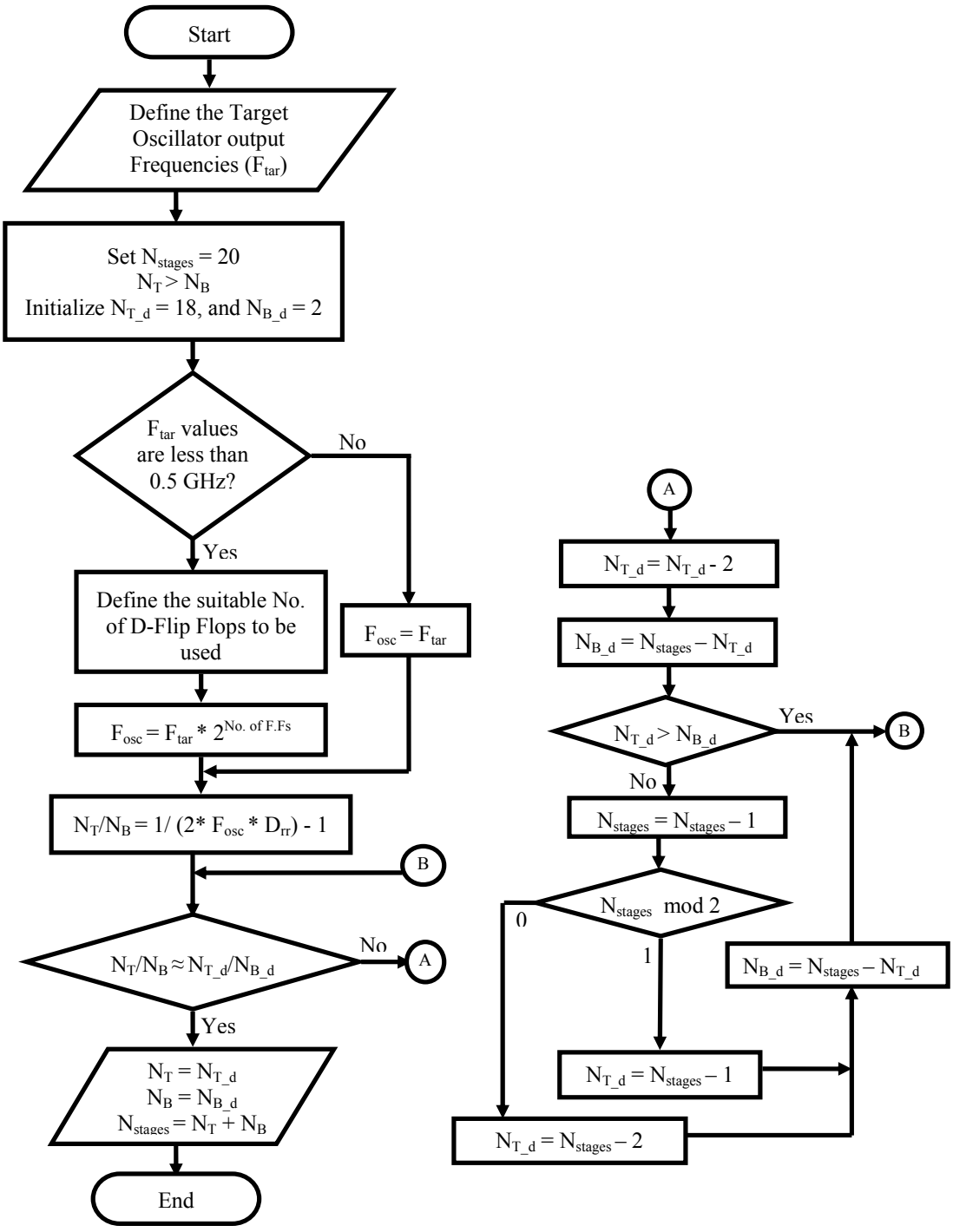


Figure 3.15: Design Flow for PSTR Oscillators.

3.6 Programmable Stoppable Oscillator (PSO)

Note: since silicon data are confidential, no exact delay-values are shown in this section. However, the complete performance figures are given in Section 3.7.

Based on the PSTR proposed in Section 3.5, a fully Programmable-Stoppable Oscillator, "PSO", is designed. The main goal of our design is to provide a communication protocol between the processor and the PSO so that:

- The processor can Pause/Reprogram the PSO clock output.
- The clock is paused when switching from one frequency to another and the length of the pausing period is controlled by the processor.
- While taking into consideration Metastability and racings, the Start/Stop clock-periods are always complete without any truncated periods or glitches.

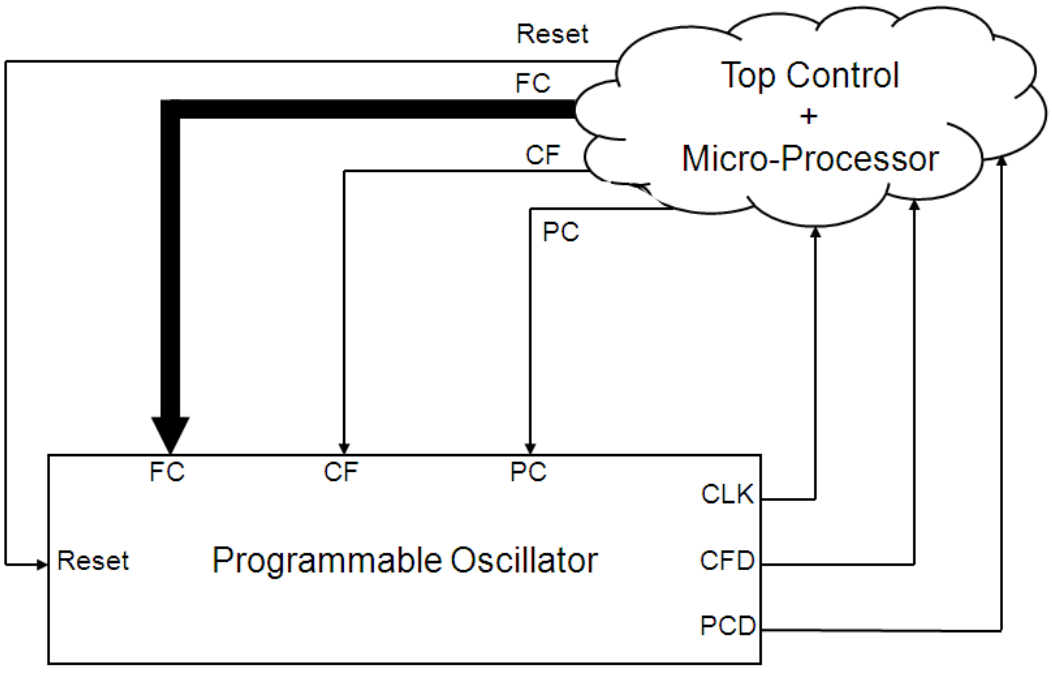


Figure 3.16: The Interface between Micro-Processor and PSO.

Figure 3.16 shows the connections between the processor and our *PSO*. The *PSO* receives two commands from the processor. Change Frequency, "*CF*", which is putting the *PSO* in change frequency mode, "Mode1", where it uses the Frequency Code, "*FC*", to program the *PSTR* by the required frequency. The other command is Pause Clock,

"*PC*", which puts the *PSO* in a Pause mode, "Mode2", where the output clock is paused until it is released, with the same frequency, by the processor. The communication protocol is as follows. In Mode1, the processor sends *CF* high and the *FC* is set to the appropriate code for the required frequency. When the *PSO* is ready to change the frequency, it pauses the clock and lowers the Change Frequency Done signal, "*CFD"*. The clock is paused until the *CF* command is lowered by the processor. When *CF* is lowered, the clock is released with the new frequency and the *CFD* is returned to high. By the same scenario in Mode2, the processor raises *PC* command when the clock is needed to be paused. The *PSO* pauses the clock and sets its Pause Clock Done signal, "*PCD"*, to zero. Whenever the processor lowers the *PC* command, the *PSO* raises its *PCD* out and continues sending clock output with the last frequency before pausing.

By means of the upper handshaking protocol between the processor and the *PSO*, the processor has a full control on the pausing period between two different frequencies. The importance of that, from our point of view, is to give the processor (or its upper controller) the control on the time between two frequencies. In this way different techniques can be applied on the processor, (task rescheduling, DVS, HW reconfiguration …), without caring about the time needed by these techniques.

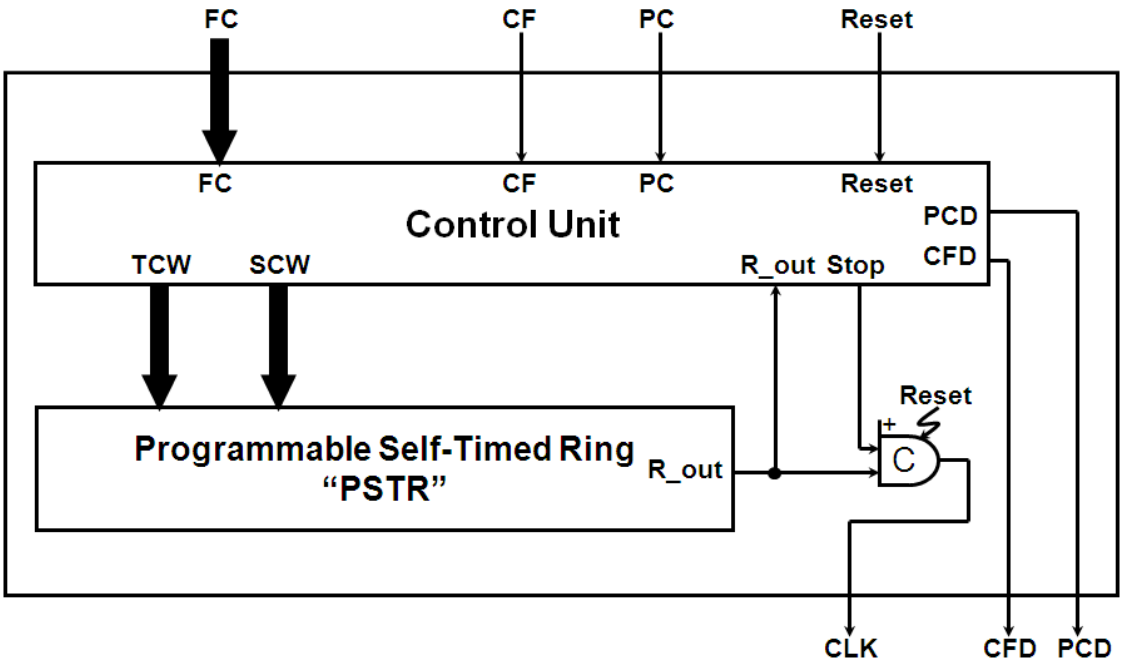


Figure 3.17: Programmable/Stoppable Oscillator.

In Figure 3.17, the details of the *PSO* are depicted. It is composed of a Control Unit, "*CU*", and the *PSTR*. The *CU* outs the *TCW* and the *SCW* which are determining the number of Tokens/Bubbles and the number of stages respectively. The *CU* is clocked by the output of the *PSTR*, "*R_Out*". The *Clk* out which is sent to the processor is the output of the dissymmetric Muller gate "C1". The output "*Stop*" from the *CU* is controlling whether *R_Out* is connected to *Clk* or not. When *Stop* is high *R_Out* is connected to *Clk*. When it is low, any change in *R_Out*, including truncated periods and glitches, is filtered by the Muller gate. Figure 3.18 shows the details of the *CU*.

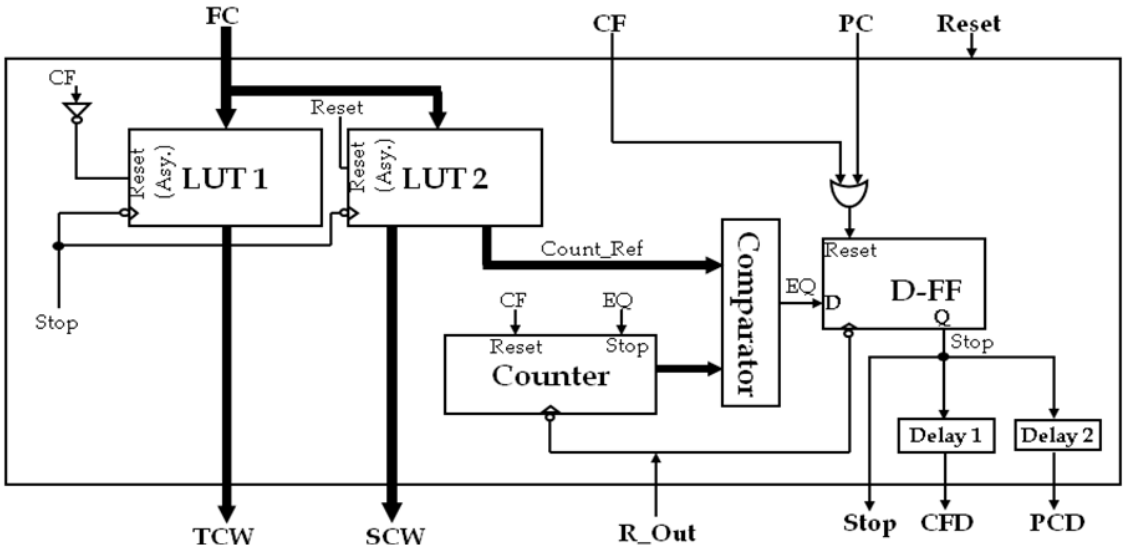


Figure 3.18: PSO Control Unit.

In the two lookup tables, LUT1 and LUT2, the map of the frequency code to the corresponding *TCW* and *SCW* is stored respectively; this map of the frequency code is determined by using the model presented in Section 3.4 for all Token/Bubble possible configurations. In LUT2, there is another stored value per each frequency; it is the number of clocks needed by the *PSTR* to reach its steady state, where it's calculated using the simulation. This output is denoted by *Count_Ref*. *Count_Ref* and the output of stoppable binary counter is compared by a comparator. The binary counter is clocked by *R_Out* to count the clocks during the transient state. The comparator outs one, "*EQ*=1", when its inputs are equal and vice versa. *EQ* is latched by a D-FF which is clocked by *R_Out*. The output of the D-FF is the Signal "*Stop*". This signal is delayed by two matching delays, Delay1 and Delay2, before sent to the processor as *CFD* and *PCD*

signals respectively. LUT1 have asynchronous Reset. However, the counter and the D-FF have synchronous resets. The Stop control of the counter is asynchronous. During normal operation, *CF* and *PC* are set to zero. The *Stop* signal is high, which makes *R_Out* connected to *Clk* by the Muller C1 in Figure 3.17. The processor could issue one of two commands *CF* for Mode1 and *PC* for Mode2.

Mode1: the processor sets *CF* to high and sends the appropriate code on *FC*. With the next negative edge on *R_Out*, the D-FF and counter outputs are set to zeros. That lowers the *Stop* signal and isolates *R_Out* from *Clk*. In addition, LUT1 and LUT2 out the new *TCW* and *SCW* for programming the *PSTR* by the new frequency. LUT2 outs the new *Count_Ref* too. Delay1 delays *Stop* by equivalent time to the access time of the LUTs plus the required time for programming the *PSTR*. This guaranties that the *CFD* is not lowered before the *PSTR* is properly programmed. When *CFD* is lowered, the processor lowers *CF* whenever it is ready to receive the new *Clk*. When *CF* is lowered, LUT1 receives a Reset signal. That forces *TCW* to be all zeros and allows the *PSTR*, Figure 3.13, to start oscillation. The Counter counts the clocks on *R_Out*. When they are equal to *Count_Ref*, the comparator sets *EQ* to high. When *EQ* is high, the counter is stopped and counts no more edges. With the next negative edge on *R_Out*, the D-FF latches *EQ* and sets *Stop* to one. That enables the Muller C1, in Figure 3.17, and connects *R_Out* to *Clk*. Now the *PSO* returns to its normal operation and the *PSTR* output is connected to the processor.

Mode2: the processor sets *PC* to high asking for pausing the *Clk* output. The D-FF receives a Reset signal. With the next negative edge on *R_Out*, *Stop* is set to zero and isolates *R_Out* from *Clk*, which is stalled at zero. Delay2 delays the *Stop* by a time equal to the delay of the Muller C1, Figure 3.17. After this delay, the *PCD* is sent to the processor. Whenever the processor, or its upper controller, wants to unpause the *Clk*, it lowers the *PC* signal. With the next negative edge on *R_Out*, the D-FF latches *EQ* again which is still one as Counter output and *Count_Ref* are still equal. That sets *Stop* to one which connects *R_Out* to *Clk* and the clock of the processor is unpaused.

The main difference between Mode1 and Mode2, is that in Mode2 *TCW* and *SCW* are not changed. As a result the LUTs accessing have a fixed output value and so no pulses are skipped by the counter. That makes the processor able to pause and unpause in a higher rate compared to the possible rate for programming and re-programming. In Mode1, the minimum delay between two requests on *CF* should be longer than the access time of the LUTs plus the *PSTR* programming time plus the time to skip the transient state clocks (by the counter). In contrast, the minimum time between two requests on *PC* is equal to the sum of the OR gate, the D-FF and the Muller gate delays. This sum is lower than the period of one clock of the *PSTR* maximum frequency. As the ring size could be long, we restrict the CU to have very low growth with respect to the ring size. In Figure 3.18, the HW which will grow with the Ring number of stages is the LUTs. For each extra stage, LUT1 has two more bits and LUT2 has 1 extra bit. This growth is really very limited.

Racing and Metastability: any racing or Metastabilty cases could affect the operation of the *PSO*. In Mode1, there are two possible scenarios for Metastability. First, when *CF* is set to one and *FC* is changed to the new code. Since the processor is synchronous and sends these controls on a negative edge of the clock and because these inputs will not affect the control unit until the next negative edge, these controls have a complete clock period to get ready at the *CU* inputs. This delay reduces the probabilities of Metastability. The second possible Metastability scenario is when *CF* is lowered by the processor; especially this is done completely asynchronously. However, the handshaking protocol ensures that *CF* can not be lowered before the *PSTR* is stalled by the *TCW*. In this way no possible Metastability or racing can appear.

In Mode2, same two scenarios are possible. Firstly when the processor sets *PC* to high, similarly as in Mode1, this scenario is safe. Secondly when the processor or its upper controller lowers the *PC*, this case could have Metastability if the Reset input at the D-FF is deasserted during a negative edge at *R_Out*. However, the probability of this Metastability is very low due to the short delay of the reset logic path.

The *PSO* is implemented on STMicroelectronics 45nm technology, the implementation details are discussed in Section 3.7. The delay information is extracted from the physical design and inserted in our timed VHDL model. By using the digital flow, we could quickly get various simulations. Thanks to including Charlie and drafting effects inside the digital model which gave us accurate results, very close to the analog simulation results. Figure 3.19 shows one example where the *PSO* is requested to switch from low frequency to a higher one. After that, the *PSO* is requested to pause the clock.

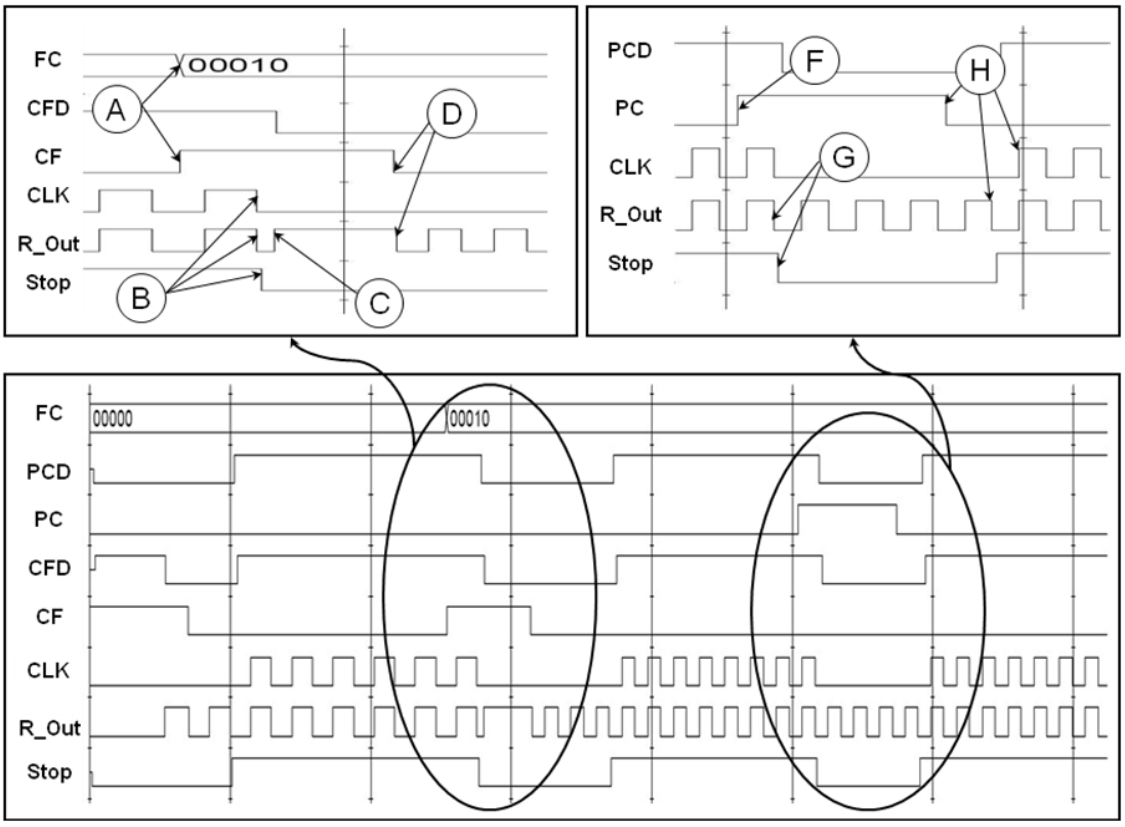


Figure 3.19: Timing Diagram of the PSO.

In Figure 3.19 at point "A", the *CF* and the new *FC* are sent. Nothing happened till the next negative edge on *R_Out*, at point "B". After this edge, the Stop signal is lowered. At point "C", it is clear that *R_Out* has a truncated clock due to the new programming pattern. However, this truncated clock never appears on Clk. CF is lowered at point "D", which makes *R_Out* to start oscillating with the new frequency and the counter starts counting the transient clocks. On the other hand, at point "F" the *PC* is sent. On the next negative edge, the Stop signal is lowered and *Clk* is stalled at zero (see point

"G"). Please note that *R_Out* is continuing oscillating with the same frequency. At point "H", the *PC* is lowered. The next negative edge on *R_Out*, forces the D-FF to latch "*EQ*=1" and set the Stop to one. Consequently, the *Clk* output starts to oscillate from the following positive edge. The design is extensively tested against many scenarios especially for the communication protocol between the processor and the *PSO*. All test cases show a correct behavior for the design.

3.7 Implementation and Results

The design presented in Sections 3.5 and 3.6 is implemented using STMicroelectronics 45nm CMOS Technology. We use our TAL "TIMA Asynchronous Library" and the STMicroelectronics 45nm standard libraries for the physical implementation. CADENCE design flow is used for the design, simulation, layout and post layout simulation.

In our design, a Muller gate with Set/Reset, as depicted in Figure 3.12, is implemented. Then a complete circuit of the ring with 11 stages as in Figure 3.13 is built and tested. The Muller gate at $Stage_n$, needs to be sized so that it can derive the stage multiplexers. As a result, we could have a well shaped clock output, as shown in Figure 3.20.

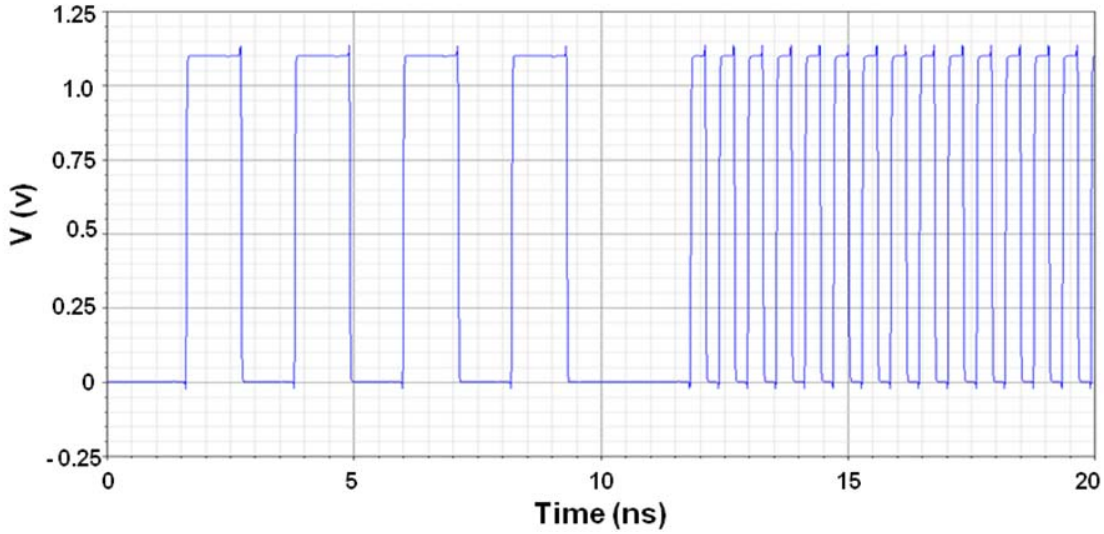


Figure 3.20: Analog Results of the PSO.

3.7.1 Analog Results

As explained in Section 3.5, we have two main strategies to control the frequency of the proposed PSTR. Table 3.1 indicates the main features of each of them. For fair comparison of power consumption, all strategies are tested in a configuration of 11 stages, 10Tokens/1Bubbles. In this case, the frequency of operation is very close in all strategies due to its dependency on D_{rr} which is almost equal in the three strategies.

Table 3.1: Results for Different Programming Strategies.

	Strategy 1	**Strategy 2**	**Strategy 3**
Frequency Range	500 MHz – 3 GHz	400 MHz – 1.7 GHz	450 MHz – 2 GHz
No. of Frequencies	5	13	9
Step Size	Irregular	100 MHz	Irregular
Static Power	8.7 nW	37.5 nW	15.94 nW
Dynamic Power (for 1 Bubble)	63.68 µW	145 µW	82.3 µW

In **strategy 1**, a ring with *11* stages is implemented, where its output frequency is controlled by only changing *TCW*. Its max oscillation frequency is 3 GHz. It has a few numbers of possible frequencies due to its coarse and irregular frequency-step size (300 to 700 MHz). It has the smallest static and dynamic power consumption, as it uses minimum hardware.

In **strategy 2**, a ring with 11 stages is implemented, where its output frequency is controlled by changing the TCW/SCW. On one hand, it provides a finer-regular frequency step (around 100 MHz). On the other hand, maximum frequency is decreased, that was expected, due to the delays added by the multiplexers. It has the largest static and dynamic power consumption, as it is implemented with more hardware.

Regarding **Strategy 3,** which is a hybrid between the first two strategies, we build it using 12 stages, and 2 multiplexers are used to implement a ring with possibilities of 12, 11, 5 stages. Strategy 3 is a compromise between strategy 1 (No multiplexers, few frequency steps, and Min. power consumption), and strategy 2 (Max. No. of multiplexers, Max. No. of frequencies, and Max. power consumption).

In strategy 2, there are 25 different configurations of frequencies with TCW and SCW. In spite of that, we get only 13 frequencies. This is due to two mean reasons. Firstly, some frequencies are repeated within these possible configurations. For example, for 10 stages with $N_T/N_B = 4/6$, we get the same frequency as for 5 stages with $N_T/N_B = 2/3$, which is 1.4 GHz. This is due to the proportionality of the output frequency of the PSTR with the ratio N_T/N_B. The second reason for this is the existence of some burst mode outputs. Within this range, we have 10 frequencies with 50% duty cycle and 3 with 30% duty cycle. A 50% duty cycle is mandatory only in double-edge trigger applications [BUI 06]. As a conclusion we have 3 different strategies for frequency control that we can choose between them, according to the requested application.

Figure 3.20 indicates how the output frequency changes by changing TCW/SCW configurations. We get the lower frequency in the left hand side of the graph with 10T, and 1B. While the higher frequency in the right hand side is obtained with 6T and 5B.

3.7.2 Frequency vs. Supply Voltage

The time response of the PSTR against its supply voltage is an interesting feature. Generally speaking, the speed of asynchronous circuits, including self-timed rings, can be naturally controlled using the supply voltage. Our implementation for a PSTR using Strategy1 is tested against the supply voltage change; results are shown in Figure 3.21. The PSTR is configured so that it oscillates on its maximum frequency. Its supply voltage is changed from zero to 1.1V. The ring could not oscillate under 0.5V. Starting from 0.8V, it shows linear change in its frequency till it reaches the maximum frequency at 1.1V. This behavior enables us to use the voltage for generating more frequencies and/or fine tuning the final oscillation. However, we did not completely include this feature in our chip as we need more work on voltage regulator and level shifters.

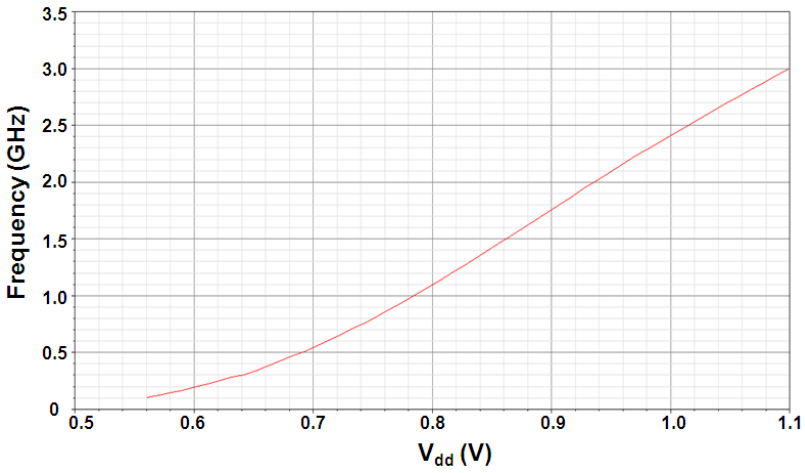


Figure 3.21: PSTR Output Frequency Change with respect to the Supply Voltage.

3.7.3 Sensitivity to Process Variability

In order to measure the effect of the process variability on the proposed ring, Monte-Carlo simulation (1000 iterations) of strategy 1 with 12 stages, 6T/6B configuration had been performed with Cadence. As shown in Figure 3.22a, for a Die-To-Die variability, we have an average frequency of 2.7 GHz, and a standard deviation of 205 MHz. This result indicates that we have a process variability effect on the clock period of 7.6%. For Within-Die variability, simulation results give us an average value of 2.6 GHz and a standard deviation of 25 MHz, which means a process variability effect on the clock period of 1%, see Figure 3.22b.

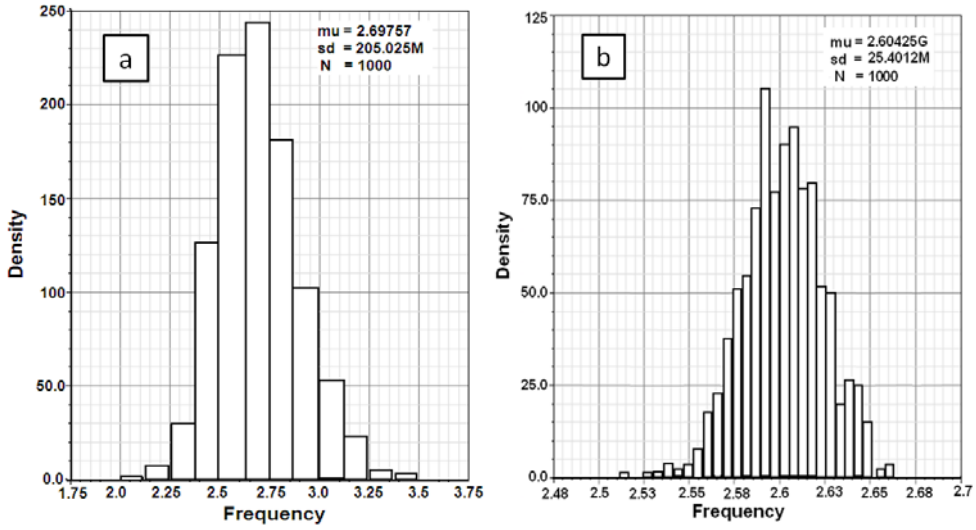


Figure 3.22: Process Variations of PSTR (a) Die-to-Die (b) Within-Die.

3.8 Conclusions

This chapter addressed the problem of designing programmable/Stoppable Oscillator targeting the applications which need a frequencies in the range of 3 GHz to 400 MHz. Self-Timed Ring is chosen as the core of the oscillator because of its reported advantages with respect to many points of view (programmability, accuracy, robustness against process variability …). A new methodology for calculating the oscillation frequency of Self-Timed Rings is proposed. By including Charlie/drafting effects to our VHDL models, we showed the possibility of using digital simulation to accurately model and simulate the Self-Timed Ring behavior.

By using two main different strategies, programmability is introduced to Self-Timed Rings. The two proposed strategies are simple architecture-based solutions. These strategies show high efficiency and flexibility while choosing the ring frequency. Based on the proposed programmable Self-Timed Ring, a complete Programmable/Stoppable Oscillator is designed and implemented. An efficient handshaking protocol between the processor and the oscillator is used to insure a proper switching from one frequency to another. The Oscillator shows glitch free and no truncated clocks at its output. Metastability scenarios are investigated and the oscillator shows no metastability when changing frequency and one possible scenario when pausing the output.

The proposed architectures are physically implemented using STMicroelectronics 45nm CMOS technology. For the different programming strategies, the implemented chip is characterized for its speed, power consumption and sensitivity to process variability. The chip shows a wide range of frequencies with regular and fine frequency step. By using Monte-Carlo simulation of 1000 iteration, the chip shows less than 1% process variability effect on the clock period due to Within-Die variability and around 7.5% process variability effect on the clock period due to Die-To-Die variability.

The effect of changing the power supply on the chip is studied as well. The chip shows very linear changes in its frequency with respect to the variations in its power supply. However, in the current version of the chip, this strategy is not fully implemented.

This design shows how Self-Timed Rings can be efficiently used to implement a Programmable/Stoppable Oscillator. The implemented chip shows High-speed, Low-Power, Low-Process Variability of the generated frequency, Wide-Range with Regular and Fine frequency step output. To the best of our knowledge, this chip is the first realization of a programmable Self-Timed Ring.

Chapter 4

PSTR Case Study in a GALS System

4.1 Introduction

As the number of processing or functional components in an on-chip system becomes larger and faster. Moreover, with the growing number of system components, the functionalities of systems components also become largely different from each other. It means that an on-chip system may include different processors for different computation tasks, varied hardware accelerators for varied functions, and various interface controllers for various peripheral devices. Therefore, these heterogeneous system components have different clock frequencies according to the tasks that they are handling. When integrating all heterogeneous components into an on-chip system, coordinating different clock domains is a challenge. This Chapter starts with stating the main challenges a designer could face when dealing with multi clock GALS systems. Then, it presents a comparison between different techniques of synchronization. After that, a new clock synchronization scheme is proposed. This clock synchronization scheme is based on the use of the PSTR presented in Chapter 3. In this scheme each synchronous module has both an incoming and an outgoing clock signal, which have been obtained by opening the PSTR oscillator module. Since these clock signals also behave as handshake signals, handshake circuits can be used to synchronize the clocks. Finally, the idea of synchronizing a point to point GALS system is generalized to a multipoint GALS interconnection, where three different topologies were presented.

4.2 Multi Clock Challenges and GALS Scheme

From the viewpoint of a chip design, as addressed in [OLS 99], for large high-speed globally synchronous systems, designing the clock distribution net becomes a troublesome task because of the problems caused by clock skew, by growing die sizes and shrinking clock periods. At the same time, the power consumption is increasing

tremendously because the working clock frequency driven by demanding applications is getting higher in the scale of Gigahertz. Therefore, one solution of the challenges mentioned above is to enable different processing or functional system components to work at their own clock rates. Thus, the following challenge that a SoC designer needs to handle is how to integrate the clock independent components into one system. In this situation, the Globally Asynchronous Locally Synchronous (GALS) scheme is proposed to solve the system integration challenge, [CHA 84]. The basic idea of applying GALS scheme into on-chip systems is to partition the system into several independently clock domains that communicate with each other in an asynchronous fashion.

GALS design style holds the promise of combining the advantages of both synchronous and asynchronous operation, [CHA 84]. Our GALS employs a self-timed communication scheme between coarse-grained circuit blocks and combine the following features:

1. All major modules are designed in accordance to proven synchronous clocking discipline.
2. Data exchange between two modules strictly follows a full handshake protocol.
3. Each module is allowed to run from its own local clock frequency (and supply voltage), making scaling far more convenient than with the standard synchronous approach, which in accordance contributes to power savings.
4. All asynchronous circuitry necessary for coordinating the clock-driven with the self-timed operation is confined to “self-timed wrappers” arranged around each clock domain.
5. The GALS architecture can mitigate the impact of process and temperature variations, because a globally asynchronous system does not require that the global frequency was dictated by the longest path delay (the critical path) of the whole chip. In this case, each clock-domain frequency is only determined by the slowest path in its domain, [MAR 05].

The methods and challenges of designing a GALS on-chip network will be presented in the following two sections, with a case study for the application of the PSTR (presented in Chapter 3) in synchronizing data communication within a GALS system.

4.3 Data Synchronization in GALS Systems

In a GALS system, each synchronous part, usually referred as clock domain, operates with its own clock signal. The different domains are mutually asynchronous as they run at different clock frequencies, see Figure 4.1. Therefore synchronization remains an issue either the clock domains work with different frequencies, or with the same central clock frequency. This is so, because the communications among clock domains demand harmonization of the different clock phases in order to guarantee that data is reliably transferred among clock domains. The term, synchronous domain, used in this thesis refers to the group of design blocks which work under the dictation of clock signals in a SoC, while, the term of asynchronous domain refers to the group of blocks which work in a self-timed manner without any clock signal.

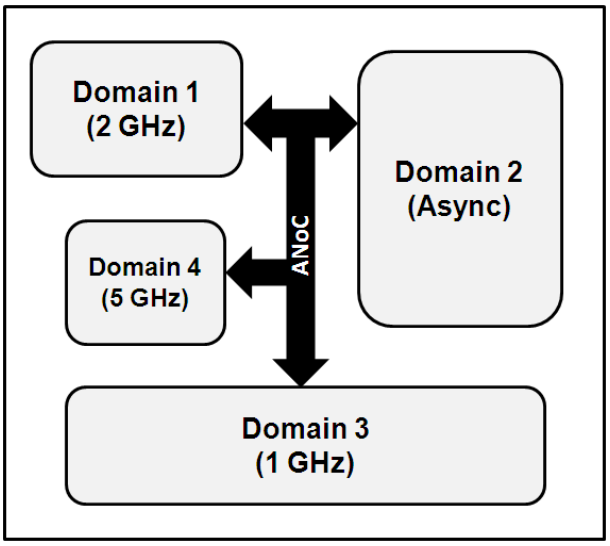


Figure 4.1: GALS Architecture.

Several synchronization schemes or structures for data transfers among independent clock domains in a GALS system have been presented. One category of solutions is to synchronize the signals from asynchronous domain with the local clock in an arbitrary timing relationship and limit synchronization failures within an acceptable level. The most widely applied scheme in this category is the double-latching as illustrated in Figure 4.2. It consists of two serially connected D-Flip-Flop (D-FF) components to latch the input signals with the reference clock of the receiver. It is possible that the first D-FF enters into metastable state if input signal transitions violate the setup or hold timing requirement. In this situation, the second D-FF gives a whole clock cycle for the first D-FF to resolve the metastability before latching its output. However, in the double-latching scheme, there still exists the failure possibility if the first latch can not get rid of metastability state before the second flip-flop samples its output. Therefore, Mean Time Between Failure (MTBF) is introduced to measure the safety of a synchronizer. MTBF gives indication about how often a synchronization failure occurs. The performance analysis of double-latching synchronizers and the equation of calculating MTBF of a synchronizer are presented in [DIK 99] and [KIN 02].

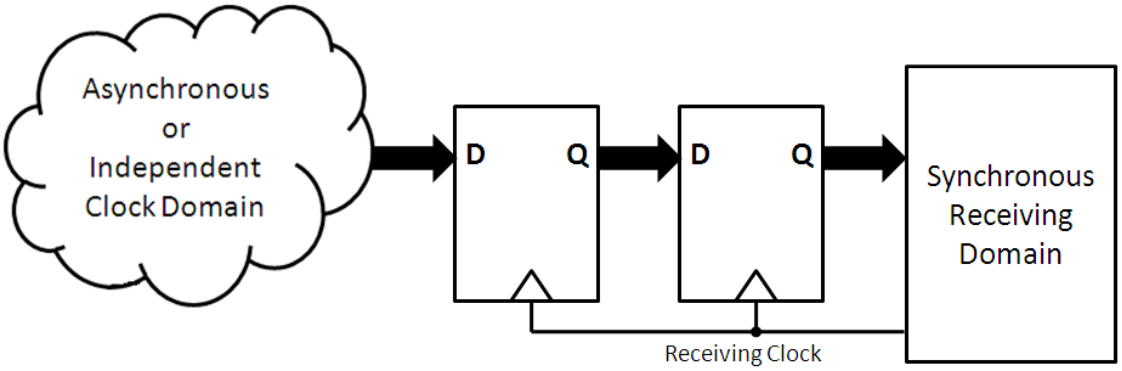


Figure 4.2: Double-Latching Synchronization Scheme.

Another type of solutions of data synchronization in a GALS system is to avoid synchronization failures by adjusting the clock signal of the local synchronous module or by generating a controllable clock signal in the synchronization interface. Clock Domain Crossing (CDC) strategies are required to achieve this purpose. Several hardware techniques have been proposed to deal with this problem. For example, the work

presented in [MAT 05] develops a stoppable clock structure to build a deterministic wrapper. The work in [MUT 99], and [ZHU 02] presents stretchable clock schemes to avoid synchronization failure in the interface between synchronous and asynchronous domains. A plausible clock scheme is firstly presented in [YUN 96a] to manage the data transfers between independent clock domains without synchronization failure. The work presented in [JOY 04], and [MUT 00] further develops the plausible clock scheme. The work in [BOR 97] presents an asynchronous wrapper which combines the stretchable and pausible clock schemes together. This wrapper can avoid synchronization failures caused by metastability in circuits. An interesting taxonomy of these different CDC hardware techniques is summarized and presented in [CUM 09]. Three main strategies are outlined, namely, pausible-clock generators, FIFO buffers, and boundary synchronization.

4.3.1 GALS Wrapper with Pausible Clocking

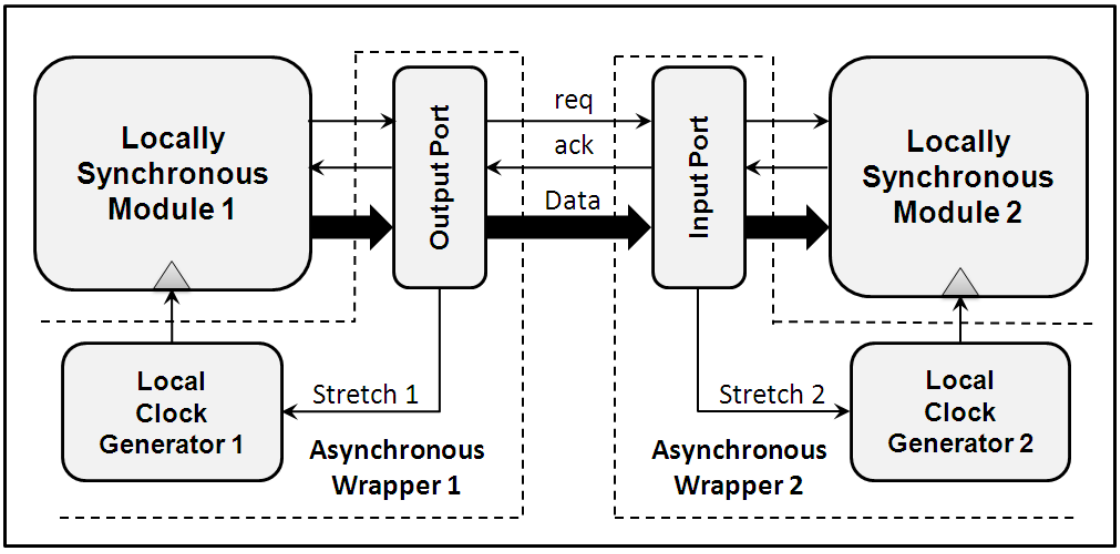


Figure 4.3: GALS System with Pausible Clocking.

Many GALS systems presented in the past few years use (plausible, stretchable, or data-driven) clocking [YUN 96a] and [MUT 00]. The basic idea of all these proposals is similar: transferring data between wrappers when both the data transmitter and data receiver clocks are stopped. This elegantly solves the problem of synchronization between the two clock domains. Figure 4.3 illustrates the general structure of such a system. The asynchronous wrapper contains input and output ports that perform the

handshake process between the locally synchronous modules, and generates a stretch signal to stop the activity of both clocks. The basic GALS method focuses on point-to-point communication between blocks.

4.3.2 FIFO Solutions

Another approach for interfacing locally synchronous blocks is using specially designed asynchronous FIFO buffers and hiding the system synchronization problem within the FIFO buffers [CHE 00], [CHA 03], and [BEI 06]. Such a system can tolerate very large interconnect delays and is also robust with regard to metastability. Designers can use this method to interconnect asynchronous and synchronous systems and also to construct synchronous-synchronous and asynchronous-asynchronous interfaces. Fig. 4.4 diagrams a typical FIFO interface, which achieves an acceptable data throughput [CHE 00]. In addition to the data cells, the FIFO structure includes an empty/full detector and a special deadlock detector.

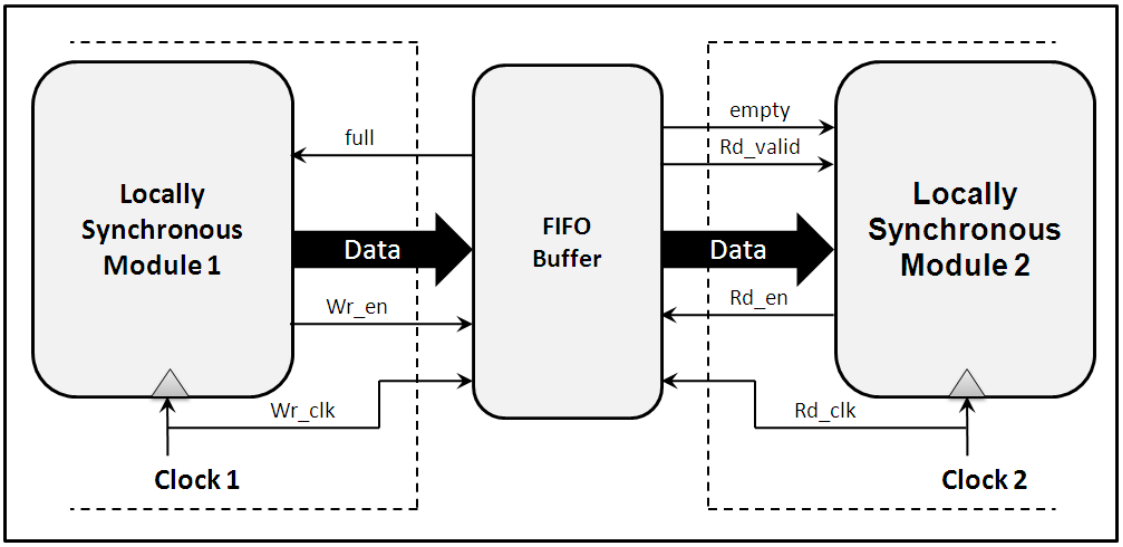


Figure 4.4: Typical FIFO Based GALS System.

The advantage of FIFO synchronizers is that they do not affect the locally synchronous modules operation. However, with very wide interconnect data buses; FIFO structures can be costly in silicon area. Also, they require specialized complex cells to generate the empty/full flags used for flow control. The introduced latency might be

significant and unacceptable for high-speed and low-latency applications. As an alternative, in [BEI 06] a synchronous-asynchronous FIFO based on the bisynchronous classical FIFO design using gray code has been designed, for the specific case of an asynchronous Network-on-Chip (NoC) interface. Their aim was to maintain compatibility with existing design solutions and to use standard CAD tools. Thus, even with some performance degradation or suboptimal architecture, designers can achieve the main goal of designing GALS systems in the standard design environment.

4.3.3 Boundary Synchronization

A third solution is to perform data synchronization at the borders of the locally synchronous island, without affecting the inner operation of locally synchronous blocks and without relying on FIFO buffers. For this purpose, designers can use standard two-flop, one-flop, predictive, or adaptive synchronizers for mesochronous systems, or locally delayed latching [GIN 03, DOB 04]. This method can achieve very reliable data transfer between locally synchronous blocks. On the other hand, such solutions generally increase latency and reduce data throughput, resulting in limited applicability for high-speed systems.

Table 4.1 summarizes the properties of GALS systems synchronization methods. Contrary to earlier expectations, GALS-based solutions do not automatically offer performance gains. Inter-block communication incurs some penalty in all GALS systems. In pausible-clock systems, the clock can be stretched when transferring data on slow communication links, reducing the locally synchronous modules operating frequency. FIFO-based systems, depending on the communication link, suffer from additional latency. If designed carefully, performance degradation in a GALS system will be insignificant; however, in some examples (for various reasons), the reported performance degradation of the GALS system was as high as 23% [MUT 00]. The GALS approach is a vehicle for block interconnects. A crucial parameter for such an application is data throughput and latency. For many GALS solutions, the problem of data throughput is critical. Some pausible-clocking schemes can theoretically reach a maximum data throughput of one data item per clock cycle [MUT 00]. However, more often, data transfers are limited to every second clock cycle or even every fourth or fifth clock cycle

of the locally synchronous block. In addition, in an environment with intensive data transfers the performance degrades significantly. For FIFO-based solutions, the throughput problem is less severe, but latency increases.

Table 4.1: Properties of GALS Systems.

Property	Synchronization Method		
	Pausible Clocking	**FIFO-Based**	**Boundary Synchronization**
Area Overhead	Low	Medium to High	Low
Latency	Low	High	Medium
Throughput	Lowered according to clock pause rate	High	Medium
Power Consumption	Low	High	Medium
Additional Cells	Mutex, Delay-line, Muller-C	Empty/Full flag	Muller-C, Mutex
Advantages	No Metastability	Simple Solution, Throughput	Low Overhead
Disadvantages	Local Clock Generators, Throughput	Area Overhead, Latency	Requires Verification, Throughput

4.4 Application of PSTR for GALS Data Synchronization

In this section we present a new method for synchronizing different clock frequencies in a GALS system. Our GALS system uses the PSTR oscillator presented in Chapter 3 as the main generating source of clocks in each synchronous domain. In contrast to most conventional GALS schemes, the method is not based on including in each ring oscillator a synchronizing element (such as for instance an arbiter) which on one side can pause the clock and on the other side offers a handshake interface. Instead, we propose a scheme in which each synchronous module has both an incoming and an

outgoing clock signal, which have been obtained by opening the ring oscillator module. Since these clock signals also behave as handshake signals, handshake circuits can be used to synchronize the clocks.

4.4.1 Circuit Design

The clock synchronization approaches mentioned in the previous sections have all one property in common: they include in the ring oscillator a synchronizing element that on one side can pause the clock and on the other side offers a handshake interface. Each approach has its specific synchronizing element (such as for instance an arbiter). The fact that either the clock or the environment may proceed implies that input and output operations are mutually exclusive (no overlap in the data validity intervals). A communication between two clock domains then implies two conversions: first from one clock domain to handshakes and then from handshakes to the other clock domain. Therefore, during such a communication one clock domain will be running while the other side clock domain is paused.

In this subsection we propose a scheme in which the handshake signals are obtained directly by opening the PSTR oscillator feedback. The simplest form of clock synchronization is then by means of a C-element which synchronizes the two clock domains directly with a small and predictable timing overhead (and without the need for any conversion). The clock signal generated by each synchronous GALS domain C-element can be used to support bidirectional communication. Arbiters are only needed when they are unavoidable, for example for sharing resources between independent clock domains. The main advantage of this method is that it uses asynchronous logic in securing data communication between different clock domains and avoiding the problem of metastability. Moreover, as process variability could have different effects on the two communicating clock domains even if they are programmed to work at the same clock frequency, this method enables us to remove this slight difference between the clock frequencies and to make them completely synchronized. One more advantage is that it avoids stopping data communication between different clock domains during their synchronization phase.

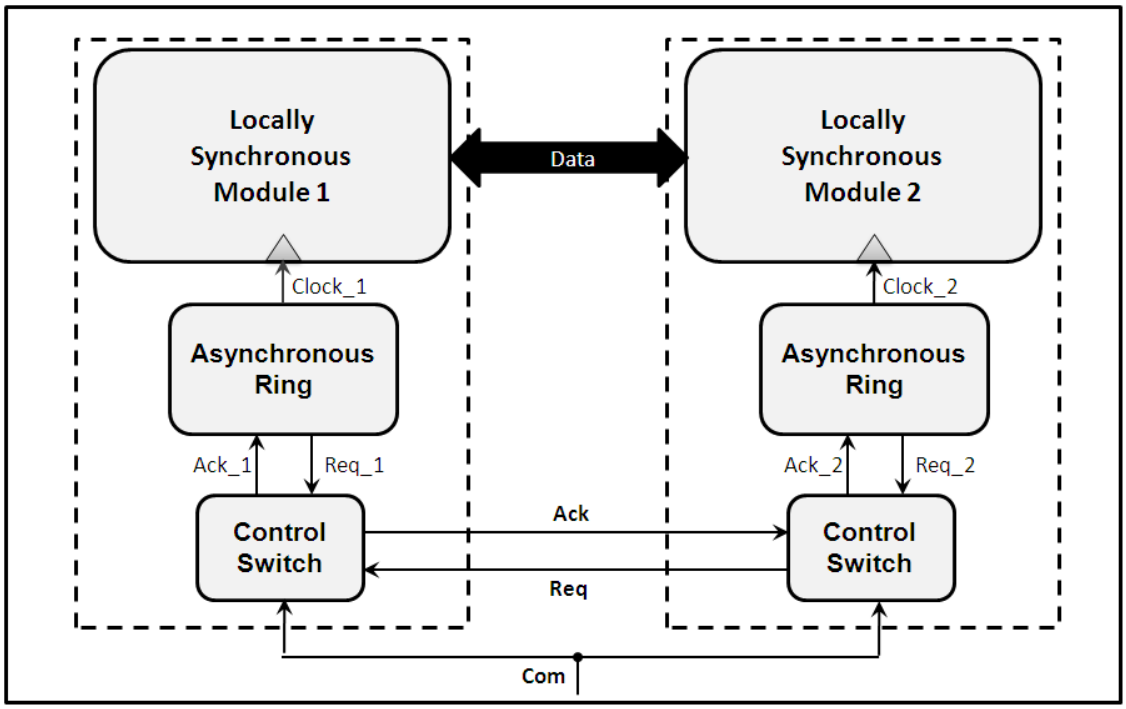


Figure 4.5: PSTR Based GALS System.

Figure 4.5 shows a simplified block diagram of such PSTR based GALS system. It uses a PSTR (i.e. asynchronous ring) oscillator and a kind of control switch to safely synchronize the data transfer by means of the asynchronous handshaking request *Req* and acknowledgement *Ack* signals. The *Com* signal is used to define whether or not the synchronous clock domains are in a data exchange mode. The detailed interconnections between the asynchronous ring and the control switch are depicted in Figure 4.6.

The control switch is used to manage the connections of *Ring_out_Req* and *Ring_out_Ack* signals with the GALS network through the handshaking *Req_ANoC* and *Ack_ANoC* signals. If *Com* equal 1 (i.e. we have data exchange) the *Ring_out_Req* and *Ring_out_Ack* will be connected directly to *Req_ANoC* and *Ack_ANoC* respectively. In this case the two communicating domain asynchronous ring outputs will be connected together through the control switch. Then, the main functionality of the control switch will be to synchronize the two communicating clock domains together, so that they are working with the clock frequency of the slower communicating one. This synchronization has to be done without the need to reprogram each PSTR oscillator, just

by the mean of a C-element (as shown in Figure 4.7). Once *Com* signal equals 0 (i.e. no communication) the feedback connections of the asynchronous ring are closed. Each ring oscillator returns to work again with its own operating clock frequency.

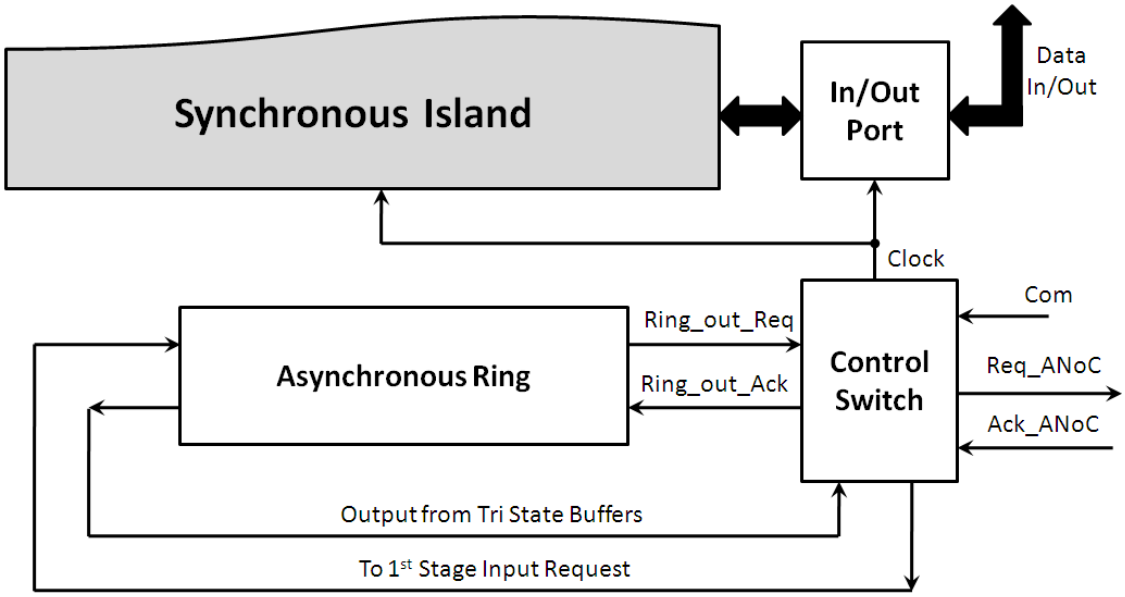


Figure 4.6: Control Switch Interconnections with the Asynchronous Ring.

The details of the control switch circuit design are depicted in Figure 4.7. The Muller C-element *C2* has one input which is connected to the output of the tri-state buffers of the PSTR shown in Figure 3.13. The second input of *C2* comes from the *Mux* output. The *Mux* defines wither the second input of *C2* will be connected to *Ack_ANoC* or it will be short circuited with the first input of *C2* according to *Sel* value. If "*Sel* = 0", then the two inputs of *C2* will be connected together. In this case *C2* will behave as a buffer gate that passes the signal coming from the tri-state buffers directly to *Ring_out_Ack*, which means that the ring backward feedback connection is now closed. *C1* and *Delay* element have been added in the forward feedback connection in order to match the delays of *C2* and the *Mux*. The output of *C1* is connected to the request input of the first stage in the PSTR. Note that *C1* and *C2* set and reset inputs are the same as that of last and first stages in the PSTR respectively. Currently, the two PSTR asynchronous ring feedback connections are completely closed and balanced, so that the ring behaves normally as explained in Chapter 3. On the other hand, if "*Sel* = 1" then the PSTR

backward feedback connection will be opened. The two inputs of *C2* will be *Ack_ANoC* and the output coming from the tri-state buffers. Given that, the clock signal of the other side communicating module is connected to the GALS network through *Ack_ANoC*. Accordingly, each of the *C2* inputs will now be connected with the clock output of the two different communicating domains.

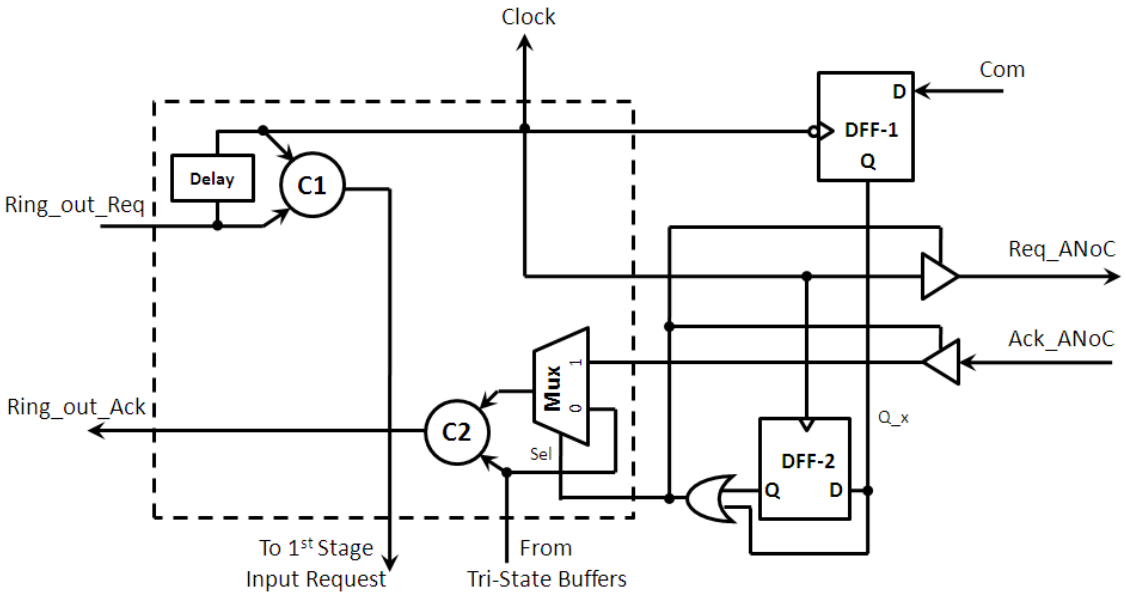


Figure 4.7: Control Switch.

In order to guaranty the correctness of our system behavior, we have to be always sure that the last clock cycle of our communicating domain has already been completed before/after the start/stop of the synchronization phase of the two clock domains; this is to avoid the presence of any glitches or truncated clock periods. So that, if *Com* signal changes its state from 0 to 1 (i.e. from no communication into data exchange state), the multiplexer *Sel* signal will not be allowed to change its state except with the next negative edge of our communicating domain. In this case, it will be always guaranteed that the other GALS communicating domain clock output will only be connected to one of *C2* inputs only when the clock of our communicating domain is active low. As Muller C-element is actually a state holding element, *C2* will not start to generate any output except when both inputs are equal to 1. Thus, the synchronization phase will only start just after the completion of our communicating domain last clock cycle. After that, *C2* output will

exactly follow the slower clock frequency of the two communicating domains with a small time shift corresponding to the Charlie effect on *C2*. Conversely, if *Com* signal changes its state from 1 to 0 (i.e. from data exchange into no communication state), DFF-1 output *Q_x* will be changed from 1 to 0 on the next negative edge of the synchronized clock. As *Q_x* is connected to the input of DFF-2, therefore with the next positive clock edge of the last synchronized clock cycle, the output of DFF-2 will be changed from 1 to 0. Thus, the two inputs of the OR gate are 0, which forces *Sel* signal to be 0. This disconnects *Ack_ANoC* from the *Mux* input, which cuts the connection between the two communicating domains. Moreover, it insures that *Ring_out_Ack* is reconnected with the output from the tri-state buffers. As a result, the backward feedback connection is again closed after completing the last clock cycle of the synchronized clock. An amazing feature of the PSTR is that, it will now consider the last positive clock edge as the start of the first original domain clock cycle and returns back to run at its normal oscillating frequency, thanks to the state holding functionality of *C1*.

4.4.2 Simulation Results

The control switch shown in Figure 4.7 is implemented on a CMOS 45nm technology from STMicroelectronics. The delay information is extracted from the physical design and inserted in our timed VHDL model. Figure 4.8 shows an example where the control switch is requested to change the synchronous module state from no communication to data exchange with another synchronous module, then it disconnects and returns back to the no communication state.

At point “A”, *Com* changes its level from 0 to 1. As a result, *Q_x* and *Sel* are changed with the next negative edge of *Clock*. Once, the *Clock* completes its last clock cycle, it starts to decide wither or not to continue at the same clock frequency. As the other communicating part clock frequency connected to *Ack_ANoC* is slower than the *Clock* frequency, the *Clock* frequency is synchronized with *Ack_ANoC* with a small time shift corresponding to *C2* delay. Consequently, the two domains are now well synchronized, with the receiving domain is sampling the correct data sent over the GALS network.

At point “B”, *Com* changes its level from 1 to 0. Again, *Q_x* is changed with the next negative edge of *Clock*, while *Sel* changes its level after the completion of the last *Clock* cycle. After that, the *Clock* returns again to oscillate at the same clock frequency it has before the data exchange state. As only a small number of additional elements have been used in this control switch circuit (C-element, Mux, and DFF). Therefore, this circuit gathers the small area (30.34 µm²) and the low power consumption advantages (12.67 µW) of the pausible clocking GALS with the high throughput advantage of the FIFO-based GALS, as we are switching directly from one frequency to the other with no need for pausing or reprogramming our clock generator.

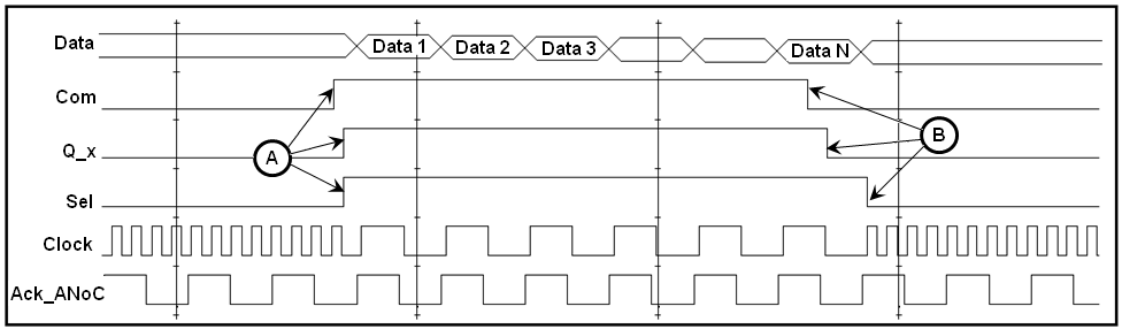


Figure 4.8: Timing Diagram of the Control Switch.

4.4.3 Multipoint Interconnection Schemes

A shared system bus or another form of multi-point data exchange is a key necessity of a modern SoC design as point-to-point exchange alone does not provide the necessary modularity and scalability. Synchronous SoC designs most often use a synchronous shared bus with central arbitration. Clock skew across the chip and the use of many timing domains in a system raise many problems concerning placement and routing and strongly limit the throughput. Our early circuits only supported point-to-point connections. To generalize the idea with the standard synchronous on-chip interconnects solutions, several interconnection approaches are being considered to add to the current GALS technique. Three different topologies were chosen for further studies.

Shared Bus

The point-to point concept shown in the previous sections can be enhanced to support a shared bus solution as shown in Figure 4.9. A central arbiter handles the access to the shared bus resources by multiple senders, and an address decoder selects the appropriate destination. The control switch located in each GALS module manages the self-timed data transfers between different GALS modules through the handshaking request and acknowledgment signals. The shared bus only needs little area, but has the disadvantage of high capacitive loads, resistances and crosstalk effects due to the rather long wiring. As every bus transaction is controlled by a four-phase handshake, long wires cause a considerable delay on the handshake cycle (Request rising → Acknowledgment rising → Request falling → Acknowledgment falling).

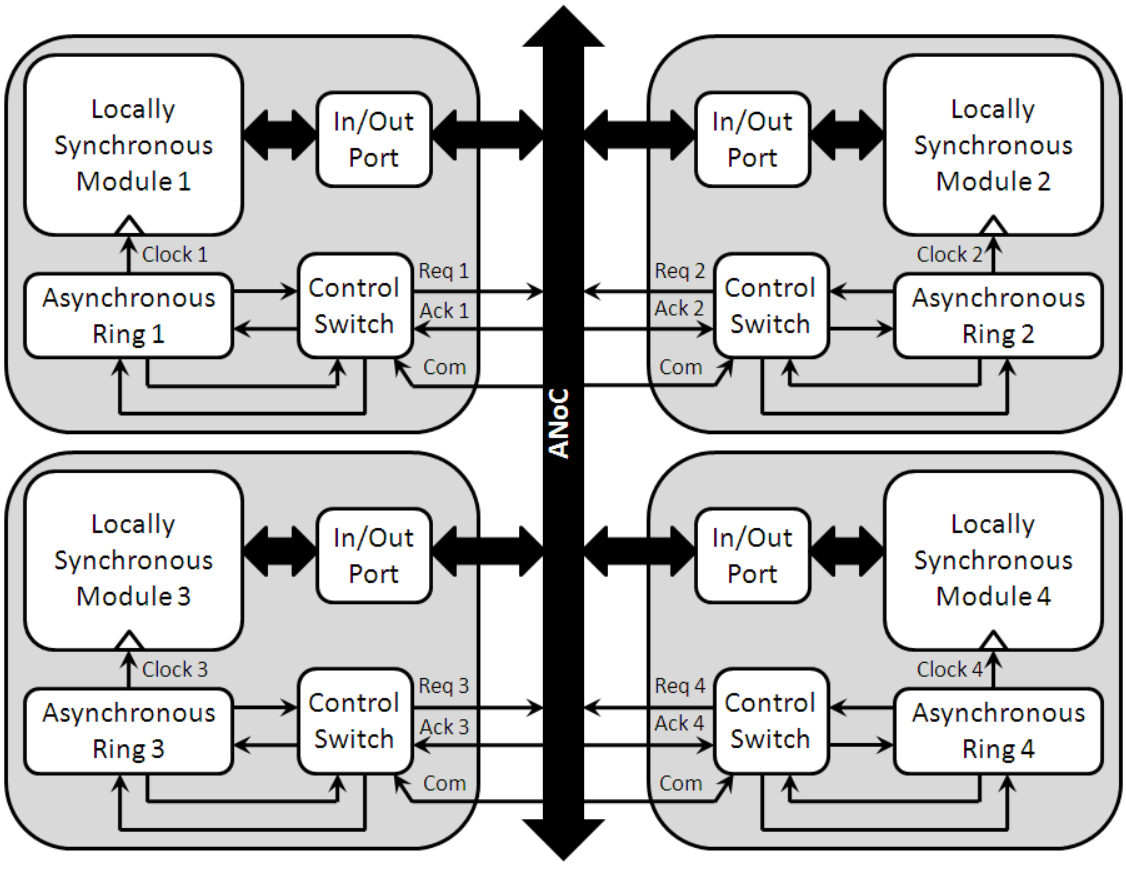


Figure 4.9: Multipoint GALS interconnects (Shared Bus).

Switch Network

In Figure 4.10 a central switch network routes the incoming requests from the senders to the appropriate receiver. Such a switch can be build from a matrix of smaller self-timed switching elements. Arbitration between the incoming requests is handled within the switching elements. The micropipeline-like structure divides the long interconnection wires into smaller parts thereby lowering capacitance, resistance as well as crosstalk effects. The time for a handshake cycle is greatly reduced as the acknowledge signal comes from the subsequent stage directly. As long as data transfers do not interfere, the capacity of the network as a whole can be a multiple of the shared bus solution due to the concurrent channels. No arbitration is necessary in this case. The price to pay is the higher end-to-end latency, quite poor scalability, and a considerable area requirement.

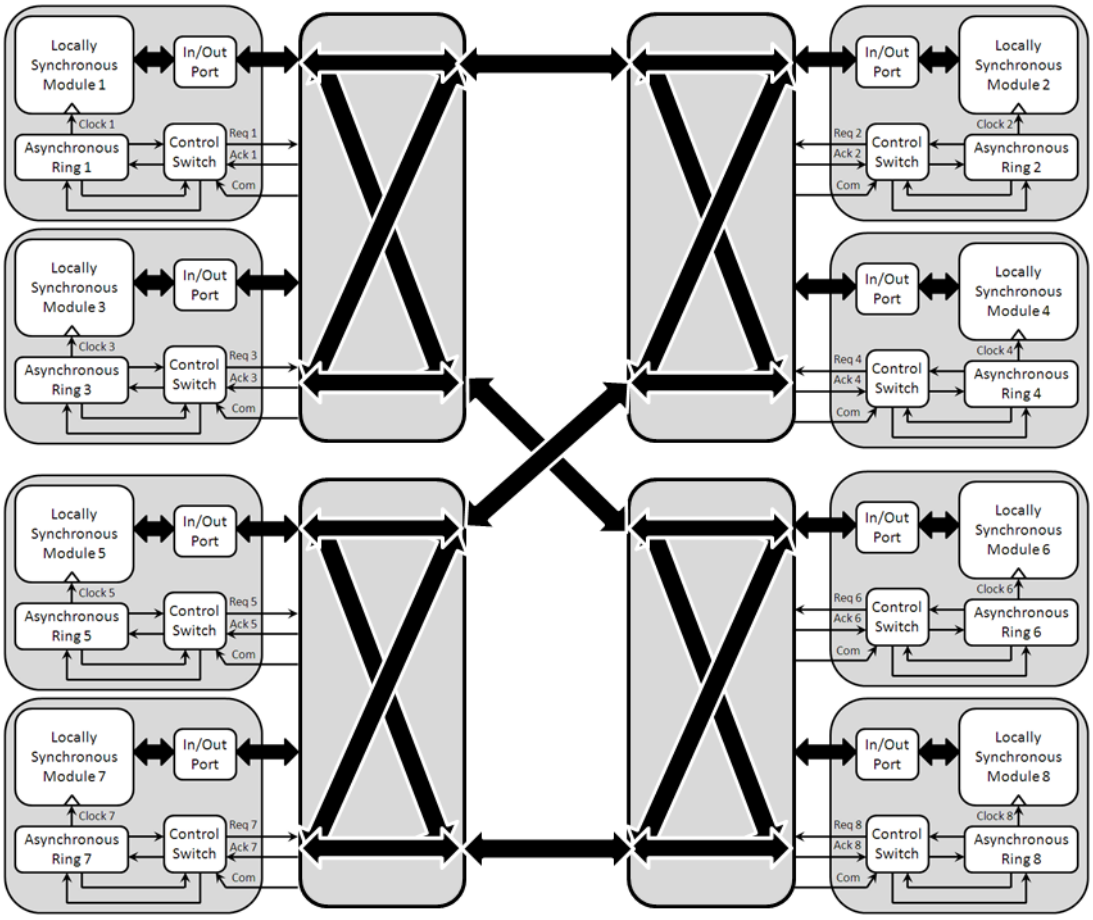


Figure 4.10: Multipoint GALS interconnects (Switch Network).

Ring Structure

Figure 4.10 shows a block diagram of a self-timed ring topology that connects GALS modules in a circular fashion. At each node, local address decoders decide whether a received data word is bound for the local node or must be passed on to the successor node. Every node can also insert new data into the ring. One can think of a feed through solution with ring nodes connected by point-to-point transfer channels only, or a bypass version with ring transceivers acting independently from the modules. The two approaches are addressed in details in [VIL 03].

Concerning throughput, latency, and area requirement, the ring occupies a position somewhere in between the shared bus and the switch solution. Because the data packets have to traverse unrelated transceivers before reaching their destination, the latency is high compared to the shared bus. On the other hand the reduction of the length of the connecting wires scales down wire capacitance and crosstalk effects as in the switch and thus higher throughput is expected. The approach is modular and scales easily. No central instance is necessary and no distinction is made between masters (that can initiate a data transfer) and slaves (that just respond to a request).

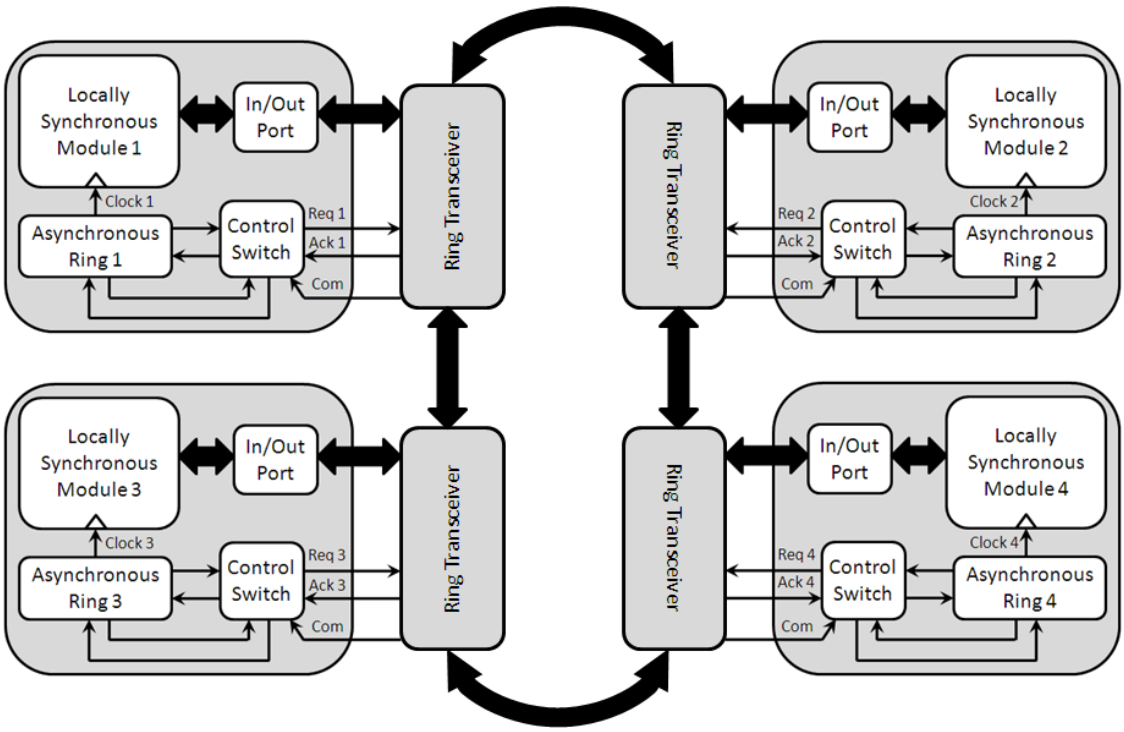


Figure 4.11: Multipoint GALS interconnects (Ring Structure).

4.5 Conclusions

The use of GALS architectures opens a range of opportunities and advantages for system design. From a synchronous designer's point of view, it provides the opportunity to completely decouple timing constraints of different modules and to assemble them without fearing metastability. This adds a high degree of modularity and the modules get much more amenable to reuse.

The GALS approach is also advantageous from an asynchronous designer's perspective. Because asynchronous modules can easily be incorporated in the self-timed network-on-chip, they can be used in system-on-chip designs wherever they are available and beneficial. Thus asynchronous design styles can demonstrate their potential and maturity in projects they otherwise wouldn't have access to.

This Chapter presents a comparison between different GALS synchronization schemes. After that, a new scheme based on the use of the PSTR as the main clock generating source in each GALS module is proposed. This scheme supports reliable communication between independently clocked GALS modules. However, instead of including in each PSTR oscillator a standard element for pausing the clock (such as for instance an arbiter), each synchronous module is proposed to have both an incoming and an outgoing clock signal, which have been obtained by opening the PSTR oscillator module. Since these clock signals behave as handshake signals, handshake circuits can be used to synchronize the clocks. The simplest way synchronizing two different clock domains is then by means of a C-element, which introduces only a small and predictable timing overhead. A multiplexer and DFF have been used in this circuit design, in order to adapt the instants on which we switch between no communication and data exchanges states, so we don't have any glitches nor truncated clock periods.

The advantages of this scheme are: due to its self-timed nature, the control switch used with the PSTR allows connecting modules running at different clock frequencies with very small power consumption. It offers high throughput compared to a pausible clock solution and a small area compared to FIFO based solution. Moreover, it can be easily applied in different multipoint GALS interconnection schemes.

Part – II

CMOS Power Reduction And Design for Yield

Chapter 5

Power Saving Techniques

5.1 Introduction

Rapid development of portable systems like laptops, PDAs, digital wrist watches, and cell phones requires low power consumption and high density ICs, which leads to a surge of innovative developments in low power devices and design techniques. In most cases, the requirements for low power consumption must be met with equally demanding goals for high chip density and high throughput circuits. Hence, the low-power digital design and digital ICs have emerged as very active fields of research and development. In this cutting-edge technology era, reduction in the power dissipation is a critical task, especially as the size of transistors is scaled down to increase the transistor density over the silicon chip. Consequently, careful consideration must be given to minimize digital ICs power dissipation without sacrificing its performance.

5.2 Sources of CMOS Power Consumption

There are three major sources of power dissipation in digital CMOS circuits which are summarized in the following Equation 5.1.

$$P = \underbrace{P_{leakage}}_{Static\ Power} + \underbrace{P_{switching} + P_{shortcircuit}}_{Dynamic\ Power} \tag{5.1}$$

The first term $P_{leakage}$ represents leakage power consumption, where leakage currents can arise from substrate injection and sub-threshold effects, which is primarily determined by fabrication technology considerations. The second term $P_{switching}$ is the switching power which occurs when a gate is switching from one logic state to another and is the result of the switching current (needed to charge and discharge internal nodes). Finally, $P_{shortcircuit}$ is the short circuit power consumption due to the direct-path short

circuit current, which arises when both the NMOS and PMOS transistors are simultaneously active, conducting current directly from supply voltage to ground.

The sum of the switching and the short circuit powers is called the dynamic power, while the leakage power is called the static power. Accordingly, power dissipation in CMOS circuits involves both static and dynamic power dissipations. In the submicron technologies, the static power dissipation, caused by leakage currents and sub-threshold currents contribute with a small percentage to the total power consumption. This static power consumption occurs when all inputs are held at the same logic level and the circuit is not in changing states. On the other hand, the dynamic power dissipation, resulting from charging and discharging of parasitic capacitive loads of interconnects and devices dominates the overall power consumption. Therefore, traditional low power management techniques for 130nm and 180nm technology nodes were focusing only on reducing the dynamic power.

Advances in CMOS technology allow MOS transistor sizes to be continuously scaled down in progressively smaller technology nodes. As transistor sizes become smaller, supply voltages can be lowered to reduce the power dissipation. In order to achieve high speed with low supply voltages, the threshold voltage must be reduced accordingly. This reduction of the threshold voltages combined to the short channel effect has led to an exponential increase in the transistors leakage current. Thinner gate oxides have also contribute to an increase in gate leakage current as well. Consequently, leakage power is no longer negligible in deep-submicron CMOS technologies.

At 90 nm and beyond, leakage power accounts for a significant portion of the total power in high-performance designs as shown in Figure 5.1 [PAN 05], therefore its management becomes essential in the ASIC design process. Leakage power is now a growing concern in the overall design process. Unlike dynamic power, which can be managed by reducing switching activity, leakage power causes its effect as long as the power is on. That is why current process technologies are pushing designers to consider new design methods to reduce both static and dynamic power consumption.

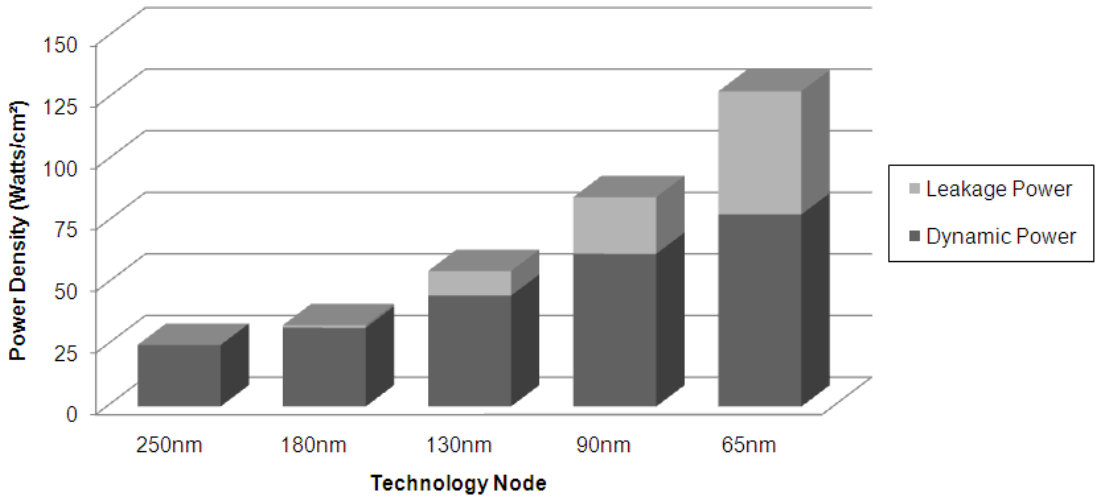


Figure 5.1: Static vs. Dynamic Power with Process Migration.

In this chapter a survey on different methods that can be used to control both the static and the dynamic CMOS power consumption will be presented. The following subsections will show using equations how each part contributes to the total CMOS circuit power consumption. This will help the circuit designers in choosing the most suitable technique for reducing the effect of each power consuming part, as will be shown in Sections 5.3 and 5.4.

5.2.1 Leakage Power

Leakage or static power dissipation is associated with logic gates when they are inactive (static); that is, not currently switching from one state to another. In this case, these gates should theoretically not be consuming any power at all. In reality, however, there is always some amount of leakage current passing through the transistors, which means they do consume a certain amount of power. Even though the static power consumption associated with an individual logic gate is extremely small, the total effect becomes significant when we come to consider today's ICs, which can contain tens of millions of gates. Furthermore, as transistors shrink in size when the industry moves from one technology to another, the level of doping has to be increased, thereby causing leakage currents to become relatively larger. The result is that, even if a large portion of

the device is completely inactive, it may still be consuming a significant amount of power. The leakage power consumption associated with the transistors is expressed as:

$$P_{leakage} = V_{dd}.I_{leakage} = V_{dd}.I_0 e^{-qV_{th}/KT} \quad (5.2)$$

where q is the elementary charge, V_{th} is the transistor's threshold voltage, K is Boltzmann's constant, I_0 is the saturation current, and T is the temperature.

One important point about this equation is that it shows that static power dissipation has an exponential dependence on temperature. This means that as the chip heats up, its static power dissipation increases exponentially. Furthermore, we see that static power dissipation has an inverse exponential dependence on the switching threshold of the transistors.

5.2.2 Switching Power

Equation 5.3 allows for the calculation of the power consumption of a switched capacitor. At the transistor-level, the load capacitance C_L includes the parasitic gate overlap and fringing capacitances as well as the Miller capacity. α models the switching probability of the transistor during a cycle of the clock toggling at frequency f. V_{dd} is the supply voltage.

$$P_{switching} = \alpha.C_L.f.V_{dd}^2 \quad (5.3)$$

From equation 5.3 we can note that switching power is not a function of the transistor size, but rather a function of switching activity and load capacitance. Thus, it is data dependent.

5.2.3 Short-Circuit Power

Short-circuit power is the second part of the dynamic power consumption. It occurs during a short period of time when both the pull-up and the pull-down networks of static CMOS-gates are conducting. Equation 5.4 gives a simple model of the short-circuit

power with β modeling the transistors' conductivity per voltage unit factoring the linear region, V_{th} is the threshold voltage, T is the inputs' rise/fall time, and τ is the gate delay.

$$P_{shortcircuit} = \frac{\beta}{12}(V_{dd} - 2V_{th})^3 \frac{\tau}{T} \quad (5.4)$$

Equation 5.4 is an overestimation by up to a factor of three. For an accurate analysis, transistor level models and transient analyses are needed [HED 87]; however, $P_{shortcircuit}$ within modules can be captured as part of the dynamic power models of the modules.

5.3 Static Power Reduction Techniques

Since, static power has become a significant contributor to the total CMOS power consumption in today's technological nodes, as the gate length and threshold voltage are scaled down. Therefore, several techniques have to be applied at the circuit level to reduce this amount of power. From equation 5.2, one can conclude that, the static power reduction is mainly related to managing the transistor threshold and/or supply voltages (V_{th} and/or V_{dd}). As a result, static power reduction techniques can include multi-threshold libraries, multiple and dynamic supply voltages, power gating and variable body biasing [BOR 05] and [PFI 07].

5.3.1 Multi-Threshold

Silicon foundries have started to offer multiple threshold devices at the same process node to address the need to control leakage current and enabling designers to trade off leakage and performance [SVI 01]. Consequently, a certain number of libraries of gates are defined for different implementations of functions, including high-voltage-threshold (HVT), standard-voltage-threshold (SVT) and low-voltage-threshold (LVT) cells, which have different speed and leakage characteristics.

It is common for the LVT device to have an order of magnitude higher leakage than the SVT device and the HVT device to have leakage characteristics an order of magnitude below the standard. This reduction in leakage is not free, however, and it

comes at the expense of the speed of the device and an increase in the cost of the fabrication process. The LVT cells are the fastest and have the highest leakage. They are used by the synthesis and optimization tools in the critical paths. The SVT and HVT gates are used in less-critical paths to reduce leakage power. The challenge for EDA tools is to use the available characteristics of the cells in the design library to create an implementation that will meet the timing constraints, while reducing the leakage current as much as possible.

5.3.2 Body Biasing

The threshold voltage and therefore the leakage current in a CMOS transistor are controllable by varying the back biasing. The change in the threshold voltage is roughly proportional to the square root of the back bias voltage. A distinct advantage of this approach is that during periods when heavy processing is needed, the threshold voltage can be reduced, thus speeding up the cells. When the cells are in a slower drowsy or idle mode, the threshold voltage can be raised, thus lowering the leakage.

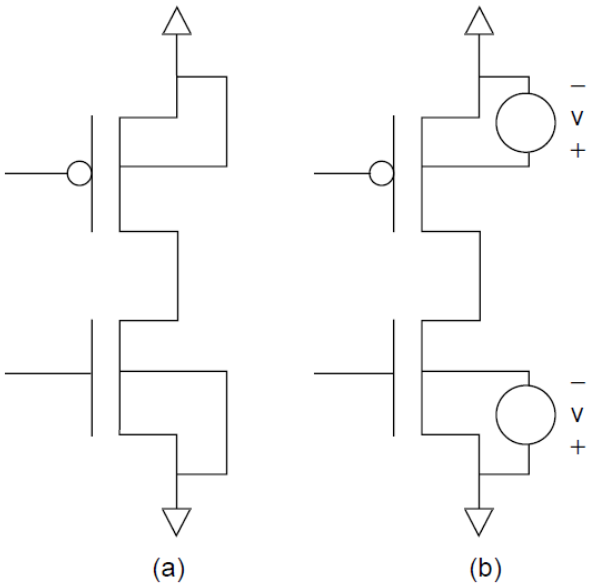


Figure 5.2: Using Bias to Control Threshold Voltages.

One significant impact of using variable back biasing is that two new terminals for each cell need to be routed. A common ASIC design practice is to create cells that tie

the N-well regions to V_{dd} and the P-well regions to ground. In the physical implementation, these are simply predefined contacts designed into the cell, which are connected as part of the power and ground routes. To enable back-biasing, new voltage lines are routed to control the bias. These can be to individual cells or, more likely, to regions that contain multiple cells sharing the same WELL and a common tie-cell to control the WELL bias, [FLA 03] and [MAR 02]. Figure 5.2 illustrates the concept of variable body biasing. Figure 5.2 (a) is a traditional implementation with wells tied to V_{dd} and V_{ss}, and Figure 5.2 (b) presents wells biased for leakage reduction.

5.3.3 Power Gating

To reduce the overall leakage power of the chip, it is highly desirable to add mechanisms to turn off blocks that are not being used. This technique is known as power gating. The basic strategy of power gating is to provide two power modes: a low power mode (i.e. sleep mode) and an active mode. The goal is to switch between these two modes at the appropriate time and in the appropriate manner to maximize power savings while minimizing the impact to performance.

Power-gating reduces leakage by reducing transistor gate to source voltage. The operation of power-gating technique is simple. A header (p-type transistor) switch is placed in between a block and power to control supply power from this block with sleep signal, see Figure 5.3. In active mode, the voltage V_V_{dd} is acting as power supply at a potential of approximately V_{dd} to the block; leakage power exists both in header and this circuit block. In standby mode, header is switched off, meaning that the voltage V_V_{dd} is beginning to drop with time. The voltage is no longer at V_{dd}, but rather at a voltage somewhat above V_{SS} at saturation point. Hence, transistor gate to source voltage is reduced. As soon as V_V_{dd} starts to fall, leakage power saving in this block begins. Yet, leakage still exists in the header. This is why sleep transistors are usually made of HVT devices to prevent from cell leakage while maintaining a high potential on V_V_{dd}. Same principle applies to a footer (n-type transistor) switch where it is inserted between a block and ground. Both headers and footers are employed alternatively in a power-gating design.

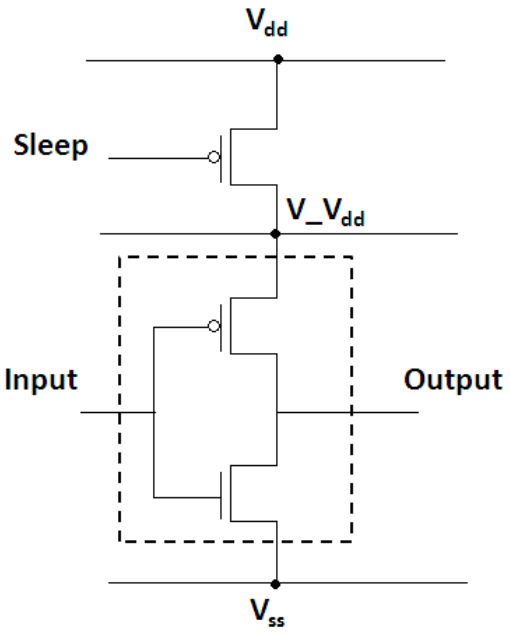


Figure 5.3: Fine Grain Power Gating with an Inverter.

Since standard cells are tagged to an additional transistor, chip area will be much bigger than a normal one. This will result a larger die size on wafer and leads to cost increasing. It is also required a set of libraries for a good power saving, timing closure and area efficiency design. This type of power-gating is known as *fine-grain power-gating*, [FRE 07]. In *coarse grain power gating*, a block of gates has its power switched by a collection of switch cells, see Figure 5.4. The sizing of a coarse grain switch network is more difficult than a fine grain switch as the exact switching activity of the logic it supplies is not known and can only be estimated. But coarse grain gating designs have significantly less area penalty than fine grain. Over the last few years, there has been a strong convergence towards coarse grain power gating as the preferred method. The area penalty for fine grain power gating has just not proven worth the savings in design effort. Today, virtually all power gated designs use coarse grain power gating.

One of the key challenges in any power gating design is managing the in-rush current when the power is reconnected. This in-rush current must be carefully controlled in order to avoid excessive voltage drop in the power network; otherwise, the function and state of powered-on blocks could be corrupted as the power gated block goes through its sleep/wakeup sequence.

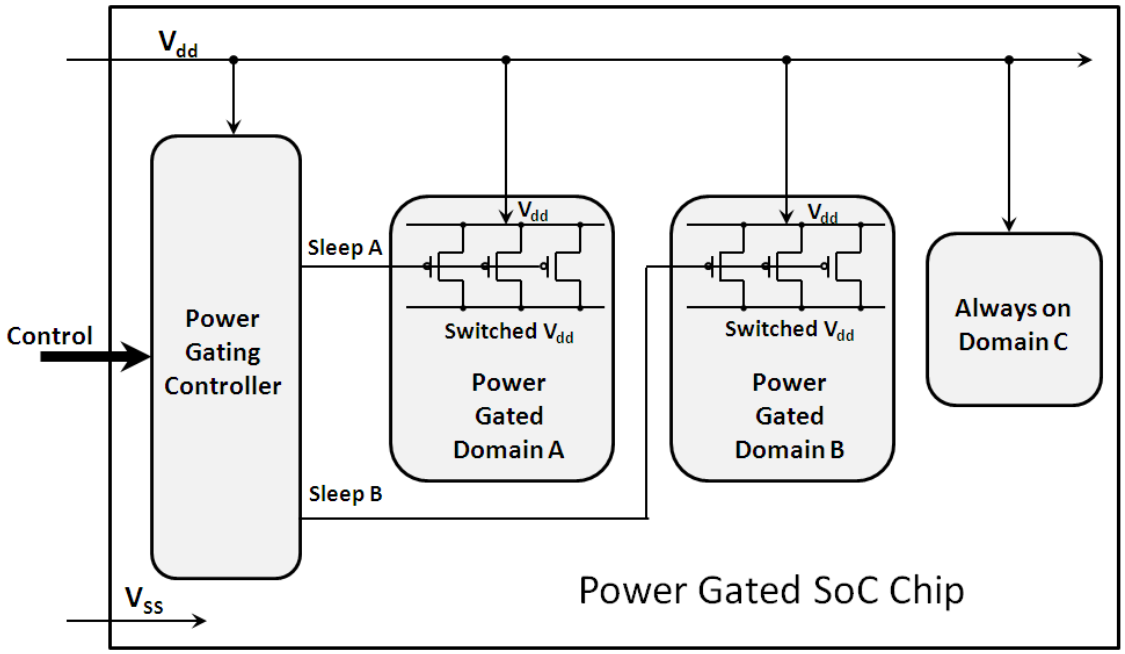


Figure 5.4: Coarse-Grain Power Gating.

5.4 Dynamic Power Reduction Techniques

The dynamic power consumption is a dominant power component for the current and future design technologies. Dynamic power substantially increases in nanometer technologies because of increased number of on-chip functions as well as a prolonging trend on getting higher clock frequencies. Since all applications running over system processors do not require fast processors to operate at the highest speed all the time, it is possible to slow down the fast circuitry and reduce the static power consumption as well as the dynamic power consumption when the maximal performance is not required.

Equation 5.3 calculates the dynamic power consumption, $P_{dynamic}$, of CMOS logic gates. It can be easily inferred that $P_{dynamic}$ is proportional to the load capacitance, C_L ,the square of V_{dd} , the switching activity, α, and the clock frequency, f. Consequently, power consumption reduction can be achieved by:

- Reducing of output capacitance, C_L
- Reducing of supply voltage, V_{dd}
- Reducing of switching activity, α
- Reducing of clock frequency, f

Thus, a designer should devise new techniques aiming at the decrease of each above mentioned parameter or any combination of them. Concerning, the load capacitance C_L, it can only be affected during chip design (for example by minimizing on chip routing capacitances and reducing external components access). Therefore the two main dynamic power reduction strategies concern the reduction of the switching activity and the supply voltage with the clock frequency. Clock gating is a very popular example of switching activity control. Dynamic Voltage and Frequency Scaling "DVFS" introduces a significant example of dynamic power reduction in most complex SoC systems by controlling their supply voltage and clock frequency. Therefore, the rest of the thesis will mainly focus on the use of DVFS in GALS systems, and how they can be improved to smartly control the process variability effect.

5.4.1 Clock Gating

A significant fraction of the dynamic power in a chip is in the distribution network of the clock. Up to 50% or even more of the dynamic power can be spent in the clock buffers. This result makes intuitive sense since these buffers have the highest toggle rate in the system, there are lots of them, and they often have high drive strength to minimize clock delay. In addition, the flops receiving the clock dissipate some dynamic power even if the inputs and outputs remain the same.

The most common way to reduce this power is to turn clocks off when they are not required. This approach is known as clock gating. Modern design tools support automatic clock gating: they can identify circuits where clock gating can be inserted without changing the functionality of the logic. Figure 5.5 shows how clock gating works. In the original design, the register is updated or not depending on a variable "EN". The same result can be achieved by gating the clock based on the same variable. If the registers involved are single bits, then a small savings occurs. If they are, say, 32-bit registers, then one clock gating cell can gate the clock to all 32 registers (and any buffers in their clock trees). This can result in considerable power savings.

In the early days of RTL design, engineers would code clock gating circuits explicitly in the RTL. This approach is error prone – it is very easy to create a clock gated

circuit that glitches during gating, producing functional errors. Today, most libraries include specific clock gating cells that are recognized by the synthesis tool. The combination of explicit clock gating cells and automatic insertion makes clock gating a simple and reliable way of reducing power. No change in the RTL is required to implement this style of clock gating.

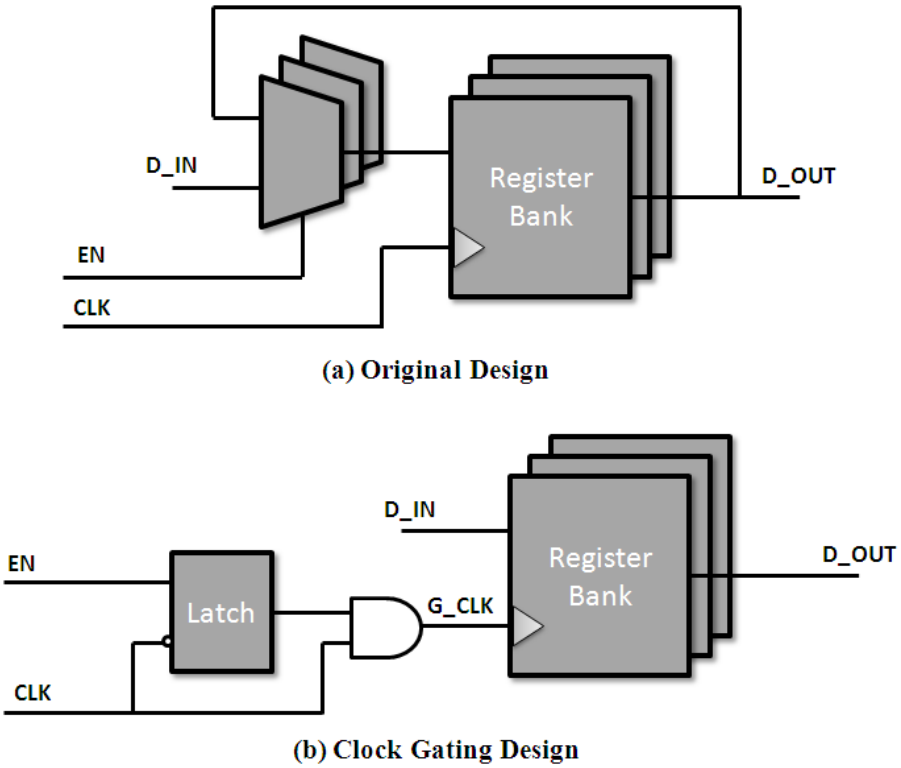


Figure 5.5: Clock Gating.

In [POK 07], Pokhrel reports on a unique opportunity his team recently had to compare a (nearly) identical chip implemented both with and without clock gating. As a power reduction project, an existing 180nm chip without clock gating was re-implemented in the same technology with clock gating. Only minor changes in the logic were implemented (some small blocks were removed and replaced by other blocks, for a small net increase in functionality). Pokhrel reports an area reduction of 20% and a power savings of 34% to 43%, depending on the operating mode. (This savings was realized on the clock gated part of the chip; the processor was a hard macro and not clock gated. Power measurements were made on the whole chip when the processor was in sleep

mode; i.e. the processor clock was turned off.) The power measurements are from actual silicon. The area savings is due to the fact that a single clock gating cell takes the place of multiple multiplexers. Pokhrel makes a couple of interesting observations:

- After some analysis and experiments, the team decided to use clock gating only on registers with a bit-width of at least three. They found that clock gating on one-bit registers was not power or area efficient.
- Much of the power savings was due to the fact that the clock gating cells were placed early in the clock path. Approximately 60% of the clock buffers came after the clock gating cell, and so had their activity reduce to zero during gating.

5.4.2 Dynamic Voltage and Frequency Scaling

Dynamic voltage and frequency scaling (DVFS) is recognized as one of the most effective power reduction techniques. Given that, switching power $P_{switching}$ is the major contributing part to total CMOS power dissipation. Therefore, controlling $P_{switching}$ will have a significant impact on the reduction of the total CMOS power consumption. Equation 5.3 shows that, reducing the supply voltage V_{dd} will decreases the switching power dissipation. The reduction in switching power is a quadratic function of the voltage V_{dd} and a linear function of the clock frequency f_{clk}. As a result, Dynamic Voltage Scaling (DVS) can be used to efficiently manage the SoC power/energy consumption [FLA 04]. Supply voltage can be reduced whenever slack is available in the critical path. However, this drop will also decrease the computational speed because of the propagation delay of gates, i.e. T_d, which is seriously increasing as V_{dd} approaches the threshold voltage of the device V_{th}, see Figure 5.6. In this figure an inverter implemented on STMicroelectronics 45nm CMOS technology using three different libraries (i.e. HVT, SVT and LVT) was taken as an example. Figure 5.6 also shows that for the same gate, the LVT implementation provides the smallest propagation delay while the HVT implementation presents the largest amount of delay, as it was explained previously in section 5.3.1. The relation between the propagation delay T_d and the supply voltage V_{dd} can be expressed as:

$$T_d \, \alpha \, \frac{V_{dd}}{(V_{dd} - V_{th})^2} \tag{5.5}$$

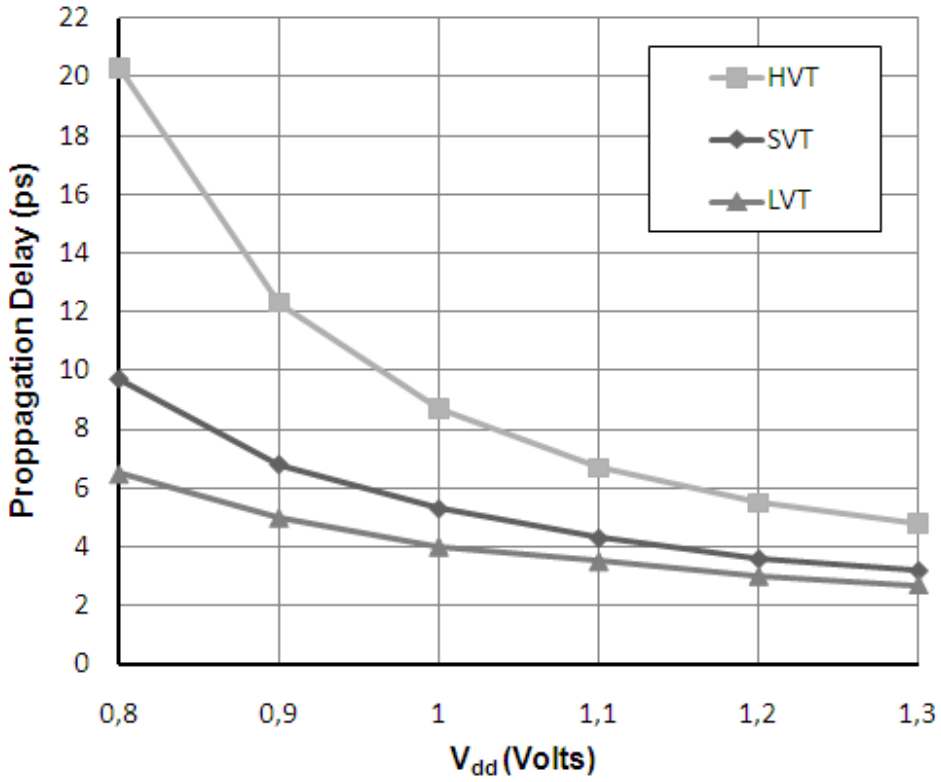


Figure 5.6: Propagation delay vs. V_{dd} for an inverter gate in STMicroelectronics 45nm CMOS process using HVT, SVT and LVT libraries.

Controlling the supply voltage is hence a power-delay tradeoff: the power consumption decreases while the delay increases. That is why the supply voltage and the clock frequency have to be controlled together to guarantee the critical path (i.e., the slowest path between two clocked components). If the frequency is too high the data cannot be computed during a clock period, which means that the results sampled by the clocked components will probably be incorrect. Therefore the critical path delay has to be lower than the clock period, [POU 01]:

$$T_d\,(critical\ path) < \frac{1}{f_{clk}} \tag{5.6}$$

Clearly, it is required to decrease the clock frequency before decreasing the supply voltage and, respectively, increase the supply voltage before increasing the clock frequency. This principle is needed in all systems to guarantee the critical path: either

with a hardware solution like some delay lines, [DHA 01] and [DHA 02], or with a software technique at least. Scaling supply voltage and clock frequency together results in a reduction in the energy computation as well. While, scaling clock frequency alone is insufficient because, reducing the clock frequency does not only reduce the processor power consumption but also increase the computation execution time (which is to a first approximation linearly dependent on clock frequency). The clock speed reduction results in a computation taking more time but using the same total amount of energy. Because the power consumption is quadratically dependent to the voltage level, scaling the voltage level proportionally along with the clock frequency offers a significant total power/energy reduction while running a processor at a reduced performance level.

As a conclusion, adapting the supply voltage is very interesting when possible but this implies the use of Dynamic Frequency Scaling (DFS) to keep correct the system behavior. The addition of DFS to DVS is called DVFS and results in simultaneously managing the frequency and voltage which provides a good consumption-performance tradeoff.

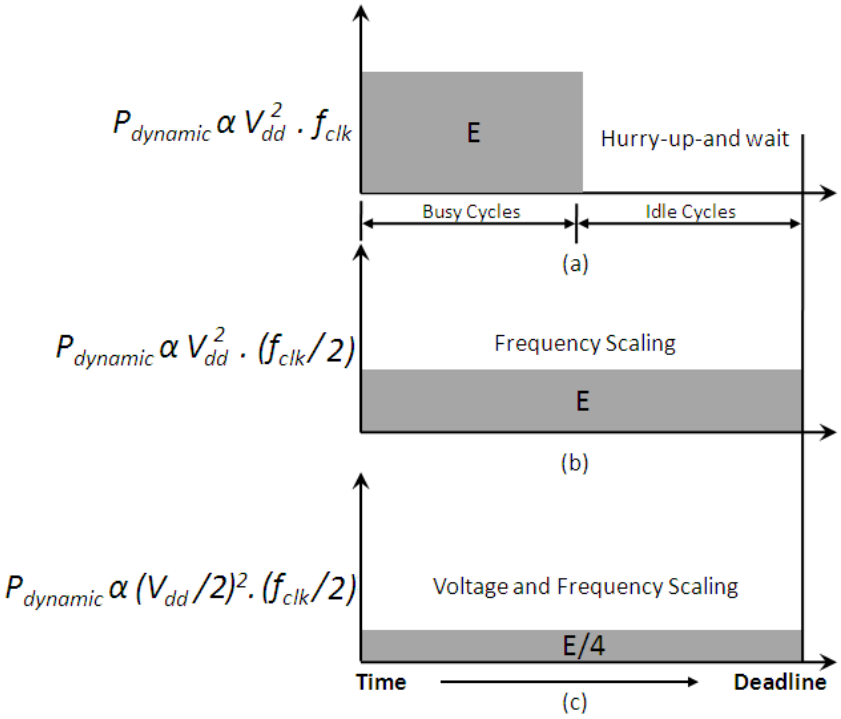


Figure 5.7: Energy Consumption vs. Power Consumption.

As applications do not require the full computational power at any time, it is possible to control speed and energy. In many cases, the only performance requirement is that the task meets a deadline, see Figure 1.3 (a). Such cases create opportunities to run the processor at a lower performance level and achieve the same perceived performance while consuming less energy. Figure 1.3 (b) shows that decreasing the processor clock frequency reduces power consumption but simply spreads the computation out over time, thereby consuming the same total energy as before. Figure 1.3 (c) shows that reducing the voltage level as well as the clock frequency achieve the desired goal of reduced energy consumption at an appropriate performance level [VAR 03].

5.5 Classification of DVFS Algorithms

For hard real-time systems, there are two kinds of scheduling approaches depending on the scaling granularity: intra-task DVFS (Intra-DVFS) and inter-task DVFS (Inter-DVFS). The intra-task DVFS algorithms adjust the voltage and the frequency within an individual task boundary, [SHI 01] and [GRU 01], while the inter-task DVFS algorithms determine the voltage and the frequency on a task-by-task basis at each scheduling point. The main difference between them is whether the slack times are used for the current task or for the tasks that follow. Inter-DVFS algorithms distribute the slack times from the current task for the following tasks, while Intra-DVFS algorithms use the slack times from the current task for the current task itself.

5.5.1 Intra-Task DVFS

In scheduling hard real-time tasks, in order to guarantee the timing constraint of each task, the execution times of tasks are usually assumed to be the worst case execution times (WCETs). However, since a task has many possible execution paths, there are large execution time variations among them. So, when the execution path taken at run time is not the worst case execution path (WCEP), the task may complete its execution before its WCET, resulting in a slack time. In that case, Intra-DVFS exploits such slack times and adjusts the processor speed. Intra-DVFS algorithms can be classified into two types depending on how to estimate slack times and how to adjust speeds.

Path Based Method

In the path-based Intra-DVFS, the voltage and the clock frequency are determined based on a predicted reference execution path, such as WCEP. For example, when the actual execution deviates from the predicted reference execution path (say, by a branch instruction), the clock speed is adjusted. If the new path takes significantly longer to complete its execution than the reference path, the clock speed is raised to meet the deadline constraint. On the other hand, if the new path can finish its execution earlier than the reference path, the clock speed is lowered to reduce the energy consumption. In the path-based Intra-DVFS, program locations for possible speed scaling are identified using static program analysis [SHI 01] or execution time profiling [LEE 00].

Stochastic Method

The stochastic method is based on the idea that it is better to start the execution at a low speed and accelerate the execution later when needed than to start with a high speed and reduce the speed later when slack time is found. By starting at a low speed, if the task finishes earlier than its WCET, it does not need to execute at a high speed. Theoretically, if the probability density function of execution times of a task is known a priori, the optimal speed schedule can be computed, [GRU 01]. Under the stochastic method, the clock speed is raised at specific time instances, regardless of the execution paths taken. Unlike the path-based Intra-DVFS that can utilize all the slack times of the task in scaling speed, the stochastic Intra-DVFS may not utilize all the potential slack times.

5.5.2 Inter-Task DVFS

Inter-DVFS algorithms exploit the "run-calculate-assign-run" strategy to determine the supply voltage and the appropriate clock frequency, which can be summarized as follows: (1) run a current task, (2) when the task is completed, calculate the maximum allowable execution time for the next task, (3) assign the supply voltage with the corresponding clock frequency for the next task, and (4) run the next task. Most Inter-DVFS algorithms differ during step (2) in computing the maximum allowed time

for the next task τ which is the sum of WCET of τ and the slack time available for τ. A generic Inter-DVFS algorithm consists of two parts: slack estimation and slack distribution. The goal of the slack estimation part is to identify as much slack times as possible while the goal of the slack distribution part is to distribute the resulting slack times so that the resulting speed schedule is as uniform as possible. Slack times generally come from two sources; *static slack times* are the extra times available for the next task that can be identified statically, while *dynamic slack times* are caused from run-time variations of the task executions.

A. Slack Estimation Methods

(1) Static Stack Estimation

Maximum constant speed

One of the most commonly used static slack estimation methods is to compute the maximum constant speed, which is defined as the lowest possible clock speed that guarantees the feasible schedule of a task set, [SHI 00]. For example, in earliest deadline first (EDF) scheduling, if the worst case processor utilization (WCPU) U of a given task set is lower than 1.0 under the maximum speed f_{max}, the task set can be scheduled with a new maximum speed $f'_{max} = U. f_{max}$. Although more complicated, the maximum constant speed can be statically calculated as well for RM scheduling, [SHI 00] and [GRU 01].

(2) Dynamic Stack Estimation

Three widely-used techniques of estimating dynamic slack times are briefly described below.

Stretching to Next Task Arrival time (NTA)

Even though a given task set is scheduled with the maximum constant speed, since the actual execution times of tasks are usually much less than their WCETs, the tasks usually have dynamic slack times. One simple method to estimate the dynamic slack time is to use the arrival time of the next task, denoted by NTA, [SHI 00]. Assume that the current task τ is scheduled at time t. If NTA of τ is later than (t + WCET (τ)), task τ can be executed at a lower speed so that its execution completes exactly at the NTA. Figure 5.8 shows examples of the Stretching-to-NTA method. When a single task τ is

activated as shown in Figure 5.8 (a), the execution of τ can be stretched to NTA. When multiple tasks are activated, there can be several alternatives in stretching options. For example, the dynamic slack time may be given to a single task or distributed equally to all activated tasks. Cases I and II of Figure 5.8 (b) illustrate these two options, respectively.

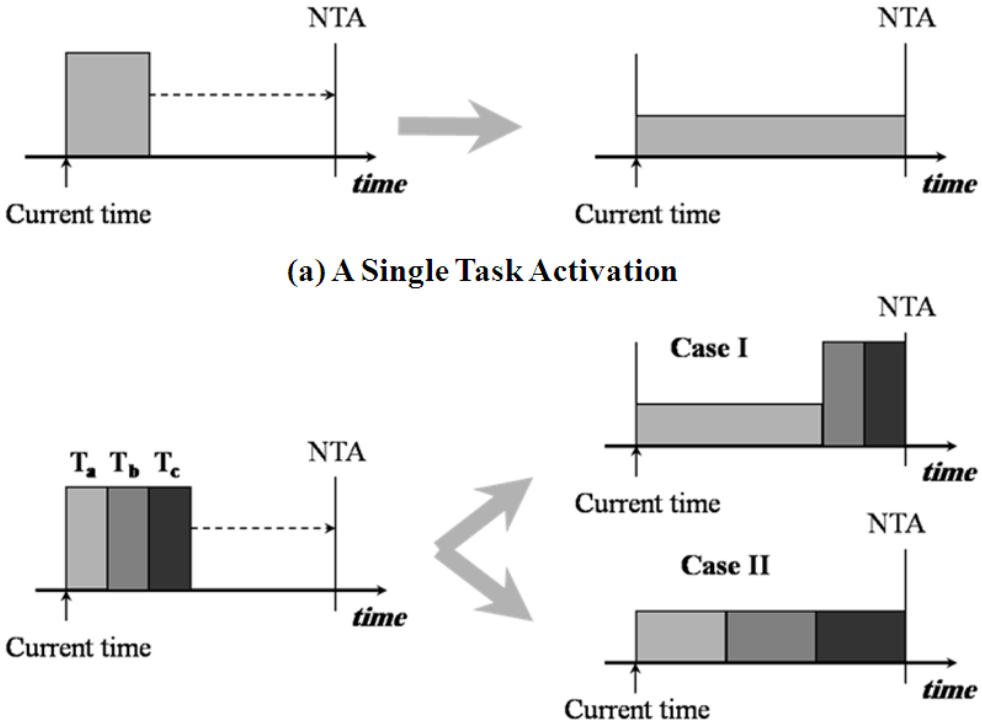


Figure 5.8: Examples of Stretching to NTA.

Priority-based slack stealing

This method exploits the basic properties of priority-driven scheduling such as EDF. The basic idea is that when a higher-priority task completes its execution earlier than its WCET, the following lower-priority tasks can use the slack time from the completed higher-priority task. It is also possible for a higher priority task to utilize the slack times from completed lower priority tasks. However, the latter type of slack stealing is computationally expensive to implement precisely. Therefore, the existing algorithms are based on heuristics, [AYD 01] and [KIM 02].

Utilization updating

The actual processor utilization during run time is usually lower than the worst case processor utilization. The utilization updating technique estimates the required processor performance at the current scheduling point by recalculating the expected worst case processor utilization using the actual execution times of completed task instances, [PIL 01]. When the processor utilization is updated, the clock speed can be adjusted accordingly. The main merit of this method is its simple implementation, since only the processor utilization of completed task instances have to be updated at each scheduling point.

B. Slack Distribution Methods

In distributing slack times, most Inter-DVFS algorithms have adopted a greedy approach, where all the slack times are given to the next activated task. This approach is not an optimal solution, but the greedy approach is widely used because of its simplicity.

Table 5.1 summarizes all previously mentioned DVFS scheduling algorithms with the main characteristics (i.e. scaling method and slack computation) of each technique.

Table 5.1: Classification of DVFS Algorithms.

DVFS Algorithm	Scaling Method	Slack Computation
Intra-DVFS	Path-based method	**Off-Line**
	Stochastic method	
Inter-DVFS	Maximum Constant Speed	
	Stretching to NTA	**On-Line**
	Priority-based slack stealing	
	Utilization Updating	

5.6 DVFS Architecture Overview

DVFS is a technique that allows the processor to dynamically alter their voltage and speed at runtime under the control of voltage scheduling algorithms. Implementing DVFS in a general-purpose microprocessor system includes three key components:

1. An operating system that can smartly vary the processor speed.
2. A regulation loop which generates the minimum voltage required for the desired speed.
3. A microprocessor which operates over a wide voltage range.

Nowadays, because of the instruction complexity and the access time to memory through different levels of cache, it is extremely hard to predict the execution time of a program and consequently the real speed of a processor. DVFS implementation generally assumes that the speed of the processor is constant (do not care about the cache mechanisms) and takes the worst case. With such an assumption, the processors run faster than the minimal speed enabling the highest energy savings. With a feedback system, it is possible to control with a high accuracy the power/energy consumed by the circuit. This is done respecting all the specifications and needs of the applications executed by the processor.

Fig. 5.9 shows the block diagram of the DVFS system. The sensor integrates an instruction counter and the clock is used as a time reference. Because the processor informs the sensor every time an instruction completes, it can calculate the real speed of the processor in Millions Instructions per Second (MIPS), averaged on a period of computation. Indeed, each executed application indicates to the co-processor through a software layer (OS) the speed it requires. The information about the speed can be inserted statically into the code at compile time (API - Application Programming Interface – function calls in the code) or computed dynamically at run time by the operating system. Therefore combining the information about the real-time requirements (estimated in MIPS) of the applications and OS running on the microprocessor enables us to create a computational load profile with respect to time. Consequently, using this profile it is

possible to apply a fine-grain power/energy management (software) allowing application deadlines to be met. Note that task scheduling is an important issue when using this technique. Sophisticated scheduling policies, as the one proposed in, [ZHU 03], can easily take advantage of this work by simply interfacing to the hardware features proposed using an API. The co-processor integrates with the sensor two main parts, the load error prediction unit and the digital controller part. The load error prediction computes the amount of error in the load profile (i.e. the difference between the set point sent by the OS and the calculated average speed by the sensor). The digital controller part dynamically controls the voltage/frequency scaling hardware which integrates a DC/DC converter to scale the supply voltage as well as the programmable clock generator to control the clock frequency in order to satisfy the application computational needs with an appropriate management strategy. This leads to a lower supply voltage and frequency reducing the system power/energy consumption. Indeed, the lower the processor speed, the lower the power consumption.

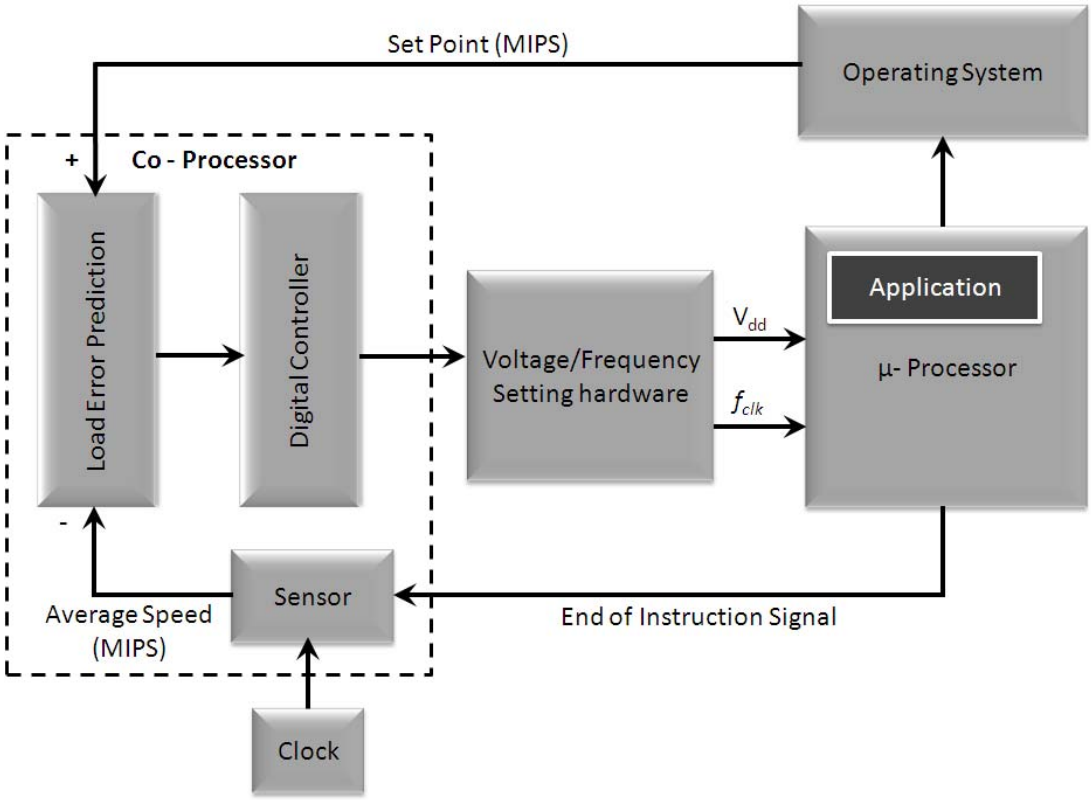


Figure 5.9: Voltage/Frequency Control Simplified Architecture.

There are many ways to control the voltage/frequency scaling hardware (Proportional Integral Derivative (PID) controller, fuzzy logic compensators, etc). Since the control is performed by the digital controller of the co-processor, this enables us to choose the way the control operates. The control depends on the microprocessor and the DC/DC converter. For each circuit, it is necessary to perform a complete characterization of the whole system in order to tune the control parameters at best. For instance, the period of the external sensor clock is crucial since it determines the accuracy of the calculated average speed. Moreover, it determines the system speed response. In fact the coprocessor sensor integrates also a register to memorize the number of executed instructions (i.e. the counter output) on a predefined period and determines the average speed on each rising-edge of the clock. If this period is short, the system will be fast but the calculated average speed will not be accurate. On the opposite, a long period leads to a slow system but to a more accurate speed. Therefore the period used to estimate the speed must be chosen carefully

5.7 Conclusions

The CMOS power consumption in the nanometric scales is becoming a really important design constraint. Therefore, optimization is required at all circuit levels in order to get low-power circuits with an acceptable level of performance. The goal of this chapter was to provide to the user the necessary concepts to understand and well characterize the causes of different sources of CMOS power dissipation. From which, designers can choose the appropriate way to reduce the effect of each contributing part in a CMOS circuit. Therefore different low-power solutions were investigated, for example, the impact of leakage power can be reduced using multi-threshold, body biasing or power gating techniques. On the other hand, the impact of dynamic power consumption can be reduced by clock gating or DVFS techniques. Since supply voltage scaling affects both static and dynamic power, DVFS with power gating are chosen during the rest of the thesis as the promising techniques for a global CMOS power reduction. Therefore different DVFS algorithms and the main DVFS architecture were explained in details.

Chapter 6

Controlling Uncertainty and Handling the Process Variability

6.1 Introduction

Due to continuous scaling of CMOS technology, systems containing a large number of processors and on-chip memories have become a reality. Consequently, novel architectures that allow various cores communicate to each other via the Network-on-Chip (NoC) paradigm have emerged as a promising alternative to traditional bus-based or point-to-point communication solutions [DAL 01]. By eliminating global wires, the NoC approach provides the needed scalability and predictability, while facilitating design reuse. Moreover, the NoC approach offers a matchless platform for implementing the globally asynchronous, locally synchronous (GALS) design paradigm, [CHA 84] and [MUT 00]; this makes the clock distribution and timing closure problems more manageable. In addition, a GALS design style fits nicely with the concept of different voltage-frequency domains, which provide better power-performance tradeoffs than its single voltage, single clock frequency counterpart, while taking advantage of the natural partitioning and mapping of applications onto the NoC platform.

Besides this huge increase in system complexity, systems designed in nanoscale technologies suffer from systematic and random variations in process, voltage, and temperature (PVT). Nowadays, within die variations play an increasingly important role in system power consumption and performance as the technology scales down [BOR 03]. Since designers cannot rely on the accuracy of the nominal parameter values, there is a tremendous need for on-line techniques that can cope with such dynamic variations. More precisely, there is a need for efficient algorithms and built-in circuitry able to adapt the system behavior to workload variations and, at the same time, cope with the parameter variations which cannot be predicted or accurately modeled at design time.

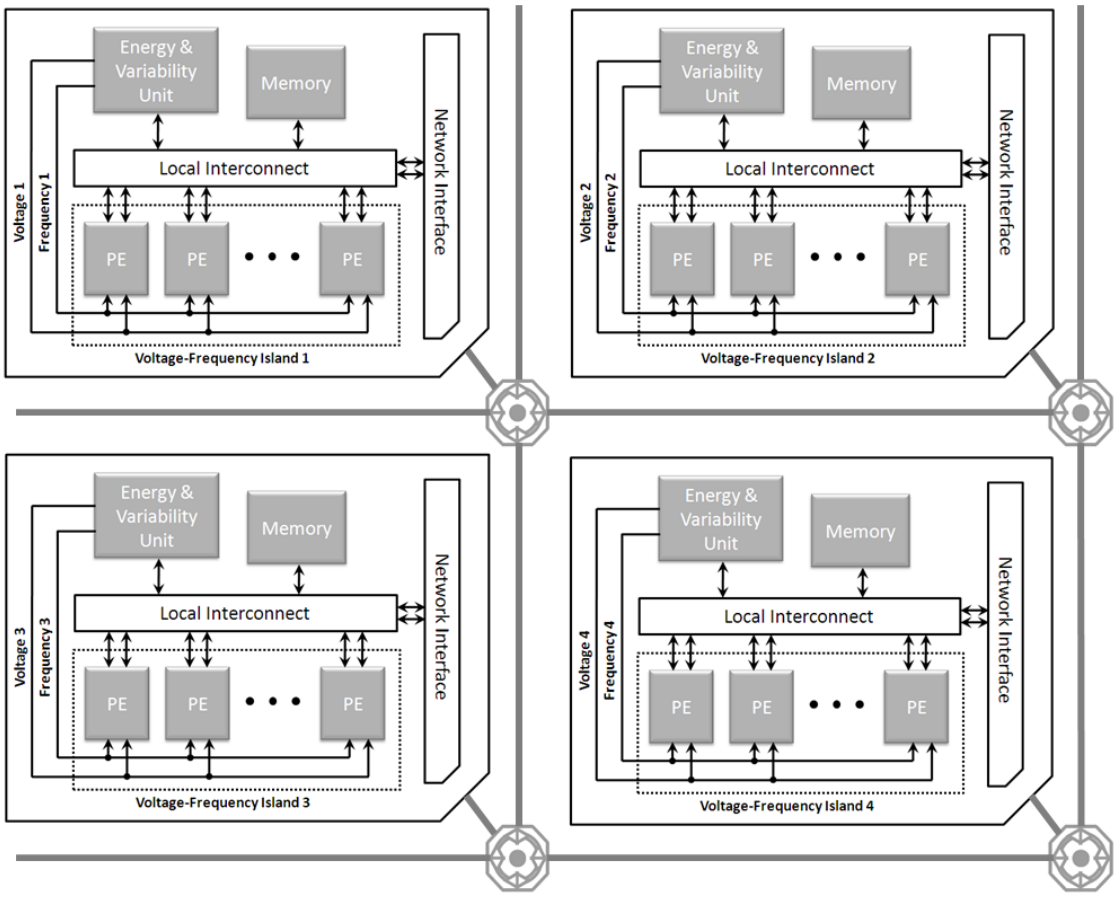


Figure 6.1: A NoC Architecture with Multiple Voltage-Frequency Domains.

Not surprisingly, designing appropriate dynamic voltage and frequency scaling (DVFS) control algorithms for run-time control of different voltage-frequency domains in a GALS system is a matter of great importance. While this problem has been addressed before by a number of authors [OGR 08], [WU 04] and [NIY 05], no attention has been paid to add the feature of controlling the impact of manufacturing process variations to the capabilities of the DVFS controllers. Starting from these overarching ideas, in this chapter, we specifically focus on proposing a new NoC architecture, which have a process variability robust adaptive control technique. This control technique can improve the performance, power consumption, and reliability of future NoC architectures in a synergistic manner. We consider NoC architectures consisting of multiple voltage-frequency islands, or voltage-clock domains, as shown in Figure 6.1. Each island is a synchronous region, i.e., the cores within the same island share a single clock and supply

voltage that can be controlled independently of other islands. Communication across different clock domains is asynchronous and it can be achieved through the use of the control switches integrated within the PSTR as presented previously in Chapter 4.

The contribution of our work is twofold: First, we propose a novel methodology to dynamically control the speed of each island. The controller not only considers the dynamic workload variations, but also ensures that the operating frequency does not exceed the maximum allowed value for a given process variability effect. Then, on the following Chapter we will study, analyze and design a complete circuit for the adaptive control system and its different parts on a real processor on STMicroelectronics 45nm CMOS technology.

6.2 Related Work Contributions

Generally speaking, feedback control reduces the systems sensitivity to parameter variations. So, design of nanoscale MPSoCs is likely to benefit from the systematic approaches from modern control theory [REB 10] and [JUA 05]. As such, in [MAR 02], the authors propose using adaptive body biasing and dynamic voltage scaling simultaneously to reduce power in processors with a single clock domain. Partitioning NoCs into multiple voltage-clock islands to minimize the energy consumption is considered in [OGR 07]. The power management for NoCs have been recently considered by several authors. The work in [SIM 04] presents a stochastic power management technique for NoCs. In [SHA 03], the authors present a run-time technique to satisfy peak power consumption constraints in interconnection networks by controlling the local power consumption of each router. An on-line DVFS algorithm based on proportional-integral-derivative (PID) controller for multiple clock domain processors is presented in [WU 04]. PID control requires manual tuning of the control gain which may become prohibitive as the number of voltage-clock domains increases. To address these shortcomings, we develop the architecture of the voltage-frequency domains used within a NoC, put limits on the maximum operating clock frequency for each voltage-frequency island according to the impact of the process variability effect on it and take the advantage of tools from modern control theory to build a process variability robust energy controller. More precisely, we propose:

- A novel NoC architecture model for multiple voltage-frequency island design
- Activity monitors integration in different voltage-frequency islands to limit the maximum allowable clock frequency used according to the process variability impact on this island.
- Feedback control algorithms that control the speed of each voltage-frequency island to cope with the PVT and workload variations.

6.3 Process Variability Robust NoC Architecture

In a GALS system, process variability and fabrication yield can be improved by smartly removing tasks over fault or low performance nodes and assign them into other high performance ones, as shown in Figure 6.2. As each processing node performance could be measured by some kind of an activity monitor, a global system manager is able to distribute the tasks over the nodes. The task assignment takes into account the node performances and the task processing loads. The main target of this manager is to guaranty an overall chip performance. With this kind of approach, it is no more required to separately guaranty a performance for each node which relaxes the fabrication constraints and permits a yield enhancement.

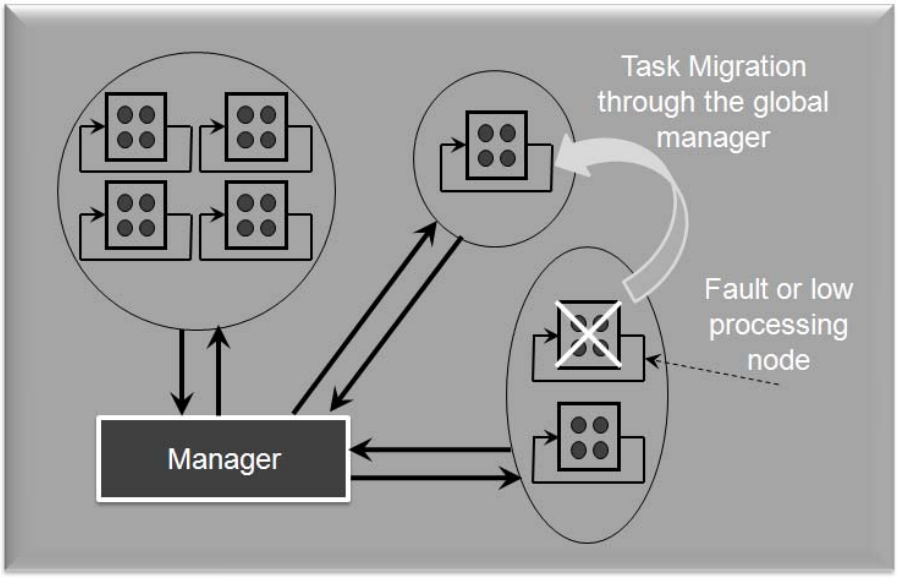


Figure 6.2: Fabrication yield control by distributing tasks over different processing nodes in a GALS system.

Based on the previous ideas we propose a solution to manage the impact of process variability in a multiprocessor GALS system. In a GALS system, we can choose to slow down some parts of the circuit while allowing others to operate at the maximal frequency. This enables more energy saving opportunities than conventional systems built around one processor and allows adapting the clock speed to the local process quality [ZAK 10b] and [REB 10]. Moreover, it has also been shown that multiple-clock designs (GALS) with voltage scaling are even better not only in terms of power and performance, but also in terms of variability [MAR 05]. As a result building a system based on the implementation of hardware resources whose performances are unpredictable at the fabrication time requires having total management strategies of the performance by adaptation of the voltage/frequency in order to respect the real time constraints of the application and the allocated energy budget. So it is proposed to use automatic feedback loops based on:

1. Measurement of the real local performances of silicon and the actuation of the parameters voltage/frequency (hardware level).
2. The suitable hardware resource allocation for the execution of a task in the assigned time/energy budget (operating system level).

The idea is the use of dynamic DVFS with task scheduling techniques to dynamically manage not only the energy budget but also the activity of the processing node based on advanced automatic control techniques. These techniques will allow an optimal regulation of the clock frequency and the supply voltage according to the computational load and the load distribution in the various GALS processors. In order to compensate for the process variation due to the technology dispersion, and optimize the operation of the circuit, the dynamic voltage/frequency regulation itself should be self-adjustable and robust against process variability with the variable loads and dispersion models. Implementing DVFS in a GALS system presents many design challenges:

1. Specific mechanisms are needed, allowing the different domains to communicate and synchronize in a reliable manner.
2. Splitting the computational load on the processors must be done as a trade-off with the communication load. Usually the communication systems consume a lot of energy.

3. Interdependencies between different clock domains impact the overall performances. A domain operating at a low speed tends to slow down the communications and to reduce the processor speed in communication.
4. Sensing the local process quality and processor speed can also be a design issue.

Many researches have been done in this field to develop DVFS and to guaranty a correct system behavior even if the process variations are huge. In [SEM 02], a GALS system model has been developed to gradually decrease the clock frequency when there are little observable changes in the workload. Compiler for controlling GALS systems have been developed too as in [MAG 03]. However, there are very few studies integrating power and process variation management together.

One of the propositions to handle the uncertainty of a processing node over a GALS system due to the impact of process variability and also to reduce its energy-consumption by means of automatic control methods is the use of activity monitors. The activity monitors can be embedded in each voltage-frequency island (i.e. processing node) in order to provide a real performance measurement of different voltage-frequency islands after the fabrication process, which will be used afterwards by the operating system to distribute tasks over different processing nodes and assign those low or fault processing node tasks over idle ones. This means the need for rescheduling tasks in each processing node to meet the new assigned deadlines. This will be achieved by controlling the supply voltage/ clock frequency, which in accordance will control the power/energy of consumption.

Figure 6.3 shows a simplified architecture of this control system using three different control loops [ZAK 10a]. The control loops are applied in different architecture levels: control in energy of consumption (supply voltage), control in the processing power (supply voltage/clock frequency) and in the management of the Quality of Service (QoS) provided by the application. Voltage, frequency and energy control loops are used in order to adapt the energy of consumption and the process variability effect. The other control loop is needed to deal with the QoS (at the application level), the limitation of processing and/or communication power and the constraints in energy consumption. Note that the QoS control loop is out of the scope of this thesis.

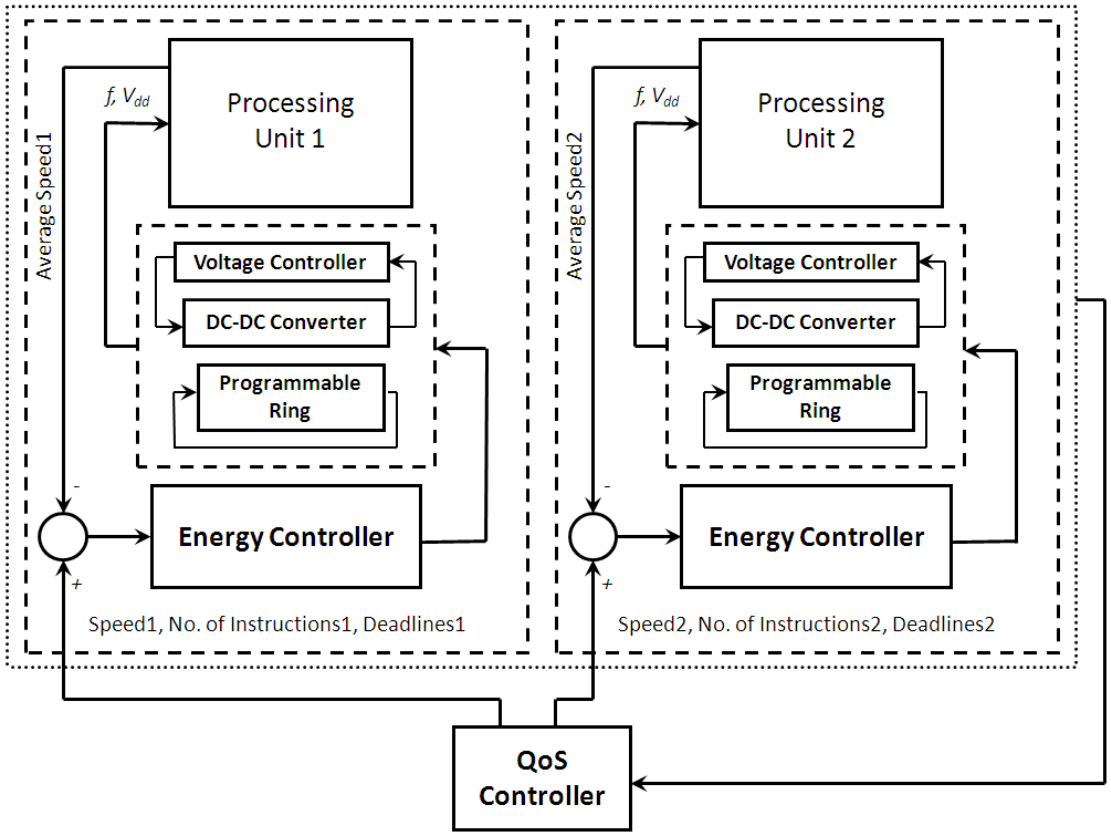


Figure 6.3: Energy/Performance Control Simplified Architecture.

In Figure 6.3, the QoS controller sends a set of information (required speed, No. of instructions, and deadlines for each instruction), which are given by the operating system (that can be statically inserted into the code or dynamically computed at run time by the OS). There are also sensors embedded in each processing unit in order to provide real-time measurements of the processor speed. Therefore, combining this information about the real-time requirements of the applications and the OS enables us to create a computational load profile with respect to time. Consequently, using such a profile makes possible to apply a fine-grain power/energy management allowing application deadlines to be met. Note that task scheduling is an important issue when using this technique. Sophisticated scheduling policies, as the one proposed in [ZHU 03], can easily take advantage of this work by simply interfacing to the hardware features proposed using an API (Application Programmable Interface). Here, the DVFS hardware part contains a DC-DC converter for voltage regulation and a programmable clock generator for frequency regulation. The energy controller dynamically controls the DC-DC converter to

scale the supply voltage as well as the clock frequency in order to satisfy the application computational needs with an appropriate management strategy. Note that, the maximum allowable clock frequency of the programmable oscillators is defined on the system start-up according to the evaluation of process variability impact on each voltage-frequency island. This information is received from an activity monitors located in each processing domain. By this way, we could manage the impact of process variability by the application of DVFS with task scheduling techniques, under the control of an Energy/Performance controller.

6.3.1 Overall Architecture

Using both a Network on Chip (NoC) distributed communication scheme and a GALS approach offers an easy integration of different functional units thanks to a local clock generation [KRS 07]. Moreover, it can allow better energy savings since each functional unit can easily have its own independent clock frequency and supply voltage. Hence, NoC architectures combined with a multi-power domain GALS system appear as natural enablers for distributed power management systems as well as for local DVFS. In this section, we propose a new architecture for the voltage-frequency islands in a GALS-NoC system. This architecture helps in modeling both DVFS and local process quality management.

In Figure 6.4, a detailed block diagram of the novel voltage-frequency island architecture in a GALS-NoC system is shown, [ZAK 10a]. Since in advanced technologies, the associated processor leakage has an important contribution to the system energy consumption, the Sleep Mode Management block is used to put useless processors into a sleep mode in order to limit their static power consumption. On the other hand, the Speed Sensor is used to calculate the *Real-Time Speed* of the processing node in MIPS. The Activity Monitors disseminated into each voltage-frequency island are used to locally evaluate the process quality in terms of its relative speed with respect to the other processing nodes (i.e. *Intrinsic Speed*). The *Intrinsic Speed* value defines the upper clock frequency bound of the operating voltage-frequency island. We have also two actuators: the DC-DC converter which provides the supply voltage and the Asynchronous Programmable Ring which generates the desired clock frequency to each

local processing node. The Digital Controller manages the voltage/frequency couple. It processes the error between the Unit Speed and the speed set point information extracted from the information sent by the operating system (i.e. No. of instructions and deadlines) within a closed loop system, and by applying a well-suited compensator sends the desired voltage and frequency code values to the DC-DC converter and to the clock generator. Consequently, the system is able to locally adapt their output voltage and clock frequency values clock domain by clock domain in order to limit their dynamic power consumption.

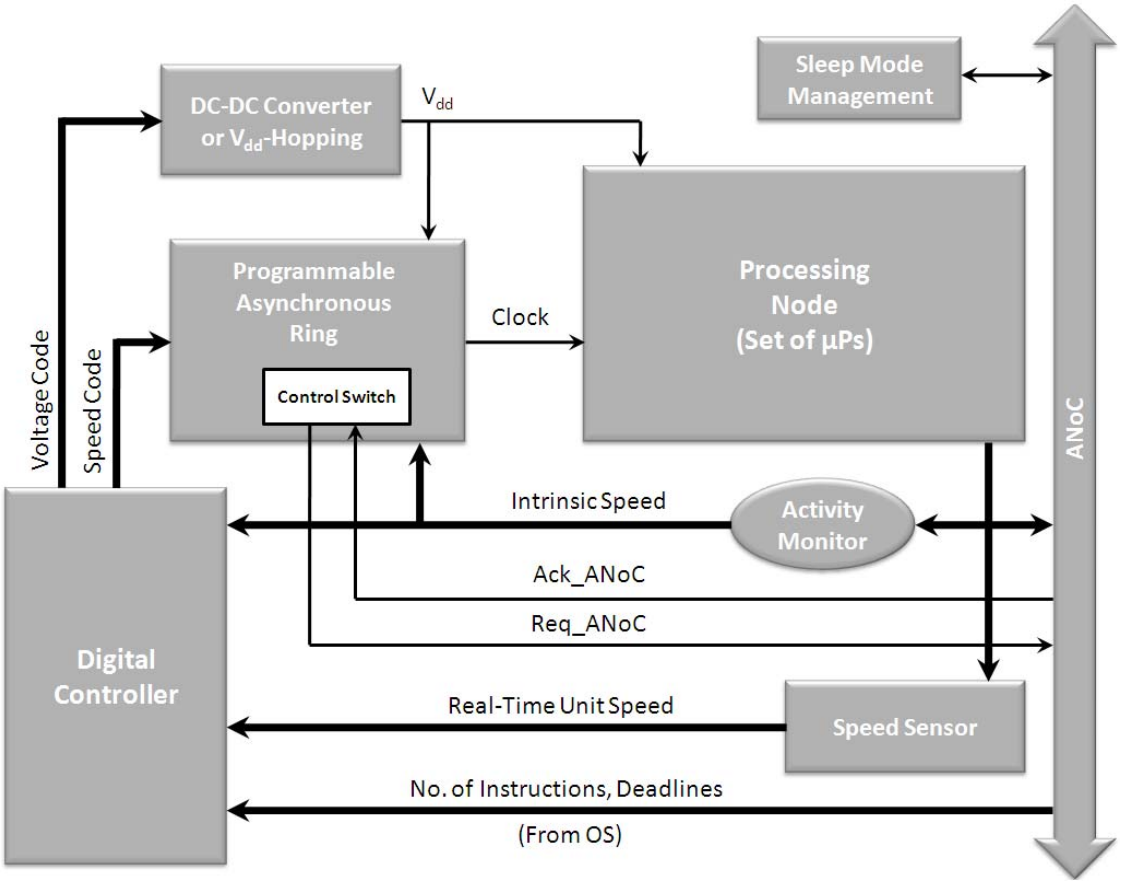


Figure 6.4: Energy/Performance Management Architecture for a Voltage-Frequency Island in a GALS-NoC System.

In this model, the ANoC (Asynchronous NoC) is the reliable communication path between the different clock domains. Therefore, the communication across different clock domains is asynchronous and it can be achieved through the use of the control switches integrated within the PSTR as presented previously in Chapter 4. This can fix the speed of data communications between any two different processing nodes to the

slowest communicating node. This local synchronization mechanism is based on Req_ANoC and Ack_ANoC handshake signals. As a result, we could have a secure communication with no metastability problem at the clock domain crossing and an adaptation to process variability. Within this framework, we explore two feedback control techniques, namely regulation and tracking, as follows:

1- **Start-Up Regulation**: For this scenario, we assume that the activity monitor located in each voltage-frequency island evaluates the impact of the manufacturing process variability within it in terms of the *Intrinsic Speed*. Subsequently, this value is sent through the ANoC to the OS, which will use it in evaluating the relative performance of the different NoC voltage-frequency islands. As a result, the OS can distribute tasks over different processing nodes, and assign those low or fault processing node tasks over idle ones. The task assignment takes into account the node performances and the task processing loads. Moreover, the *Intrinsic Speed* value is also used by the programmable oscillator to define the maximum allowable clock frequency to be generated for this processing node. So that, the maximum number of processing nodes in a chip could be used even at an acceptable lower performance level with respect to other ones. This guaranties the overall chip performance instead of separately guarantying each node performance, which relaxes fabrication constrains and permits more yield enhancement.

2- **On-line tracking**: This scenario is more general, in the sense that it can scale the voltage and clock frequency within a range of values in order to minimize the energy/power consumption, as well as provide robustness in operation. In this case, we have an upper bound on the operating clock frequency, which is previously defined for each voltage-frequency island by the activity monitors. Moreover, we assume there exists a lower bound on the operating frequency (hence on the voltage) for a subset of voltage-frequency islands that doesn't violate the critical path condition of their processing nodes. The proposed controller sets the speed of each island such that it minimizes the penalizing high voltage (hence clock frequency) run time while guaranteeing a computational speed performance.

6.3.2 Sensing the Computational Activity

As mentioned in Chapter 5 activity sensors play a critical role in DVFS systems and must be selected carefully. However we are facing many difficulties in order to implement a sensor that measures the processor activity accurately. Different assumptions were proposed, here is an example for two of them:

Current Activity Sensor

It consists of an analog solution to monitor the activity of the system with respect to the amount of current consumed. In [RIU 06], a monitor fabricated in a 90nm CMOS technology which is able to estimate the circuit activity is proposed. Unfortunately, this kind of sensors has very limited applications, for example in our case it's difficult to say if the activity is strong or not. Indeed, the current is data-dependant for a part and activity-dependant for the other part (we can say that the current gives an image of the executed instructions). Moreover, it is also difficult to plan the deadline of the task, so this sensor is not well-suited for our DVFS application.

Instruction Counter

There exist two possible solutions to implement this kind of activity sensing:

- *Direct instruction count*: by incrementing the counter each time a new instruction has been executed and calculates the average value with respect to a reference clock. It is a simple and efficient way to estimate the activity and the progress in a task. The limit is that it is difficult to accurately compute the time. Indeed, with CISC machine (not RISC), it is difficult to predict the remaining clock cycles or the speed for reaching a deadline.

- *Statically computing the clock cycles*: at compile time, we calculate a bound for the number of clock cycles. We estimate here, instruction by instruction, the number of cycles. This is a little bit more complex and more accurate. The gain is not formally proven compared to a simple counter.

6.3.3 DC-DC Converter

The DC/DC converter is a circuit that converts a voltage source (of direct current) from one voltage to another. Two kinds of DC/DC converter can be used. The first class is a continuous DC/DC converter which provides an accurate supply voltage, but with a weak efficiency. The second kind of converter used is a digitally controlled step-converter (V_{dd}-hopping converter) that has a better efficiency but discrete output values. The V_{dd}-hopping technique was modeled in our system, where its principle is described in [ALB 08] and [ALB 09]. Two voltage levels are available (V_{low} and V_{high}) and the one or the other could be achieved with a certain transition time and dynamics that depend upon the internal controller of the V_{dd}-hopping, as shown in Figure 6.5 [MIE 07].

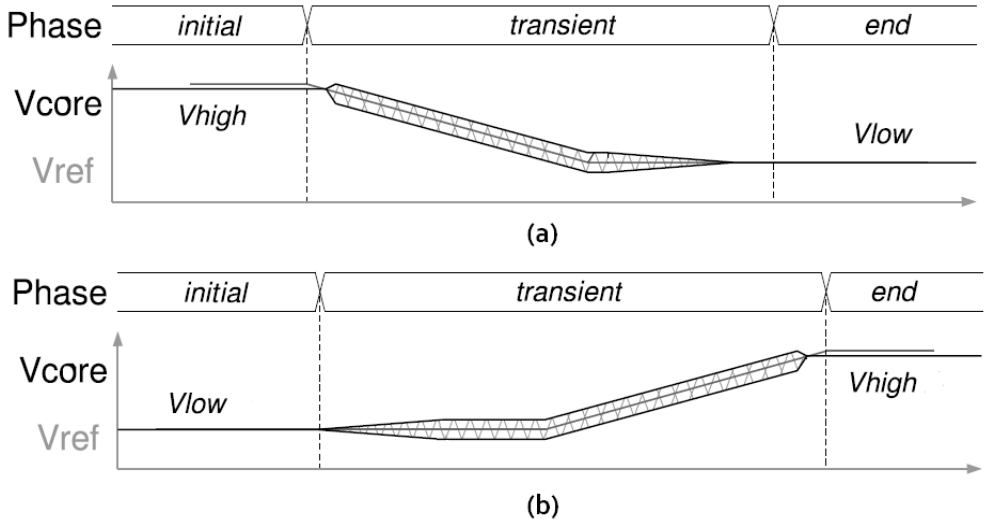


Figure 6.5: Chronogram of V_{dd}-Hopping (a) Falling and (b) Rising Transitions.

6.3.4 Frequency Controller

The frequency controller is a vital part in our proposed NoC architecture. Given that, our goal is to design process variability robust voltage-frequency islands in a NoC system which has the minimum possible energy consumption. This could be done by limiting the upper clock frequency bound of the different processing islands with respect to process variability impact on each. Subsequently, adapt their clock frequency to dynamic workload variations. Therefore, we have to carefully choose our clock frequency generator.

Asynchronous PSTR are considered promising solution for generating clocks. It has the low power consumption and process variability robustness advantages of asynchronous circuits, as it was shown in Chapter 3. Moreover, PSTR can be easily reconfigured in order to change its output clock frequency. Furthermore, PSTR output frequency has a linear change with respect to the supply voltage. Consequently, it could be supplied with the output of the V_{dd}-Hopping to give the corresponding clock frequency at different voltage levels. As a result, our design in the rest of the thesis will be mainly focused on the use of the asynchronous PSTR as the clock generator part in the system.

6.4 NoC Control Method

As the GALS-NoC voltage frequency islands can run at several voltage levels with an available frequency range for each level. In order to control the energy-performance tradeoff we propose to calculate a *Speed Set Point* to track, denoted ω_{sp} (in MIPS), for each task that the processing domain has to execute. This set point is based on some task information sent by the operating system: for each task T_i the operating system provides the computational load (i.e. the number of instructions C_i) and the deadline N_i, as shown in Figure 6.4.

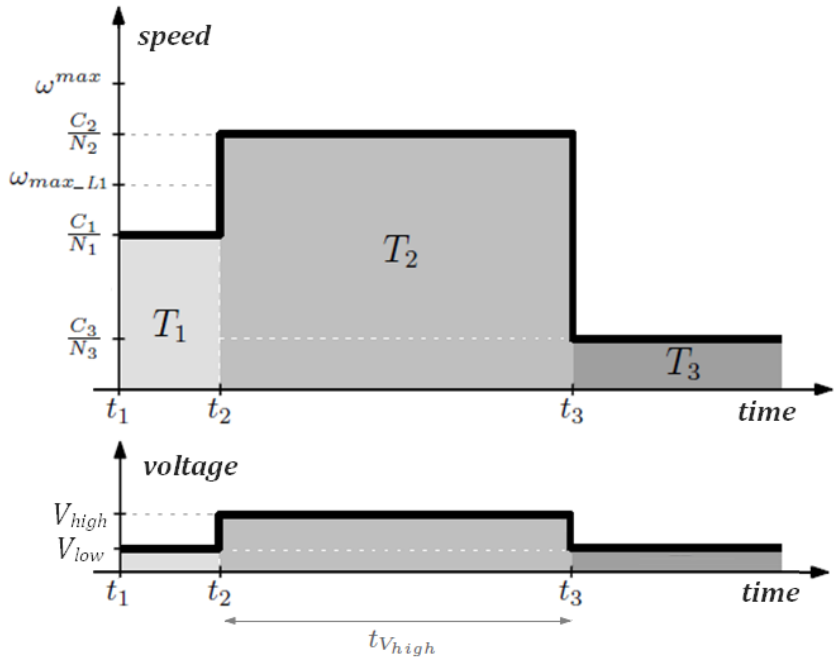
The presence of deadline and time horizon to compute tasks naturally leads to predictive control. Predictive control consists in finding an open-loop control profile over some time horizon and in applying it until the next time instant. The stability relies in the way the open loop control is chosen. The most classical strategy consists in taking the open-loop control that minimizes a cost function. The horizon can be constant, infinite or less classically contractive as in [DUR 09]. For further details on predictive control, one can refer to [MAY 90]. The key point of the present control problem will be the choice of the open-loop strategy and its computational cost. Indeed, if predictive control is known to be a robust approach, it is also often associated to high computational cost which is not acceptable in the present case. The strategy adopted here is called fast predictive control and consist in taking advantage of the structure of the dynamical system to fasten the finding of the open-loop control, [ALA 06]. The simplicity of the system considered here is therefore very suitable for such strategies.

Two speed set point building methods are illustrated on Figure 6.6 for a three-task example. The digital controller is an upstream from the V_{dd}-Hopping and the asynchronous PSTR, where it calculates the voltage and frequency values that have to be sent to these actuators, in order to minimize the energy consumption while guaranteeing the computational performance [DUR 09a]. Therefore, the control behavior is restricted to only two voltage levels $V_{level} = \{V_{high}, V_{low}\}$ with three different frequency levels $F_{level} = \{F_{high}, F_{low1}, F_{low2}\}$, where F_{high} is the maximum possible frequency at V_{high}, F_{low1} is the maximum possible frequency at V_{low}, and $F_{low1} > F_{low2}$. Let ω^{max}, ω_{max_L1} and ω_{max_L2} denotes the processing unit speed at F_{high}, F_{low1} and F_{low2} respectively. An intuitive method would consist in building the average speed set point of each task that is the ratio C_i/N_i. With such an approach the C_i instructions are performed and ended at N_i, Figure 6.6 (a). However, this method is not energy-efficient. Indeed, one could see that, as soon as the average speed set point of a task is higher than ω_{max_L1} - such as for the task T_2. The task has to be executed all the time at V_{high} to not miss the deadline. Therefore, the chip will consume a lot by running the whole task at the penalizing highest supply voltage. A solution to avoid that is to split the tasks into two time intervals, as follows:

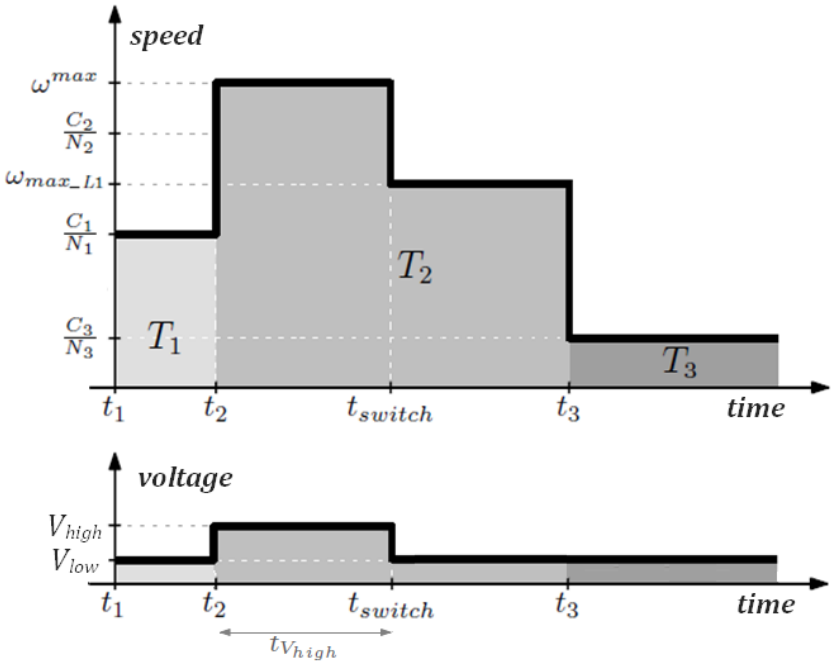
i) First (if required), the chip begins to run at V_{high} with the maximum available frequency in order to achieve the maximum possible speed ω^{max} - such as for T_2 from t_2 to t_{switch} on Figure 6.6 (b).

ii) Then, the task can be finished at the low voltage with a speed under ω_{max_L1}. The switching time to go from V_{high} to V_{low} - denoted t_{switch} in the example - has to be suitably calculated in order that the performed number of instructions during the two time intervals corresponds to the number of instructions to do C_i, i.e. the task fits with its deadline and the performances are so guaranteed.

Thus, this proposal allows reducing the penalizing high voltage running time tV_{high} and so the energy consumption as much as possible since the maximum available frequency is automatically used when the system is running at high voltage.



(a) Intuitive Average Computational Speed Set Point Building.



(b): Energy-Efficient Speed Set Point Building.

Figure 6.6: Different Computational Speed Set Point Buildings and their Impact on the Energy Consumption (i.e. the Penalizing High Voltage Running Time).

Indeed, the computational load used to treat a given task is equal with both strategies since the darken surface areas of the task T_2 are respectively the same on both drawings in Figure 6.6. Nevertheless, the switching time t_{switch} could not be *a priori* known and therefore, a predictive control law is required to dynamically calculate this switching time [DUR 09b]. That point will be detailed in the following subsections. Furthermore, the task information given by the operating system is not enough anymore to build such an energy-efficient computational speed set point. We also need some information about the system resources, such as the maximum speed for the different voltage levels, i.e. ω^{max}, ω_{max_L1} and ω_{max_L2}. Moreover, we need to know what it has already been done, and for this reason we propose to use a closed-loop system in order to predict the minimum high voltage running time.

6.4.1 Fast Predictive Control

To minimize the energy consumption the system has to run the shortest possible time with the penalizing high supply voltage. Thus, the controller dynamically calculates if the system needs to run at V_{high} (and at ω^{max}) or at the low voltage level (and a speed lower than ω_{max_L1}) will be enough to compute the task before its deadline. This principle could be formulated as a predictive control problem: for each task T_i, what is the speed set point which will minimize the high voltage running time tV_{high} while guaranteeing that the executed instruction number is equal to the number of instructions C_i to do:

$$\min t_{V_{high}} / \int_{N_i} \omega \, dt = C_i$$

Nevertheless, the speed set point can be obtained in an easier and faster way. We simply need to know what the processor has to do and how much time is available to do it. Let ω_j and ω_{j+1} are the two computational speeds which are immediate neighbors to "predicted average speed" δ, (i.e. $\omega_j > \omega_{j+1}$ and $\omega_j > \delta \geq \omega_{j+1}$). The aim of the controller is to dynamically calculate δ which is required to perform the task exactly on its deadline, in order to know if the task has to be executed at the more penalizing voltage level V_j or on the contrary if V_{j+1} will be enough to perform it before its deadline. The value of δ can be easily described as the ratio between what the processor has to do to compute the task

minus what it has already done (that is corresponding to what it remains to do) and the remaining time before the end of the task, this can be mathematically expressed as:

$$\delta(t_{k+1}) = \frac{C_i(t_k) - \sum_{t_i}^{t_{k-t_i}} \omega(t_k)}{L_i(t_k)} \tag{6.1}$$

where t_i is the beginning of the task T_i, t_i is the current time and t_{k+1} is the next sampling time. L_i denotes, the remaining available time to complete a given task. This equation can then be implemented as following:

$$\Omega(t_k) = \Omega(t_{k-1}) + T_s \,.\, \omega(t_k)$$

$$\delta(t_{k+1}) = \frac{C_i(t_k) - \Omega(t_k)}{L_i(t_k)} \tag{6.2}$$

where Ω is the integration of the computational speed ω, T_s is the sampling period and t_{k-1} is the last sampling time. Moreover, a conditional instruction is added to be coherent with Equation (6.1): indeed, the computational speed ω is integrated on the running time of each task and so when a task execution is completed, which means in the last sampling time before its deadline, the variable Ω is reset to zero.

The computational speed set point is then deduced from the dynamic value of δ and so are the voltage and frequency levels. Indeed, the device runs with the more penalizing neighboring speed ω_j during a certain time. As the running computational speed of the device is higher than the average predicted speed because $\omega_j > \delta$, the value of δ decreases until achieving $\delta(t_{k+1}) = \omega_{j+1}$, which means that the task can be finished with the less penalizing neighboring speed ω_{j+1}.

To conclude, this control method is called a fast predictive control technique because the strategy consists in predicting the time when the system has to switch from the more penalizing voltage level to the less one, but to calculate this switching time, the controller does not need to compute a complex optimization algorithm since the computation is easy and immediate. Moreover, the computational performances are guaranteed because the speed set point to track is always higher or equal than required. Therefore, the control law is robust thanks to the prediction [DUR 11].

According to the two voltage levels and the three frequency levels we have previously defined with the corresponding computational speed at each of them, the control algorithm can be defined as:

$$\begin{aligned}
&\text{If} && \delta(t_k) > \omega_{\max_l1} \\
& && \begin{cases} V_{level}(t_{k+1}) = V_{high} \\ F_{level}(t_{k+1}) = F_{high} \end{cases} \\
&\text{Else if} && \delta(t_k) > \omega_{\max_l2} \\
& && \begin{cases} V_{level}(t_{k+1}) = V_{low} \\ F_{level}(t_{k+1}) = F_{low1} \end{cases} \\
&\text{Else} && \\
& && \begin{cases} V_{level}(t_{k+1}) = V_{low} \\ F_{level}(t_{k+1}) = F_{low2} \end{cases}
\end{aligned} \tag{6.3}$$

6.4.2 Tracking Efficiency

The proposed control strategy is easy to implement, except for the parameters ω_m, i.e. the possible computational speeds when the device is supplied with the voltage V_m and the clock frequency f_m. Since we would like to have a controller robust to process variability where the value of f_m and V_m are not known *a priori* and could vary, therefore we propose to estimate ω_m.

Let $\widetilde{\omega}_m$ denotes the estimation of the computational speeds. The solution consists in measuring the speed for each couple of voltage/frequency levels. Therefore all the speeds ω_m are measured when the system is running with the supply voltage V_m and the clock frequency f_m. In order to know that we use the previous values of the control variable and to filter the fluctuation of the measured speed, a weighted mean is used, with the weighted value $0 \leq \rho \leq 1$. The following algorithm summarizes that principle:

$$\begin{aligned}
&\text{If} && \begin{cases} V_{level}(t_k - 1) = V_{level_m} \\ F_{level}(t_k - 1) = F_{level_m} \end{cases} \\
&\text{Then} && \widetilde{\omega}_m(t_k) = (1-\rho).\widetilde{\omega}_m(t_{k-1}) + \rho.\omega_m(t_k)
\end{aligned} \tag{6.4}$$

However, a problem could appear during the voltage transitions. Indeed, the algorithm (6.3) allows to dynamically calculate the predicted average speed δ and to compare this value with the computational speeds ω_m (in fact with the estimation of the speeds $\widetilde{\omega}_m$). For example, for a given task the system runs with the voltage V_{high} and the frequency F_{high} while δ is higher than $\widetilde{\omega}_{\max\_l2}$, but as soon as δ becomes lower than $\widetilde{\omega}_{\max\_l2}$ the controller changes the voltage and frequency levels. Nevertheless, during this level transition the estimated speed could also vary (due to the fluctuations in the estimation) and so becomes lower than the current value of δ. Because of this phenomenon, the levels could hence switch and switch again and a solution is so required. For this reason, we propose bounding the value of ρ in order that the variation of the estimation is always lower than the variation of δ.

First, let $\Delta\widetilde{\omega}_m$ denotes the variation of the estimation of the computational speeds, obtained from Equation (6.4):

$$\Delta\widetilde{\omega}_m(t_k) = \frac{\widetilde{\omega}_m(t_k) - \widetilde{\omega}_m(t_{k-1})}{T_s}$$

$$\Delta\widetilde{\omega}_m(t_k) = \frac{\rho}{T_s} \cdot [\omega(t_k) - \widetilde{\omega}_m(t_{k-1})] \tag{6.5}$$

Then, let $\Delta\delta$ denotes the variation of the predicted average speed. By substituting the value of $\Omega(t_k)$ into Equation (6.2), then $\delta(t_{k+1})$ can be expressed as:

$$\delta(t_{k+1}) = \frac{C_i(t_k) - \Omega(t_{k-1})}{L_i(t_k)} - \frac{T_s \cdot \omega(t_k)}{L_i(t_k)} \tag{6.6}$$

As the number of instructions C_i usually does not change for a given task, then $C_i(t_k) = C_i(t_{k-1}) = C_i$. Moreover, $L_i(t_k) = L_i(t_{k-1}) - T_s \simeq L_i(t_{k-1})$ for small values of T_s. Consequently, the first term on the right hand side of Equation (6.6) can be approximated as $\delta(t_k)$. As a result, the variation of δ is given by:

$$\Delta\delta(t_k) = \frac{\delta(t_{k+1}) - \delta(t_k)}{T_s} = -\frac{\omega(t_k)}{L_i(t_k)} \tag{6.7}$$

Finally, we need to get the condition for which $|\Delta\widetilde{\omega}_m(t_k)| \leq |\Delta\delta(t_k)|$. From Equations (6.5) and (6.7), this can be expressed as follows:

$$\frac{\rho}{T_s} \cdot |\omega(t_k) - \widetilde{\omega}_m(t_{k-1})| \leq \frac{\omega(t_k)}{L_i(t_k)}$$

$$\rho \leq \frac{T_s \cdot \omega(t_k)}{L_i(t_k) \cdot |\omega(t_k) - \widetilde{\omega}_m(t_{k-1})|} \tag{6.8}$$

Therefore, Equation (6.8) helps us in finding the proper ρ value between zero and one to have the best tracking efficiency. Simulation results in the following sections shows that choosing smaller ρ values close to zero achieve better tracking, as we limit large fluctuations in estimated computational speed $\widetilde{\omega}_{\mathrm{m}}$ at the specified voltage and frequency levels, by limiting the amount of contribution of measured speed ω to it, see Equation (6.4).

6.4.3 Digital Controller Algorithm

Equation (6.4) allows as tracking the system variations efficiently by selecting a suitable value for ρ. Furthermore, no information on the system parameters is required at all, that is very important for process variability since the voltages and the frequencies are not known and could vary. Thus, the controller just needs to measure the computational speed ω and get the C_i and N_i information from the operating system in order to estimate the value of the computational speeds $\widetilde{\omega}_{\mathrm{m}}$ at the two different frequencies defined at V_{low}, i.e. $\widetilde{\omega}_{\mathrm{max_l1}}$and $\widetilde{\omega}_{\mathrm{max\_l2}}$. Note that each $\widetilde{\omega}_{\mathrm{m}}$ is updated only as long as we stay longer at its corresponding frequency level, otherwise it remains unchanged. By multiplying $\widetilde{\omega}_{\mathrm{max_l1}}$and $\widetilde{\omega}_{\mathrm{max\_l2}}$ by the remaining time L_i respectively, we could know how many instructions can be executed at these two threshold speed values (i.e. $\Delta_1 = \widetilde{\omega}_{\mathrm{max\_l1}}(t_k) \cdot L_i(t_k)$ and $\Delta_2 = \widetilde{\omega}_{\mathrm{max\_2}}(t_k) \cdot L_i(t_k)$). Comparing the remaining number of instructions to be executed $\Delta = \delta(t_k) \cdot L_i(t_k) = C_i(t_k) - \Omega(t_k)$, from Equation (6.2) with Δ_1 and Δ_2, we could specify at which operating region we should be as shown previously by equation (6.3). Finally, if the remaining time $L_i(t_k)$ is less than T_s (i.e. the task has been completed and we are going to start a new one, we reset the value of Ω to zero and set $L_i(t_k)$ to be equal to the new task deadline N_i. Otherwise, the loop to update the value of the computational speeds $\widetilde{\omega}_{\mathrm{max\_l1}}$ and $\widetilde{\omega}_{\mathrm{max\_l2}}$, estimate Δ_1 and Δ_2, and compare them with the value of Δ is repeated. Figure (6.7) explains in details the control algorithm of the digital controller used in each voltage-frequency island GALS-NoC.

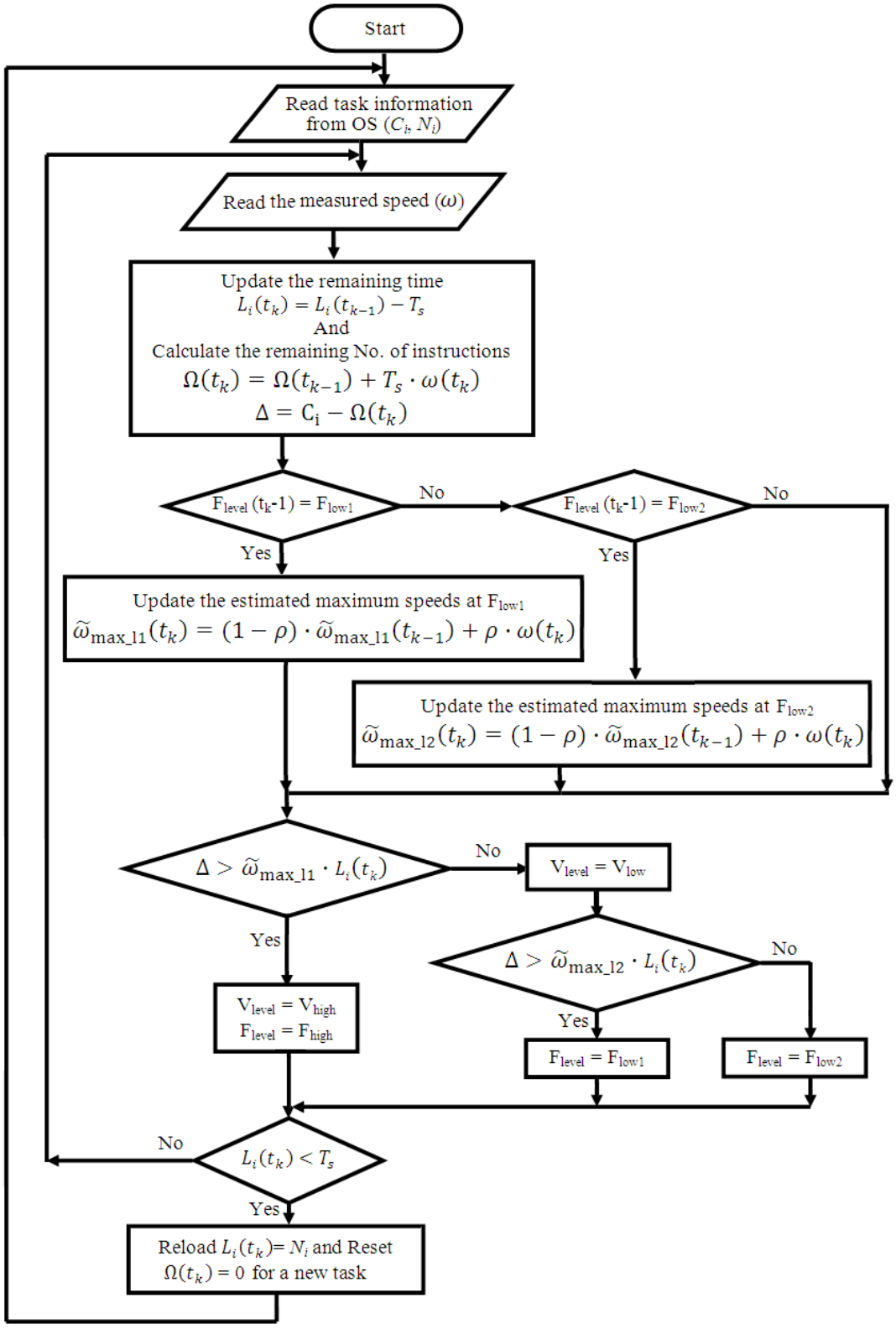


Figure 6.7: Flow Chart of the Digital Controller Algorithm.

6.5 Simulation Results

6.5.1 Load Modeling

In order to build and test a MATLAB behavioral model for a GALS-NoC processing node, we have used the definitions for the main sources of power dissipation shown in Chapter 5. A fictive load that behaves more like a real loaded processor has been built using MATLAB/Simulink [ZAK 10a]. It uses Equation (5.1), (5.2), and (5.3) in representing its output current waveform and power dissipation variations. Figure 6.8 shows the processing node behavior observed with such a simulation. It gives the current variation and the processed number of instructions of our fictive processor which depends on the applied voltage and frequency values. The effect of loading a new instruction into the processor on the current waveform is also shown by the added irregular ripples with each instruction. Also the sampling effect on the processor current waveform has been taken into consideration. Moreover, this model is enabled to change its computational speed with respect to the voltage/frequency pair applied to it.

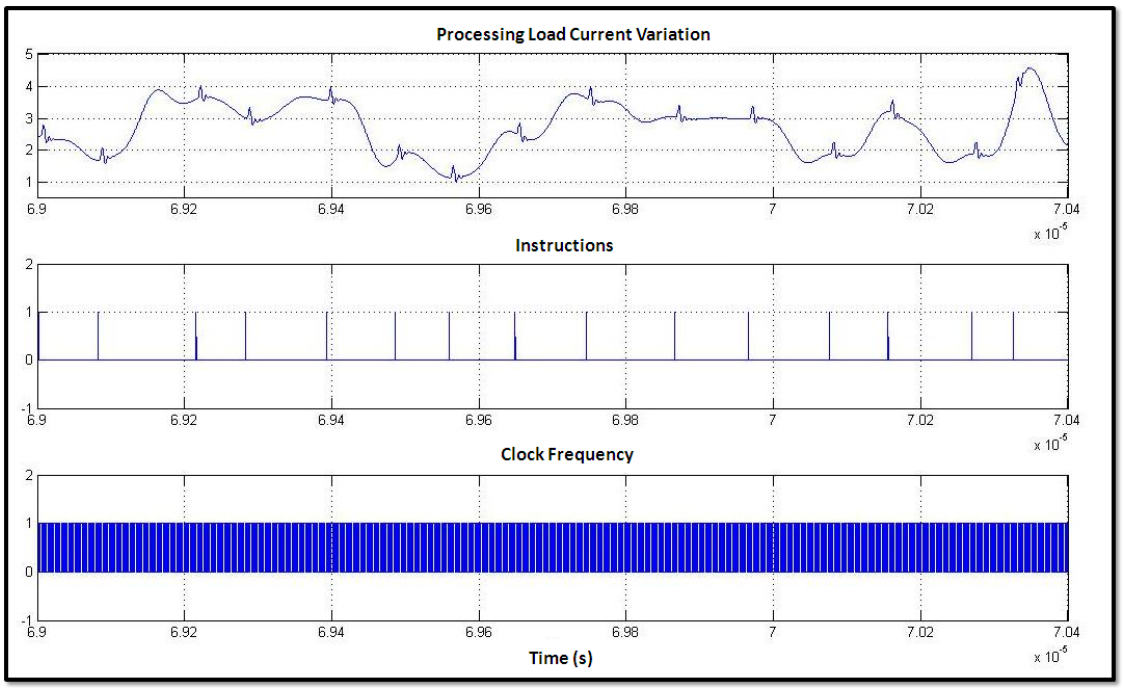


Figure 6.8: Simulink simulation for the processing node load model showing its current variations with respect to different processing instructions.

This model helps us to verify the proposed NoC control methodology by testing the effect of applying DVFS through the use of the digital controller in minimizing the average power and energy consumption while guarantying the system performance with parameter variations.

6.5.2 Workload Tracking Efficiency

This subsection presents some simulation results to evaluate the tracking efficiency of the proposed digital controller. A scenario with three tasks to execute is proposed: the first task starts with 4 instructions to do in 0.5µs, then a 65 instruction task has to be executed in 2.5µs and the last one has to compute 10 instructions in 1µs, these data are provided by the OS. Figure 6.9 shows the simulation result of the system with 2 voltage levels and 3 frequency levels. The top plot shows the average speed set point C_i/N_i of each task (for guideline), the predicted average speed δ (for guideline) and the measured computational speed ω, while the bottom plot shows the supply voltage V_{dd}.

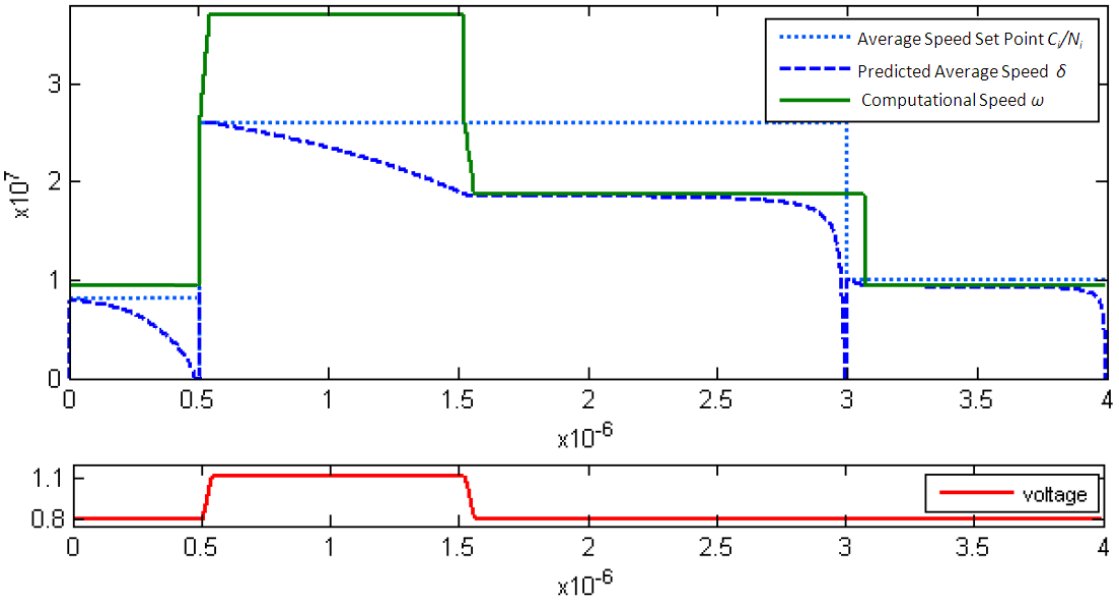


Figure 6.9: Matlab Simulation Result of the Digital Controller.

The energy consumption is calculated in order to have an idea of the reduction achieved. Thus, the Equation (5.1) is used and a ratio of this power consumption is added due to the V_{dd}-hopping principle: 20% more during the voltage transition times and 3% more during the steady state [DUR 09]. Finally, the integration during the whole running

time gives the total energy consumption. The control strategy is compared with a system without Dynamic Voltage and Frequency Scaling (DVFS) mechanism: the supply voltage is fixed to the most penalizing level, i.e. $V_{dd} = V_{high}$, and so is the clock frequency $f_{clk} = f_{high}$. Figure 6.9 shows that with the 2 possible voltage levels the system runs during more than 75% of the simulation time at the low voltage. As a result, a reduction of the energy consumption more than 20% is achieved (in comparison to a system without DVFS mechanism) with the 3-task test bench proposed [ZAK 10a]. Note that as far as we have frequency levels less than the two maximal possible clock frequencies (i.e. f_{high}, f_{low1}), this will contribute to the instant dynamic power saving but not to the energy consumption. More simulation results will be shown for the system workload tracking on MIPS-R2000 processor with the 45nm CMOS parameter variations on Chapter 7.

6.5.3 Robustness to Process Variability

As the proposed control strategy does not use any information on some system parameters, the controller adapts itself with these uncertainties. Figure 6.10 shows how the system is still working for different process variability effects.

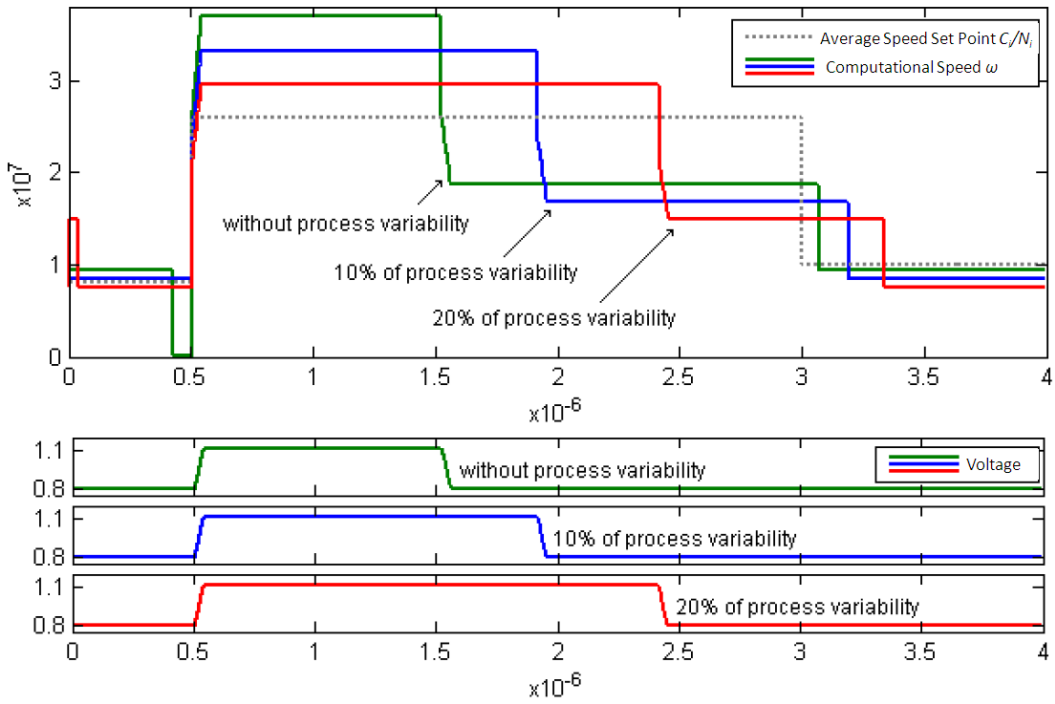


Figure 6.10: Matlab Simulation Results to Test the System Robustness with Different Degrees of the Process Variability Effect.

In Figure 6.10 different degrees for the process variability effect were tested, which correspond to “no variability”, “10% of variability” and “20% of variability”. One could see that the estimation of the maximum computational speed ω_m allows the system to still work, even if the processing node does not work as expected (i.e. with a computational speed ω lower than expected). Of course, in order to compensate a lower computational speed induced by the process variability, the system will run a longer time at the penalizing supply voltage [ZAK 10a]. The robustness is limited by the maximum possible activity of the processing node anyway. Indeed, if the chip is not enough fast to compute the task while running at the maximal speed (the chip runs with the highest voltage and highest frequency), the controller would not be able to do anything to solve this failure. The only way is to migrate the task allocated to this processing node to a high performance one, and this has to be done by the operating system, see Figure 6.2.

6.6 Conclusions

In this Chapter, we have addressed the design of GALS-based NoC communication architectures with multiple voltage-frequency domains. Firstly, we have introduced a new architecture based on the use of the asynchronous PSTR that adapts its maximal output clock frequency according to the performance evaluation of the processing domain. Evaluating the fabrication process quality and the local environmental parameters (voltage, temperature) is done with the help of activity monitors embedded in each processing domain. Secondly, we have presented a variation-adaptive feedback control methodology for NoC architectures with multiple voltage-clock domains. More precisely, we developed techniques to dynamically control the speed of each voltage-frequency island and provide robustness against the uncertainties and variations in the design parameters. At the same time, our techniques adapt the operating voltage and frequency to the variations in the workload to save power and energy consumption. Simulation results demonstrate robustness to parameter variations and more than 20% energy savings for a behavioral-like Matlab load model. An implementation of the proposed digital controller in the CMOS 45nm technology from STMicroelectronics will be shown in the following chapter. The system will be tested on MIPS-R2000 processor under different process variability corners.

Chapter 7

Designing Process Variability Robust DVFS Control

7.1 Introduction

In the previous chapter, it has been shown that integrating activity monitors in different clock domains in a GALS-NoC system is a promising solutions to reduce the process variability impact. A complete modeling methodology was introduced to build a process variability robust NoC-node with minimum energy consumption. The proposed feedback controller was proven to especially adapt more smartly voltages and frequencies (energy/performance) with strong process variability. The DFVS controller is based on a fast predictive control, which is shown to be more energy-efficient than the intuitive approach method.

In this chapter, we will study, analyze and design a complete circuit for different parts of the process variability robust energy-efficient DVFS control system. This DVFS control technique improves the performance, power consumption, and reliability of future NoC-based architectures in a synergistic manner. We consider NoC-based architectures consisting of multiple voltage-frequency islands. Each island may contain several processing cores that share a single clock, supply voltage, and digital control unit.

First, the MIPS R2000 microprocessor is synthesized on STMicroelectronics 45nm CMOS technology, in order to represent our processing load case of study. Then, the information extracted from the analysis of the MIPS R2000 processor on different process variability corners has been used in the programmability of our asynchronous PSTR. Finally, the digital control system is designed. The gain in the power/energy consumption reduction and the area overhead of the proposed control system is compared with a system without DVFS, and the one that uses the intuitive DVFS approach method.

7.2 Case Study: MIPS-R2000

MIPS-R2000 is a 32-bit reduced instruction set computer (RISC) initially developed by the Stanford University, where it stands for a Microprocessor without Interlocked Pipeline Stages. This means that it has single execution cycle instructions so that the compiler can schedule them to avoid conflicts. The MIPS architecture includes thirty-two general-purpose 32-bit registers and fifty-eight instructions, each 32 bits long. The instructions are processed in a five-stage pipeline: fetch, decode, execute, memory, and write back, see Figure 7.1. MIPS R2000 includes also a coprocessor to handle exceptions and hold configuration bits. Only a few configuration bits are used for the R2000 architecture, and are mostly used to enable/disable exceptions and to configure the caches. The MIPS architecture supports exception handling and interrupts. Due to the MIPS R2000 simplicity in terms of architecture, programming model, and instruction set as well as availability as an open core, it has been used as our main case of study in the DVFS control system design for a GALS-NoC, that compensate for the 45nm CMOS uncertainties. Details of the MIPS R2000 synthesis and analysis results are shown in the following section.

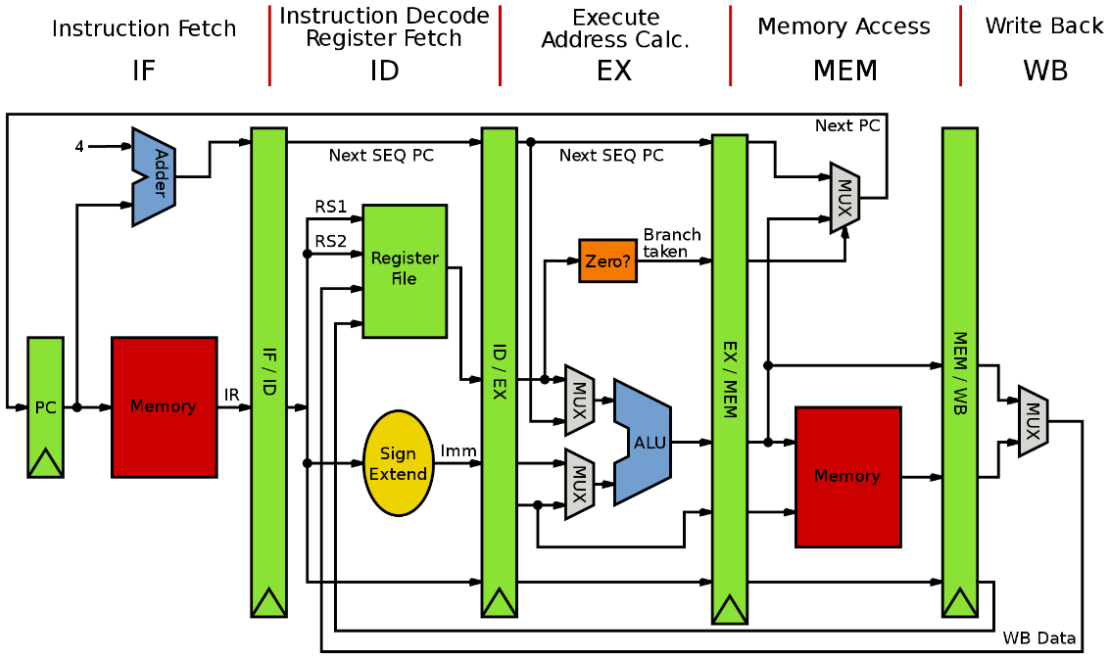


Figure 7.1: Internal Architecture of the Pipelined MIPS (5 Stages).

7.3 Analysis of MIPS R2000 Critical Path Delay Variations

Presently, the variability is captured in the design by using simulation corners, which correspond to the values of certain process parameters that deviate by a certain standard deviations from their typical value. In STMicroelectronics 45nm CMOS technology, three PVT (Process, Voltage, and Temperature) corners are available: Best, Nominal and Worst. All standard logic cells were characterized at each of these three corners. So, we use Design Vision tool to implement MIPS-R2000 using STMicroelectronics 45nm CMOS libraries in order to test its behavior at each of the previously specified PVT corners.

Since, our main goal is to define the optimum operating clock frequency needed by the processing load that compensates for the propagation delay variations due to the process variability impact. Therefore, the critical path delay of the synthesized MIPS R2000 with respect to supply voltage is analyzed at the three different PVT corners, as shown in Figure 7.2.

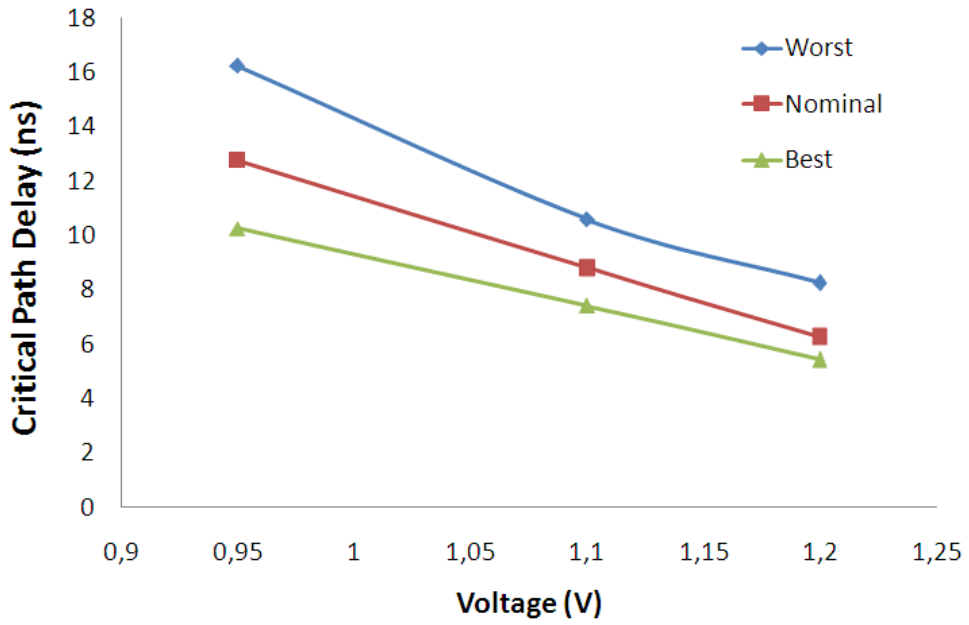


Figure 7.2: MIPS R2000 Critical Path Delay Variation with Respect to Supply Voltage at Three Different PVT Corners.

In Chapter 6, we have defined two main operating voltage levels outputs from V_{dd}-Hopping unit (V_{low} and V_{high}), where extra voltage levels will have a direct impact on the control system complexity and its relevant power consumption [DUR 09]. According to STMicroelectronics 45nm libraries, we choose V_{low} = 0.95 volts and V_{high} = 1.1 volts. Consequently, the optimum clock frequency needed by the MIPS R2000 at the specified two voltage levels with the three different process variability corners are defined as shown in Table 7.1:

Table 7.1: MIPS R2000 Optimum Clock Frequencies Required to Compensate for the Process Variability Impact on 45nm CMOS Technology.

Clock Frequency (MHz)		Process Variability Condition		
		Worst	Nominal	Best
Voltage Level	0.95	60	75	85
	1.1	95	115	145

Figure 7.3 shows how the MIPS R2000 processor is integrated as our main processing load into the previously proposed process variability robust DVFS architecture for a GALS-NoC system, depicted in Figure 6.4. Rom is loaded with a factorial program to test its execution by the processor. Each time a new instruction is loaded into *ram_data* bus, a pulse is generated on the *ram_ack* signal to indicate the availability of a new data for the processor.

Speed sensor is used to provide to the digital controller the computational speed information ω_i in terms of number of instructions executed per unit time. Therefore, the speed sensor contains an instruction counter which is clocked with the *ram_ack* signal, and a reference clock. Since, the period of the reference clock is crucial since it determines the accuracy of the calculated average speed. Moreover, it determines the system speed response. In fact the, speed sensor integrates also a register to memorize the instruction counter output (i.e. computational speed ω_i) on a predefined period and determines the average speed on each rising-edge of the reference clock (i.e. RST signal). If this period is short, the system will be fast but the calculated average speed will not be

accurate. On the opposite, a long period leads to a slow system but to a more accurate average speed. Therefore, according to the set of clock frequencies available for the MIPS R2000, see Table 7.1, the reference clock frequency was chosen to be 2MHz, in order to count a considerable amount of instructions with a proper system response. Note that, the RST signal is also used to reset the instruction counter every 500 ns to be ready for a new measurement of the computational speed ω_i. To conclude, the computational speed ω_i is now applied in terms of number of instructions executed per 500 ns to the digital controller.

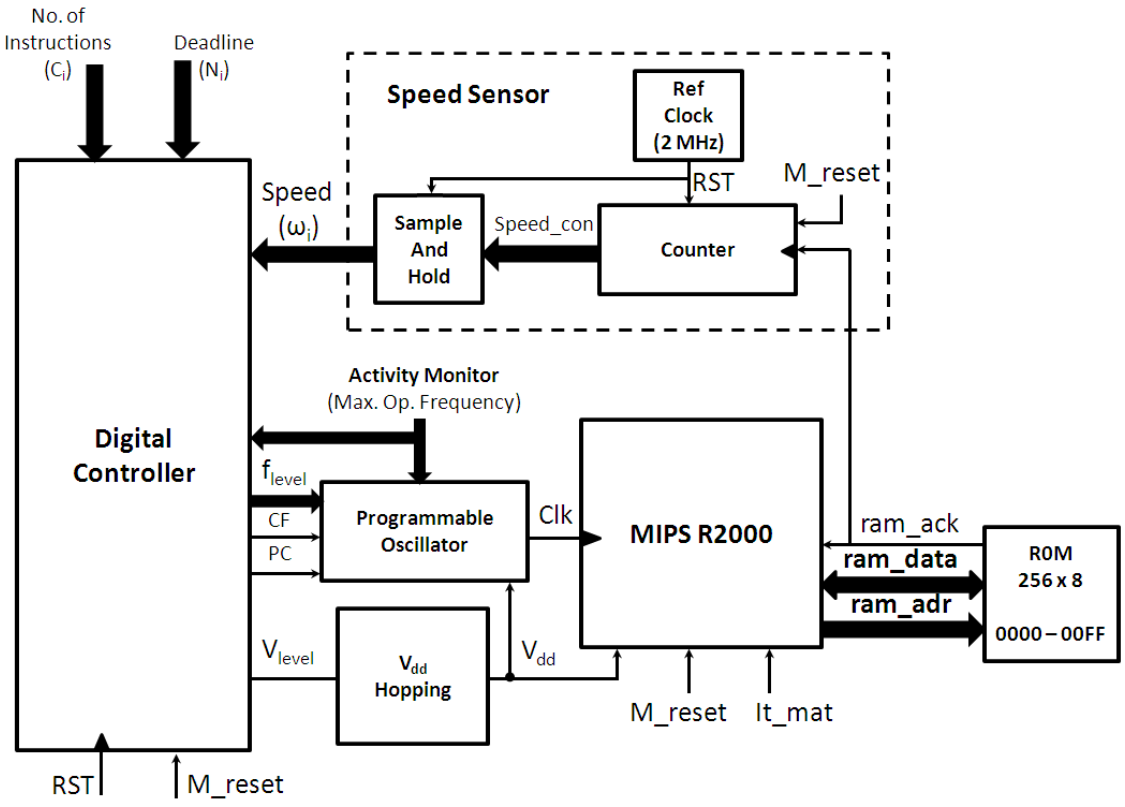


Figure 7.3: Process Variability Robust Energy/Performance Management Architecture (MIPS R2000 Case Study).

The digital controller predicts the corresponding set of speed set points based on the information sent by the OS (i.e. No. of instructions C_i and deadlines N_i), and the integrated process variability monitors (i.e. activity monitors). Afterward, it processes the error between computational speed ω_i and the predicted average speed point, to send the

proper voltage V_{level} and frequency F_{level} code values to the V_{dd}-Hopping and to the programmable oscillator respectively. *CF* and *PC* are the change frequency and pause clock pulses used by the programmable stoppable oscillator (PSO), refer to Chapter 3.

The activity monitor output is also used by the programmable oscillator to select the optimum set of clock frequencies suitable to be used with the present situation of process variability impact, further details is shown in the following section. Consequently, the system is able to locally adapt the output voltage and clock frequency values with respect to the actual process variability impact, in order to get the most suitable performance while limiting the system energy/dynamic power consumption.

7.4 PSTR Programmability to manage MIPS R2000 Variations

7.4.1 PSTR 45nm CMOS Delay Parameters

From Table 7.1 now it's clear that we have to design a programmable ring which has an operating frequency range from 60 to 145 MHz. Since, we are trying to build a correct behavioral VHDL model for the asynchronous PSTR ring, so the next step was to extract delay parameters of the PSTR ring from an analog tool. We used Cadence and the results are as shown in Table 7.2:

Table 7.2: Extracted Delay Values of the Asynchronous PSTR Functional Parameters on STMicroelectronics 45nm CMOS.

Asynchronous PSTR Parameter	Supply Voltage (Volts)	Propagation Delay (ps)
Asynchronous Ring Stage Forward Delay (D_{ff})	1.1	62
	0.95	79
Asynchronous Ring Stage Reverse Delay (D_{rr})	1.1	73
	0.95	92

7.4.2 PSTR Architecture

The requested target frequencies of the MIPS R2000 shown in Table 7.1 is in the range of MHz while the asynchronous ring normally generate outputs in the range of GHz. Therefore, either we will have to use an asynchronous PSTR with at least 50 stages, which means extra hardware, power consumption and area overhead. Otherwise, we have to add frequency dividers on the output of the asynchronous PSTR. According to the frequency values shown in Table 7.1, it was decided to use an asynchronous PSTR with a maximum of 20 stages and add 4 extra D-Flip Flops to divide the generated output frequency by 16, as it was previously depicted in Figure 3.14.

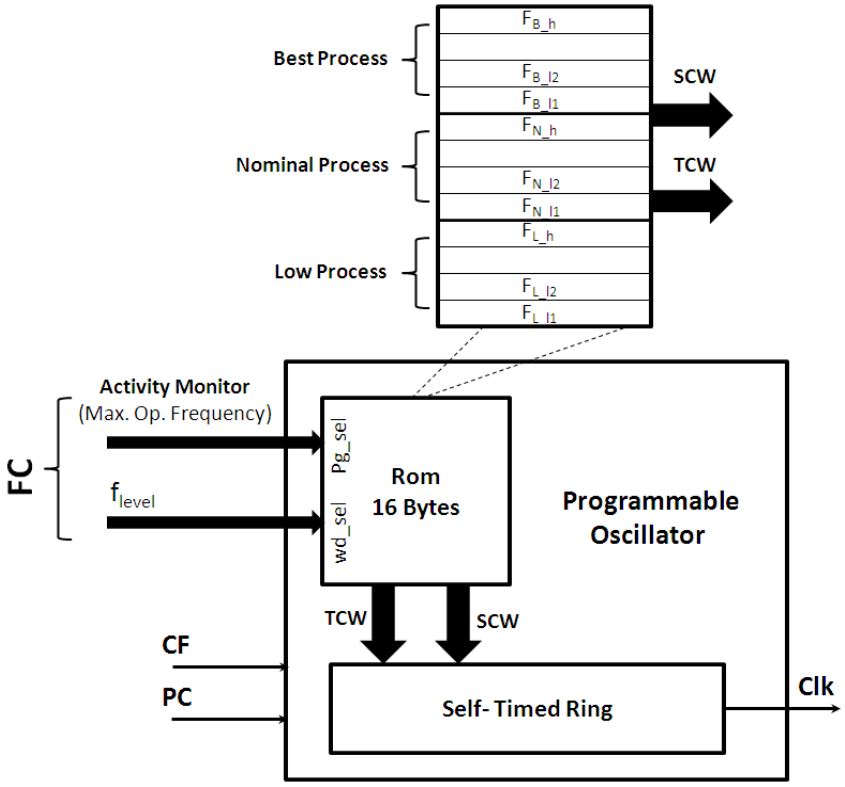


Figure 7.4: Memory Mapping of the Programmable Oscillator.

The main goal of our design is to adapt the generated clock frequency with respect to the current located process variability impact and to the processed workload. In the 45nm CMOS libraries provided by STMicroelectronics, we have three defined process variability corners. Consequently, we will split the contents of our programmable oscillator code memory (i.e. LUT1 and LUT2) shown in Figure 3.18 into three main

pages. Based upon the activity monitor output, the corresponding page will be selected. Each page contains a set of programming codes (i.e. TCW and SCW) that generates the suitable clock frequencies for the MIPS R2000 which compensate for the delay variation due to the process variability effect. In each programming code set, we have one code corresponding to the clock frequency F_{high} at the high voltage V_{high}, and two other codes corresponding to the clock frequencies F_{low1} and F_{low2} at the low voltage V_{low}. Consequently, our programmable oscillator explained in Chapter 3 can be simply represented by the block diagram shown in Figure 7.4.

7.4.3 PSTR Configuration

Table 7.1 defines the MIPS R2000 maximal clock frequencies (i.e. F_{high} and F_{low1}), which are needed to compensate for different process variability corners at the two specified voltage levels. For the third clock frequency F_{low2}, it was chosen to be lower than F_{low1} for each process variability corner, as shown in Table 7.3. Using the PSTR programmability design flow, shown in Figure 3.15, we get the PSTR configuration (i.e. the number of tokens, the number of bubbles, and the number of stages) for each of these frequencies (F_{high}, F_{low1} and F_{low2}) at the three different process variability corners. This helps us to specify the TCW and SCW contents of our programmable oscillator memory shown in Figure 7.4. The full PSTR programmability results are as shown in Table 7.3.

Table 7.3: PSTR Programmability to Manage Process Variations of the 45nm CMOS Technology on the MIPS R2000.

Process Variability	Voltage	Frequency	Target Frequency (MHz)	PSTR Programmability		
				No. Tokens	No. Bubbles	No. Stages
Best	1.1	F_{high}	145	12	6	18
	0.95	F_{low1}	85	12	4	16
		F_{low2}	60	14	3	17
Nominal	1.1	F_{high}	115	14	5	19
	0.95	F_{low1}	75	14	4	18
		F_{low2}	45	13	2	15
Worst	1.1	F_{high}	95	14	4	18
	0.95	F_{low1}	60	14	3	17
		F_{low2}	35	17	2	19

7.5 Digital Controller Design

In this section the complete design of the digital controller part in our proposed process variability robust DVFS architecture depicted in Figure 7.3 will be presented. Its main principle of operation is based on implementing the energy-efficient control algorithm, described by the flow chart shown in Figure 6.7.

7.5.1 Overall Architecture

In Figure 7.5 the details of the digital controller are depicted. For each task, the OS sends the total number of instructions to be executed C_i, and the task deadline N_i, where N_i is represented in terms of the requested number of clock cycles per reference clock output RST. Therefore, at the beginning of each task a counter is loaded with N_{i-1} and then it is regularly decremented on each positive edge of the RST signal to present the remaining time (i.e. number of reference clock cycles) L_i to complete the allocated task.

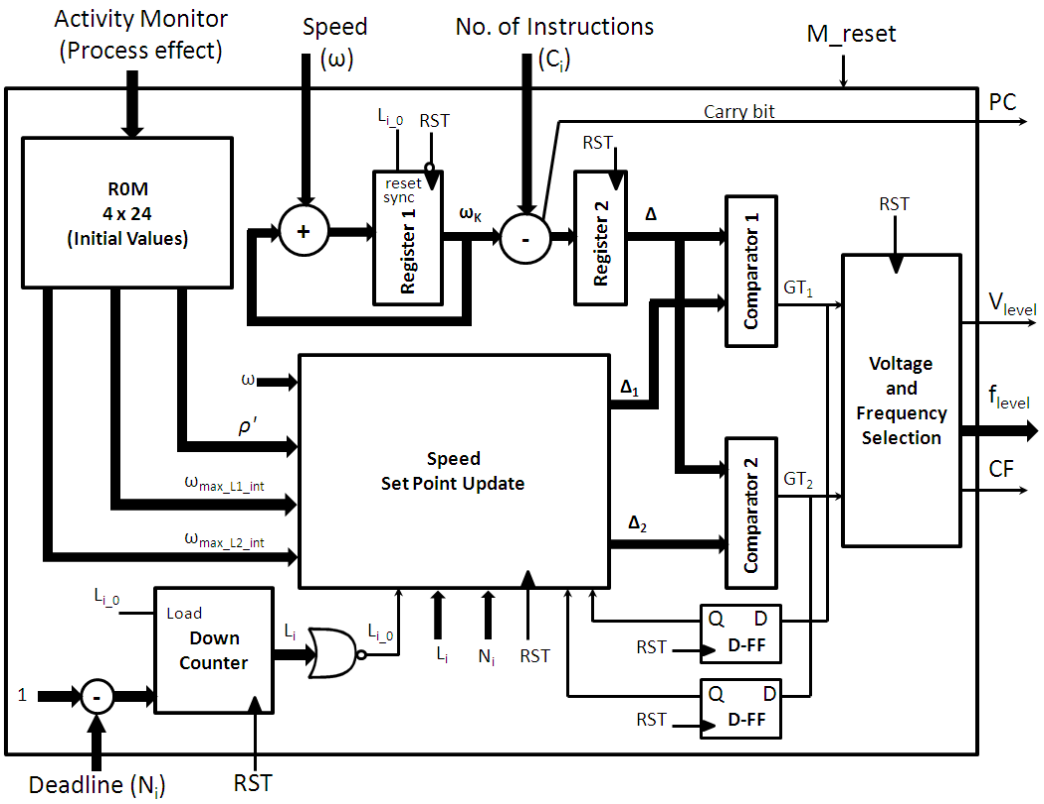


Figure 7.5: Digital Controller Architecture.

The computational speed information ω sent by the speed sensor represents the number of executed instructions per reference clock cycle. We need to know the total number of instructions executed by the system since the beginning of the task until the last sampling instant, i.e. ω_k. Therefore, an adder is used to accumulate ω as long as the task is running. This accumulated value is stored in Register 1 on each negative edge of the RST signal. Once the down counter output is zero (i.e. $L_{i\_0} = 1$), this means that we have one last RST clock cycle to complete the allocated task. As a result, Register 1 is reset (i.e. $\omega_k = 0$) on the next negative edge of the RST signal to be ready for a new task execution.

Providing that the task is still running, C_i is subtracted from ω_k to estimate the remaining number of task instructions to be executed i.e. Δ. If the result of the subtraction is negative, this means that our system has already completed the allocated task before its deadline. Therefore, a PC signal is sent to the programmable oscillator to stop the generated clock output till the assignment of a new task to the processor. This adds more energy saving opportunities by relaxing our system as it has no more tasks to execute. Otherwise, the digital controller will regulate the generated clock frequency with respect to the predicted workload.

Our energy-efficient digital controller is based on using an adaptive set of speed set points, rather than the concept of using a fixed speed set point as in the intuitive average control method, for more details refer to Chapter 5. Therefore, the digital controller contains a special unit called speed set point update. On the start up regulation phase of the system, this unit accepts the proper initial set of computational speeds set points (i.e. $\omega_{max\_L1\_init}$ and $\omega_{max\_L2\_init}$). $\omega_{max\_L1\_init}$ and $\omega_{max\_L2\_init}$ correspond to the maximum predicted computational speed of the system at F_{low1} and F_{low2} respectively with respect to the current situation of process variability impact. Subsequently, the speed set point update unit keeps adapting these two maximum predicted computational speeds set points with regard to the last sampled computational speed value ω and the update factor ρ'. Afterwards, the main functionality of this unit will be to estimate the maximum number of instructions that can be executed by the system if it was supplied by either F_{low1} or F_{low2} (i.e. Δ_1 or Δ_2 respectively).

By comparing Δ with Δ_1 and Δ_2 (i.e. GT1 and GT2), we can decide which couple of voltage and frequency levels will be sufficient to supply our processor, in order to meet the allocated task deadline. Here comes the main functionality of the voltage and frequency selection unit. The supply voltage and the clock frequency have to be controlled together in order to ensure the maximum delay over the critical path. Clearly, the voltage and frequency selection unit is required to increase the voltage before increasing the frequency and, respectively, to decrease the frequency before decreasing the voltage.

7.5.2 Speed Set Point Update

The architecture of the speed set point update unit is depicted in Figure 7.6. The main goal of this unit is to continually predict the maximum number of instruction Δ_1 and Δ_2 that can be executed by the system within the specified remaining time for the task L_i before its deadline. Actually, Δ_1 and Δ_2 are estimated to predict the system computation capabilities when it will be supplied by V_{low} with either F_{low1} or F_{low2}, respectively.

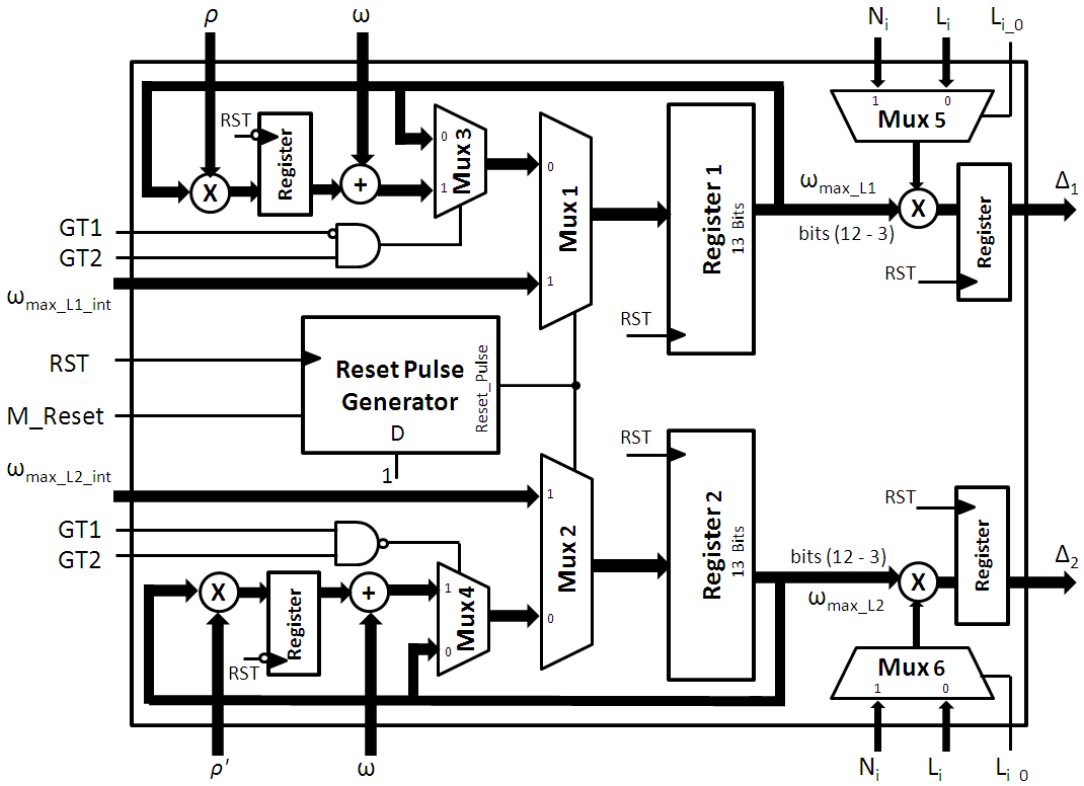


Figure 7.6: Speed Set Point Update Unit.

The speed set point update unit is composed of two symmetrical branches. Each of the two branches applies the formula defined by Equation (6.4) with $\rho = 1/8$ in order to get the two estimated computational speeds ω_{max_L1} and ω_{max_L2}. To facilitate the implementation of this part we first get ω_{max_L1}* 8 and ω_{max_L2}* 8, using $\rho' = 7$, from MUX1 and MUX2 outputs respectively. Then, MUX1 and MUX2 outputs are simply logically shifted 3 bits to the right to divide their outputs by 8, so we get ω_{max_L1} and ω_{max_L2} from Register 1 and Register 2 outputs respectively. Subsequently, each of the two outputs ω_{max_L1} and ω_{max_L2} are multiplied by L_i. As a result, we obtain the corresponding maximum number of instructions Δ_1 and Δ_2, that can be executed by the system during the remaining number of reference clock cycles L_i for the allocated task, wither the system was working with the predicted computational speed set point ω_{max_L1} or ω_{max_L2}. Note that, on the last sampling instant of the current allocated task ($L_{i\_0} = 1$), MUX5 and MUX6 select N_i of the next task to be executed. So that, on the next sampling instant (i.e. next positive edge of the RST signal), $\Delta_1 = \omega_{max\_L1} * N_i$ and $\Delta_2 = \omega_{max\_L2} * N_i$. Afterwards, as long as the task is running MUX5 and MUX6 select L_i, so that, $\Delta_1 = \omega_{max\_L1} * L_i$ and $\Delta_2 = \omega_{max\_L2} * L_i$.

As on the start up regulation phase of the system, the speed setup point update unit has to accept the proper initial set of computational speed set points (i.e. $\omega_{max\_L1\_init}$ and $\omega_{max\_L2\_init}$) with respect to the current situation of process variability impact. Therefore, the reset pulse generator unit is included. The reset pulse generator unit generates a single pulse of duration equal to the RST period, see Figure 7.7. This enables MUX1 and MUX2 to select $\omega_{max\_L1\_init}$ and $\omega_{max\_L2\_init}$ on the start up regulation phase. Subsequently, the speed set point update unit keeps fine tuning these two maximal predicted computational speeds as ω_{max_L1} and ω_{max_L2}. Note that, ω_{max_L1} or ω_{max_L2} are updated with the corresponding measured value of ω only if the system is working with F_{low1} (i.e. GT1 = 0 and GT2 = 1) or F_{low2} (i.e. GT1 = 0 and GT2 = 0) respectively. Otherwise ω_{max_L1} and ω_{max_L2} remains unchanged, as the measured value of ω in that case corresponds to a processing system under a clock frequency of F_{high}. Therefore MUX3 and MUX4 are used to enable this kind of adaption for ω_{max_L1} and ω_{max_L2}. Different timing diagrams for the system behaviour on the three different process variability corners are shown in Section 7.6 to better explain the flow of all signals.

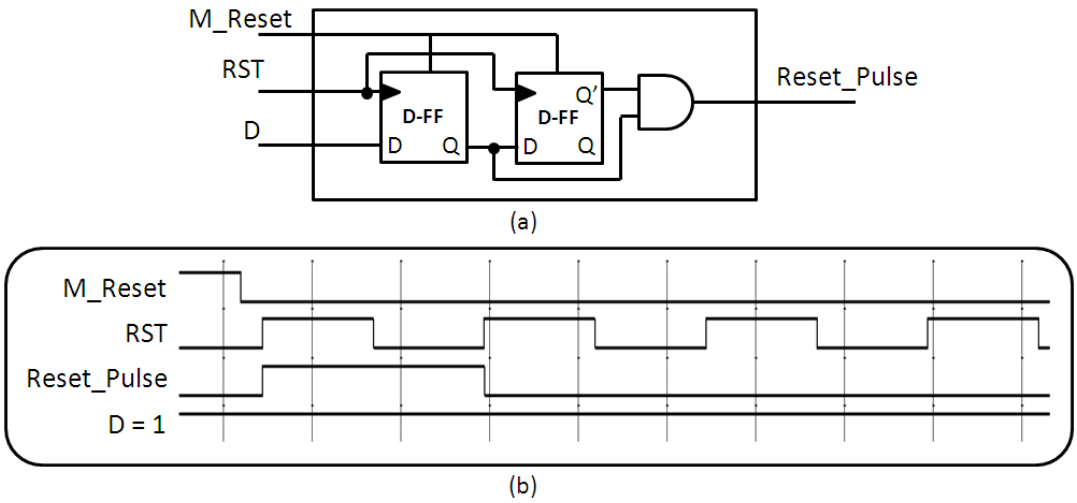


Figure 7.7: Reset Pulse Generator (a) Architecture (b) Timing Diagram.

7.5.3 Voltage and Frequency Selection

Switching performance levels take time for both voltage regulators and clock generators. Switching voltage levels is particular slow and switching frequencies is orders of magnitude faster than voltage level switching. As a result, we have to increase the voltage first then the clock frequency, and to decrease the voltage after clock frequency is lowered. The voltage and frequency selection unit shown in Figure 7.8 is able to do so through the use of Delay1 and Delay2. Delay2 corresponds to the V_{dd}-Hopping delay to switch from one voltage state to another one, which is approximately equal to 300 ns [MIE 07]. While Delay1 corresponds to the programmable oscillator delay to switch from one frequency to another, which is approximately equal to 20 ns, see Chapter 3.

Initially DFF1 and DFF2 outputs are reset to zero. There are two possible scenarios for voltage switching. First, when the system has to switch from V_{low} to V_{high} (i.e. GT1 value changes from 0 to 1). In this case, MUX3 output will be directly connected to the input port 1 of MUX1. This insures that V_{level} immediately changes its value from 0 to 1. On the other hand, this positive edge transition on MUX1 output will change XOR output to 1. As a result, MUX2 will select input port 1 to be connected to f_{level}. This delays the corresponding frequency code with an amount of time equal to Delay2, see Figure 7.9. This insures that the programmable oscillator will not change its output clock frequency before V_{dd}-Hopping output is stable.

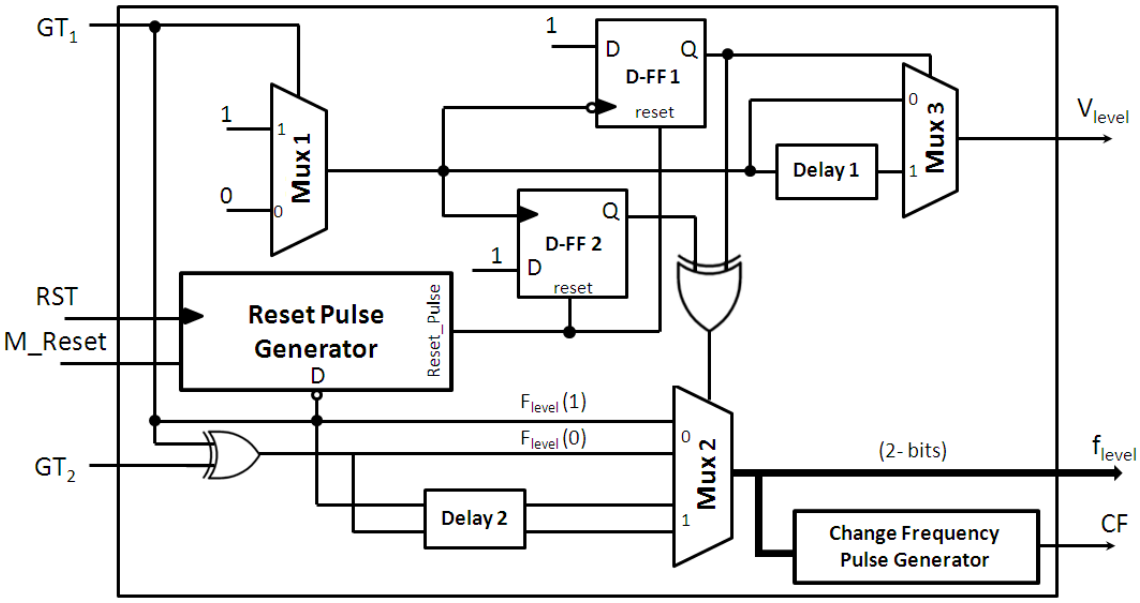


Figure 7.8: Voltage and Frequency Selection Unit.

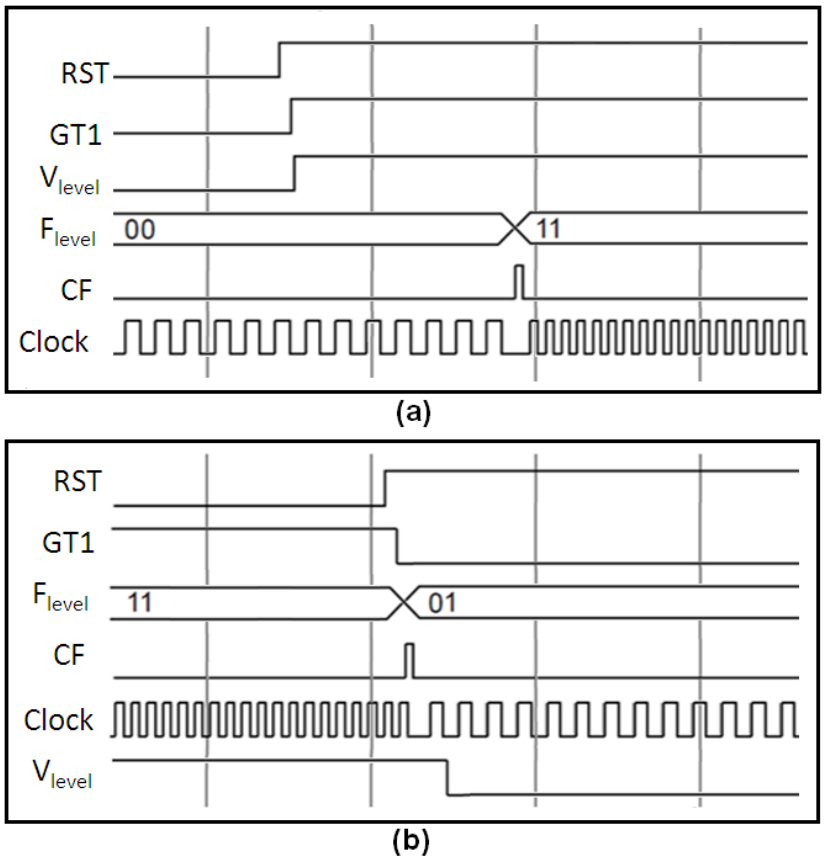


Figure 7.9: Timing Diagram of the voltage and frequency selection unit.
(a) Switching from V_{low} to $V_{high.}$ (b) Switching from V_{high} to $V_{low.}$

Second scenario, when the system has to switch from V_{high} to V_{low} (i.e. GT1 value changes from 1 to 0). In this case, DFF1 will be triggered and its output will be equal to 1. Consequently, the XOR output is equal to zero, which enables the immediate connection of f_{level} to the new frequency code value as MUX2 selects the incoming data on its input port 0. MUX3 will select the incoming data on its input port 1, which is delayed with an amount of time equal to Delay1. This insures that the V_{dd}-Hopping unit will not change its output before the programmable oscillator new clock frequency is stable, see Figure 7.9.

As our programmable oscillator (Asynchronous PSTR) proposed in chapter 3 needs a change frequency CF pulse to indicate the presence of a new frequency code. The change frequency pulse generator depicted in Figure 7.10 is used. As it is simply en event detector, it is mainly composed of XOR gates. The reset pulse generator shown in Figure 7.7 is used again in the voltage and frequency selection unit with its D input inversely connected to GT1. This will reset DFF1 and DFF2 once GT1 changes its value from 1 to 0. As a result, the voltage and frequency selection unit is allowed to adapt again the switching delayed instants between V_{level} and F_{level}.

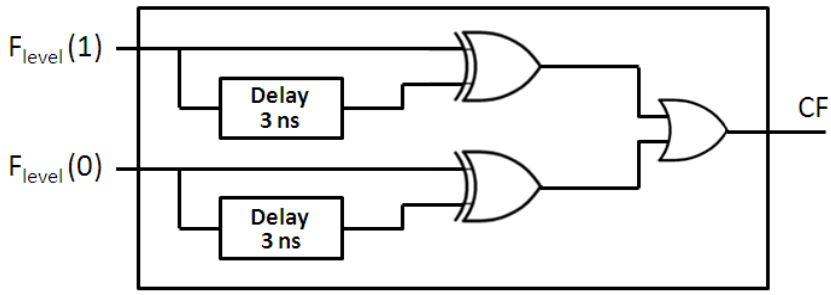


Figure 7.10: Change Frequency Pulse Generator.

7.6 Simulation Results

The design presented in Sections 7.5 is implemented using the STMicroelectronics 45nm CMOS standard libraries for the physical implementation with the Synopsys Design Vision tool. In order to evaluate the tracking efficiency of the proposed digital controller under different process variability conditions, a post layout simulation with Modelsim has been used. A scenario with three tasks is proposed: the first task starts with C_1 = 100 instructions to do in 2 µs, (i.e. N_1= 4 as T_{RST} = 500 ns), then

a $C_2 = 340$ instructions task has to be executed in 4 µs (i.e. $N_2 = 8$), and the last one has to compute $C_3 = 160$ instructions in 3 µs (i.e. $N_3 = 6$). These data are supposed to be provided by the OS.

Figure 7.11 shows the simulation result of the system under nominal process variability (i.e. activator monitor output A_Mon = 01). We can see that the used set of clock frequencies are F_{high} = 115 MHz, F_{low1} = 75 MHz and F_{low2} = 45 MHz, which corresponds to the proper set of clock frequencies that has to be used under nominal process variability as it was shown in Table 7.3. Task 1 was completed successfully with both V_{level} and F_{level} equal to 0 (i.e. $F_{clk} = F_{low2}$ = 45 MHz and $V_{level} = V_{low}$ = 0.95 volts), all over the allocated time for the task. This is because Δ during task 1 execution is always less than Δ_1 and Δ_2. Note that Δ is firstly loaded with C_1. Afterwards, it is decremented with ω_k on each positive edge of the RST signal. Moreover, ω_k is reset to zero on the RST negative edge just before the start of task 2. Each time a new instruction is executed we have a pulse on the Ins_Speed signal, where it's connected to ram_ack signal shown in Figure 7.3. Ins_Speed is added in Figure 7.11 as a guideline for the MIPS R2000 clock speed.

For task 2, Δ is firstly loaded with $C_2 = 340$, which is higher than Δ_1 and Δ_2. As a result, the digital controller directly sets $V_{level} = V_{high}$, and then after 300 ns sets $F_{level} = F_{high}$. Therefore, the digital controller speeds up the MIPS R2000 to be able to complete the task at the proposed deadline N_2. This is by setting the MIPS R2000 to work with the maximum supply voltage of 1.1 volts and clock frequency of 115 MHz. Once the digital controller has detected that task 2 can be completed with relaxed conditions, for example at $L_i = 4$, Δ is less than Δ_1, the system switches back to V_{low}. Here, we can see that the digital controller firstly switches F_{level} to F_{low1} then after 20ns V_{level} to V_{low}. Therefore after running the MIPS R2000 for 1.52µs at 1.1 volts, it is now supplied with 0.95 volts and 75 MHz. Again the processor continues to work with these operating conditions tell Δ is less than Δ_2 at $L_i = 1$. At this moment the system switches back again to the second lower clock frequency F_{low2} = 45 MHz at V_{low}. We can see that the estimated computational speed ω_{max_L1} was updated with the measured speed ω of the MIPS R2000 only when the system was supplied with F_{low1}.

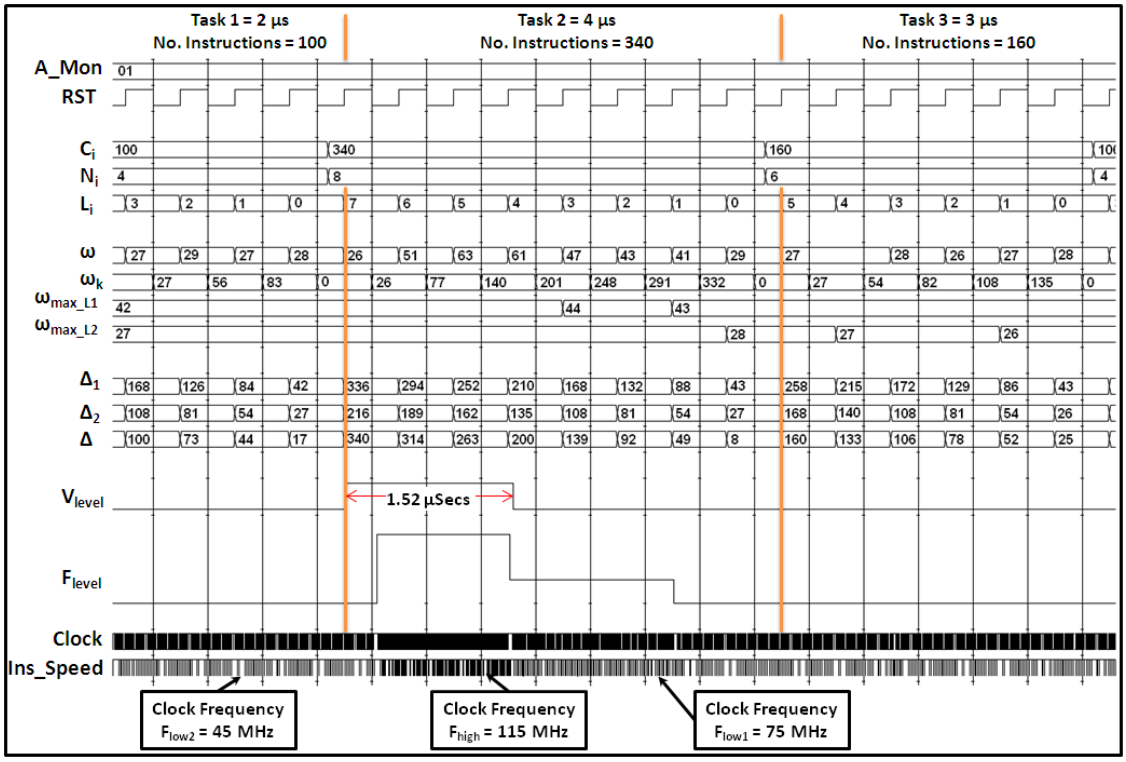


Figure 7.11: Timing Diagram of the Digital Controller Behaviour with 3 Different MIPS R2000 Workloads under Nominal Process Variability Effect.

For task 3, Δ is firstly loaded with C_3 = 160, which is lower than Δ_1 and Δ_2. As a result, the digital controller keeps supplying the MIPS R2000 with V_{low} and F_{low2} as long as Δ is lower than Δ_1 and Δ_2 all over task 3 execution time till its deadline. Note also that in this case ω_{max_L2} was updated with the measured speed ω of the MIPS R2000 as the system was supplied with F_{low2} at this time.

In Figure 7.12, the simulation result of the system under worst process variability (i.e. activator monitor output A_Mon = 00) is depicted. We can see that the used set of clock frequencies are now F_{high} = 95 MHz, F_{low1} = 60 MHz and F_{low2} = 35 MHz, as it was shown in Table 7.3. Therefore, our programmable oscillator now generates the proper set of clock frequencies that has to be used in order to compensate for the reduced performance of the MIPS R2000 (i.e. increased critical path delay), under worst process variability effect. As a result, Task 1 runs 1.5µs at F_{low1} then it switches to F_{low2} for the last 0.5 µs. Moreover, Task 2 runs for 3.02 µs at $V_{high,}$ which is 2 times longer than when

the MIPS R2000 is under nominal process viability, Figure 7.11. Then, when $L_i = 1$, it switches to $F_{low1} = 60$ MHz. Finally, it was able to complete the task successfully, where at $L_i = 0$ the estimated remaining number of instructions to be executed is only 7. Therefore, it was able to relax the processing power again by switching to $F_{low2} = 35$ MHz. For task 3, Δ is firstly loaded with $C_3 = 160$, which is now lower than Δ_1 and higher than Δ_2. As a result, the digital controller supplies the MIPS R2000 with V_{low} and F_{low1}. Once, Δ becomes lower than Δ_2 at $L_i = 1$ the digital controller relaxes the frequency to F_{low2} till task 3 deadline.

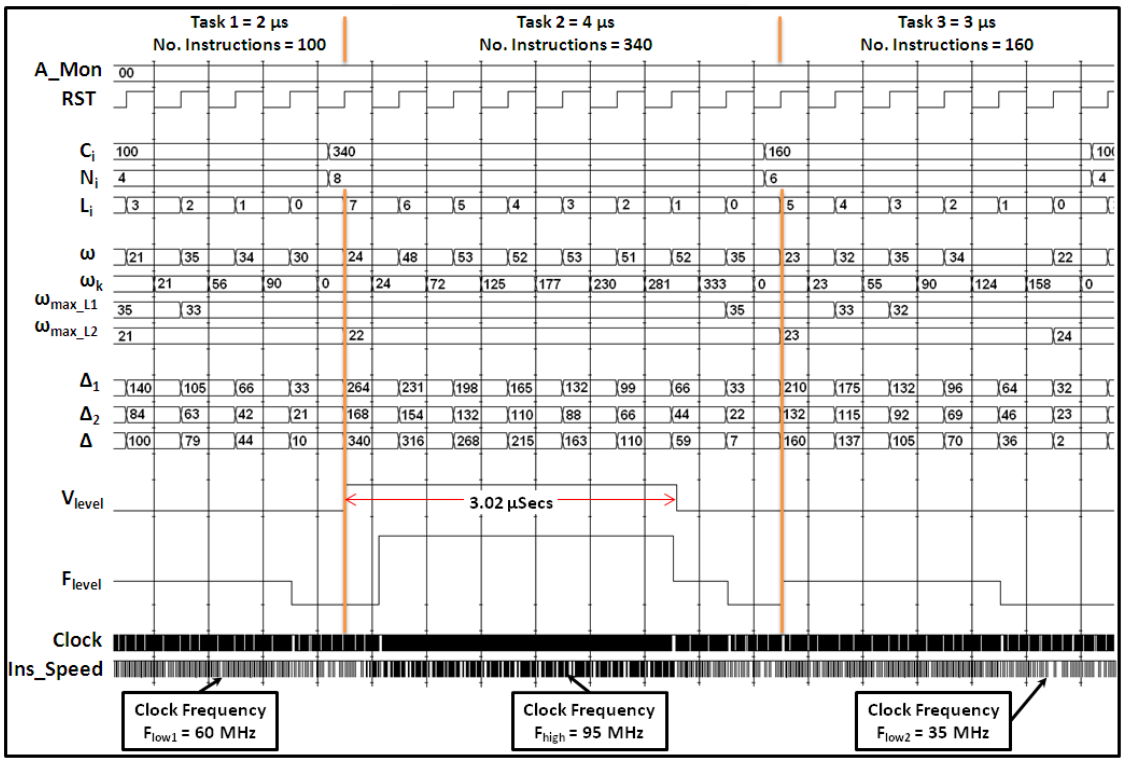


Figure 7.12: Timing Diagram of the Digital Controller Behaviour with 3 Different MIPS R2000 Workloads under Worst Process Variability Effect.

The same 3 tasks workload is simulated again for the MIPS R2000 under best process variability see Figure 7.13. Now, the used set of clock frequencies generated by the programmable oscillator are $F_{high} = 145$ MHz, $F_{low1} = 85$ MHz and $F_{low2} = 60$ MHz. These clock frequencies correspond to the proper set that has to be used under best process variability as it was shown in Table 7.3. Using these frequency configurations the MIPS R2000 is able to successfully complete all the three tasks at $V_{low,}$ which adds more

power/energy saving opportunities than under the nominal case. Therefore, our proposed DVFS control architecture is able to not only compensate for the delay variations with different process variability impacts, but also exploit the enhanced response of the system under best variability conditions to gain more in terms of energy savings. Detailed comparison results with numbers are shown in Table 7.4. Note that for task 3 the MIPS R2000 completes the task 0.75µs before its deadline. At $L_i = 1$, $\Delta = 21$ which is even less than $\Delta_2/2$. This means that the processor needs only one half RST clock cycle to complete task 3 during $L_i = 1$. Therefore, on the next negative RST clock edge ω_k is shown to be higher than C_i. Consequently, the output of subtraction shown in Figure 7.5 is negative, which is next sampled as $\Delta = -13$. Thus, a PC signal is sent to stop generated clock output by the programmable oscillator, and keeps the processor in an idle mode till it has a new assigned task. Again, this exploits the enhanced system performance by adding more energy savings as the processor has no more tasks.

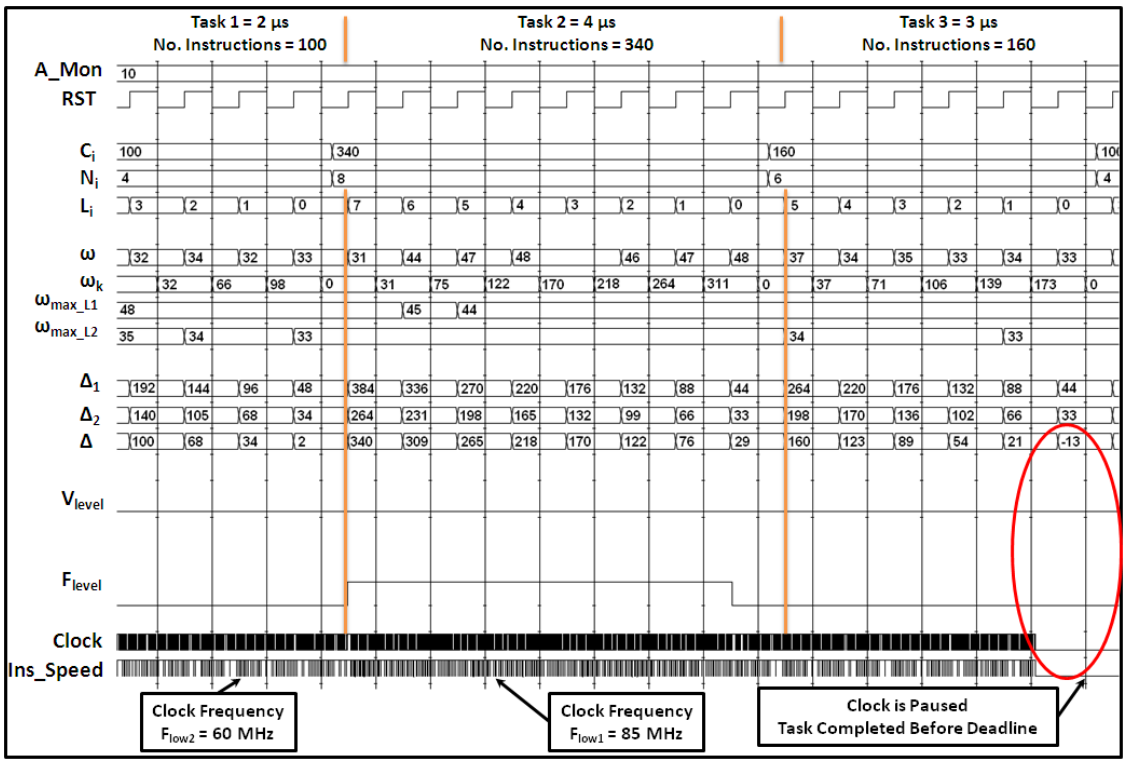


Figure 7.13: Timing Diagram of the Digital Controller Behaviour with 3 Different MIPS R2000 Workloads under Best Process Variability Effect.

To evaluate our proposed energy-efficient DVFS control for GALS-NoC architecture, the implemented chip is characterised for its average dynamic power, energy consumption, area overhead and robustness to process variability. Table 7.4 shows a comparison between the proposed energy-efficient DVFS control and the average based one, see Chapter 6, with respect to a system without DVFS. This comparison is in terms of average dynamic power and energy savings of the processor alone under different process variability corners.

Table 7.4: A Comparison between Energy-Efficient and Normal Average Based DVFS Control with Respect to a System without DVFS at Different Process Variability Corners.

<table>
<tr><th rowspan="2">Process Variability Impact</th><th colspan="2">Energy-Efficient Control
(Adaptive Speed Set Point within the Task, Figure 6.6.b)</th><th colspan="2">Average Based Control
(Fixed Speed Set Point all over the Task, Figure 6.6.a)</th></tr>
<tr><th>Average Dynamic Power Savings</th><th>Energy Savings</th><th>Average Dynamic Power Savings</th><th>Energy Savings</th></tr>
<tr><td>Best</td><td>51, 39 %</td><td>25,41 %</td><td rowspan="2">36,78 %</td><td rowspan="2">14,12 %</td></tr>
<tr><td>Nominal</td><td>51,42 %</td><td>21,18 %</td></tr>
<tr><td>Worst</td><td colspan="2">Achieve the Requested Performance with a Reduced Set of Clock Frequencies (Yield Enhancement)</td><td colspan="2">Used Set of Nominal Clock Frequencies Violates the MIPS R2000 Critical Path (Erroneous Data Outputs)</td></tr>
</table>

From table 7.4 it is clear that under nominal process variability, the DVFS control with an adaptive computational speed set point during task execution (i.e. energy-efficient control) is 1.5 more power and energy savings efficient than the one with fixed speed set point (i.e. average based control). Energy-efficient DVFS control is able to save 21.18% of the energy consumption and 51.42% of the average dynamic power consumed by a system without DVFS. Note that power savings results are approximately 2 times better than energy savings, as power savings are proportional to both voltage and frequency control, while energy savings are proportional only to voltage control.

Our proposed energy-efficient DVFS control has the ability to adapt the computational speed set points (i.e. set of clock frequencies generated by the PSTR) with respect to the process variability impact. Since under the previously the allocated 3 tasks workload, the proposed DVFS control architecture was able to reduce the high voltage running time to zero, see Figure 7.12. Therefore the energy-efficient DVFS control was able to exploit this enhanced performance of the system (i.e. reduced critical path delay) to save more energy consumption (i.e. 25.41% under best process variability impact). However, the average based DVFS control saves the same amount of energy regardless of the reduced process variability impact.

Under worst process variability condition, the used set of clock frequencies for a system without DVFS (i.e. 115 MHz) and even that for a system with average based DVFS control (i.e. 115 MHz at V_{high} and 75 MHz at V_{low}) violates the MIPS R2000 critical path delay, shown in Figure 7.2. As a result, the MIPS R2000 will have erroneous output results. Therefore this GALS-NoC processing node has to be neglected and its allocated tasks have to be distributed over other high performance processing nodes. However, with the proposed DVFS control architecture, the MIPS R2000 was still able to complete the allocated tasks successfully by using the proper set of maximum clock frequencies (i.e. 95 MHz at V_{high} and 60 MHz at V_{low}). This drastically relaxes the fabrication constraints and helps the yield enhancement.

In order to perfectly evaluate our system, the average dynamic power, the energy consumption and the area of different DVFS control architecture parts shown in Figure 7.3 has to be considered. From Table 7.5, It is clear that the two most power/energy consuming parts are the V_{dd}-Hopping and the programmable Oscillator. However, the digital controller consumes only 64.32 µW, as it has the lowest activity rate in the system of 2 MHz. Based on the values shown in Table 7.5, the average dynamic power savings, energy savings and area overhead of the whole DVFS control system are evaluated as shown in Table 7.6. These values are given for a GALS-NoC voltage-frequency island with a single processing element (i.e. MIPS R2000) and compared with another one with 8 processing elements, refer to Figure 6.1. Under nominal process variability the average dynamic power and energy savings values of the whole DVFS control system are smaller than but not too far from those presented in Table 7.4.

Table 7.5: Power, Energy and Cross Sectional Area of Each Part in the DVFS Architecture shown in Figure 7.3.

	MIPS R2000	Digital Controller	V_{dd} Hopping, [MIE 07]	PSTR Oscillator	Speed Sensor
Power Consumption (µW) at 1.1V - 25°C	15177,9	64,32	455,37	357	2,57
Energy Consumption (nJ) Over 9µSecs	108,4	0,589	4.098	3,213	,002
Area (μm^2)	20210	1858	4400	418	36

Since we have a single DVFS control system for each voltage-frequency island in a GALS-NoC system. Therefore, in a voltage-frequency island with multi processing elements, the efficacy of the DVFS control system will be more effective in saving power and energy consumption, see Table 7.6. Moreover, the area overhead of the extra DVFS hardware will be approximately divided by the number of processing elements per a GALS-NoC voltage-frequency island. For example the area overhead in a processing island with 8 processors is 4.15%. However, the area overhead in a processing island with a single processor is 33.21%.

Table 7.6: Performance Analysis of the Whole Energy-Efficient GALS-NoC DVFS Control System under Nominal Process Variability Impact.

No. of Processing elements per GALS-NoC Voltage-Frequency Island	Average Dynamic Power Savings	Energy Savings Over 9µSecs	Area Overhead
1	45,62 %	14,86 %	33.21 %
8	50,7 %	19,92 %	4.15 %

7.7 Conclusions

This chapter addressed the problem of designing the process variability robust DVFS control architecture proposed in Chapter 6. First, the MIPS R2000 microprocessor is synthesized using the STMicroelectronics 45nm CMOS libraries, in order to present our processing load case of study. Then, the information extracted from the analysis of the MIPS R2000 processor on different process variability corners has been used in the programmability of our programmable oscillator (i.e. asynchronous PSTR). Finally, the energy-efficient DVFS digital control system is fully designed and implemented on the STMicroelectronics 45nm CMOS technology. The digital controller was able to smartly adapt the voltage/frequency couple with workload variation and, at the same time, copes with the parameter variations which cannot be predicted or accurately modeled at design time. Moreover, it also considers the voltage regulator and frequency oscillator switching performance levels, in order to correctly adapt their switching instants. According to the measured degree of process variability effect on the processing load performance, the programmable oscillator was able to generate the proper set of maximum clock frequencies that does not violates the MIPS R2000 critical path delay. Afterwards, the digital controller regularly adapts the estimated computational speed set points based on the workload variations with the existing process variability impact.

For the different process variability corners defined by STMicroelectronics 45nm CMOS libraries, the proposed DVFS control system was able to successfully achieve the requested performance by the OS. This allows us to not neglect most of the voltage-frequency islands, even if they were under worst process variability effect, which has a direct impact on enhancing the fabricated yield. Moreover, it was able to approximately save 50% of the power and 25% of the energy consumed by a system without DVFS, which is even 1.5 times better than a system with intuitive average based DVFS control. In addition to that, the digital controller exploits the enhanced performance of the system under best process variability to save more power/energy consumption. One more advantage of the proposed DVFS architecture is its small area overhead, especially with GALS-NoC voltage-frequency islands, that contains more than one processing element. With 8 MIPS R2000 per voltage-frequency island, the area overhead will only be 4.15%.

Chapter 8

Conclusion and Perspectives

Regarding the recent and upcoming nanometric CMOS technologies, designing a complex system is now becoming very challenging. These challenges are due to several constraints, such as guarantying the correct system behavior with strong process variations or controlling the system speed and energy. The main problem with such systems is that their modeling is complex and their behavior is difficult to predict. Moreover, the fabrication yield is more difficult to maintain as a high rate in the nanometric technologies and it can be improved by appropriate design techniques (Design for Yield).

Indeed, process variations play an increasingly important role in system power consumption and performance as the technology scales down. This implies to consider global management strategies in order to respect energetic and real-time constraints. More precisely, there is a need for efficient algorithms and built-in circuitry able to adapt the system behavior to workload variations and, at the same time, cope with the parameter variations which cannot be predicted or accurately modeled at design time. Therefore performance estimation and management are today key points in new integrated systems. Concretely, these ideas were considered in a French national project called ARAVIS, sponsored by the global competitive cluster Minalogic.

New strategies for energy management have to be used to meet the energy and real-time constraints. Solutions such as dynamic voltage and frequency scaling (DVFS) have to be considered, they have been explored and have shown significant energy savings while meeting performance requirements. While this problem has been addressed before by a large number of engineers and researchers, no attention has been given to add the feature of controlling the impact of manufacturing process variations to the capabilities of the DVFS controllers.

In this thesis, we have first addressed the problem of designing a process variability robust source for generating adjustable clocks, in order to be applied within each voltage-frequency island in a complex GALS-NoC system. Self-Timed Ring (STR) is chosen as the core of the oscillator because of its reported advantages with respect to many points of view (programmability, accuracy, robustness against process variability). A new methodology for calculating the oscillation frequency of STRs is proposed. The method shows very high efficiency and accuracy during the different design phases. By including Charlie/drafting effects to our VHDL models, we showed the possibility of using digital simulation to accurately model and simulate the STR behavior.

Programmability has been introduced to STRs using two main different strategies. The two proposed strategies are simple architecture based solutions. These strategies show high efficiency and flexibility while choosing the ring frequency. Based on the proposed programmable STR (PSTR), a complete Programmable/Stoppable Oscillator (PSO) is designed and implemented. An efficient handshaking protocol between the processor and the oscillator is used to insure a proper switching from one frequency to another. The Oscillator shows glitch free and no truncated clocks at its output. The proposed architectures are physically implemented using the STMicroelectronics 45nm CMOS technology. The implemented chip shows High-speed, Low-Power, Low-Process Variability of the generated frequency, Wide-Range with Regular and Fine frequency step output. To the best of our knowledge, this chip is the first realization of a PSTR. As the PSTR is going to be used as one of the most important parts in the design of a process variability robust DVFS control for GALS-NoC systems, the effect of changing the power supply on the chip is studied as well. The chip shows very linear change in its frequency with respect to the variations in its power supply.

Since the voltage-frequency islands in a GALS-NoC system have different clock frequencies according to the tasks that they are handling as well as their influence by the local process variations within each island. Therefore, synchronizing theses different clock domains during communication phases was our next challenge. Consequently, we have proposed a new scheme based on the use of the PSTR as the main clock generating source in each GALS module. This scheme supports reliable communication between independently clocked GALS modules. However, instead of including in each PSTR

oscillator a standard element for pausing the clock, each synchronous module is proposed to have both an incoming and an outgoing clock signal, which have been obtained by opening the PSTR oscillator. Since these clock signals behave as handshake signals, handshake circuits can be used to synchronize the clocks. The simplest way synchronizing two different clock domains is then carried out by means of a C-element, which introduces only a small and predictable timing overhead. A multiplexer and DFF have been used in this circuit design, in order to adapt the instants on which we switch between no communication and data exchanges states, so we do not have any glitches nor truncated clock periods. The advantages of this scheme are: due to its self-timed nature, the control switch used with the PSTR allows connecting modules running at different clock frequencies with very low power consumption. It offers high throughput compared to a shared bus solution and a small area compared to FIFO-based solution. Moreover, it can be easily applied in different multipoint GALS interconnection schemes.

The next and the main objective of this thesis was to design an energy-efficient, process variability robust DVFS control system for future GALS-based NoC communication architectures with multiple voltage-frequency domains. Subsequently, we have introduced a new voltage-frequency island architecture based on the use of the asynchronous PSTR that adapts its maximum output clock frequency according to the performance evaluation of the processing domain. Evaluating the fabrication process quality and the local environmental parameters (voltage, temperature) is done with the help of activity monitors embedded in each processing domain. Afterwards, we have presented a variation-adaptive feedback control methodology for GALS-NoC architectures. More precisely, we developed techniques to dynamically control the speed of each GALS-NoC voltage-frequency island and provide robustness against the uncertainties and variations in the design parameters. At the same time, our techniques adapt the operating voltage and frequency to the variations in the workload to save power and energy consumption. Simulation results demonstrate robustness to parameter variations and efficient energy savings with a Matlab load model.

In order to design our proposed GALS-NoC control methodology and test its response on a real processing load with different process variability impacts, the MIPS R2000 microprocessor has been chosen as our main case study. First, it has been

synthesized using the STMicroelectronics 45nm CMOS libraries. Then, the information extracted from the analysis of the MIPS R2000 processor on different process variability corners has been used to parameterize the programmability of our asynchronous PSTR. Finally, the energy-efficient DVFS digital control system is fully designed and implemented on STMicroelectronics 45nm CMOS technology.

The implemented digital controller was able to smartly adapt the voltage/frequency couple with workload variations and, at the same time, copes with the process variations. Moreover, it also considers the voltage regulator and frequency oscillator switching performance levels, in order to correctly adapt their switching instants. According to the measured degree of process variability effect on the processing load performance, the programmable oscillator was able to generate the proper set of maximum clock frequencies which do not violate the MIPS R2000 critical path delay. Afterwards, the digital controller regularly adapts the estimated computational speed set points based on the workload variations with the exiting process variability impact.

For the different process variability corners defined by STMicroelectronics for their 45nm CMOS libraries, the proposed DVFS control system was able successfully achieve the requested performance by the OS. This allows us to not neglect most of the GALS-NoC voltage-frequency islands, even if they were under worst process variability effect, which has a direct impact on enhancing the fabrication yield. Moreover, the digital controller exploits the enhanced performance of the system under the best process conditions to save more power/energy consumption. One more advantage of the proposed DVFS architecture is its small area overhead, especially with GALS-NoC voltage-frequency islands, that contains more than one processing element (or larger than the MIPS R2000).

There are many possible extensions to this work such as:

1. The proposed PSTR-based GALS synchronization scheme can be extended and applied with different multipoint interconnection schemes.
2. The digital controller part of the DVFS system in the proposed GALS-NoC architecture can be developed to be completely asynchronous, in order to improve its robustness against process variations.

3. The new proposed GALS-NoC architecture can be physically implemented with different die positions per silicon wafer.
4. Experimental results with more applications and complex processors like ARM cores can be carried out to practically test the DVFS controller behavior with different workloads and process variability impact.
5. The idea can also be extended and practically tested with the 32nm CMOS technology and beyond.

Publications

Conference Publications

- Eslam Yahya, Oussama Elissati, Hatem Zakaria, Laurent Fesquet and Marc Renaudin, "Programmable/Stoppable Oscillator Based on Self-Timed Rings", ASYNC 2009 (15th IEEE International Symposium on Asynchronous Circuits and Systems), May 17-20, Chapel Hill, North Carolina, USA, pp. 3-12, 2009.

- Laurent Fesquet and Hatem Zakaria, "Controlling Energy and Process Variability in System-on-Chips: needs for control theory", MSC 2009 (3rd IEEE Multi-conference on Systems and Control), July 8-10, Saint Petersburg, RUSSIA, pp. 302-307, 2009.

- Hatem Zakaria, Sylvain Durand, Nicolas Marchand and Laurent Fesquet “Integrated Asynchronous Regulation for Nanometric Technologies”, VARI’10 (1st European workshop on CMOS Variability), May 26-27, Montpellier, France, pp. 86-91, 2010.

Workshop Publication

- Hatem Zakaria et al., "Integrated Asynchronous Regulation for Nanometric Technologies: Application to an Embedded Parallel System", MINATEC CROSSROADS, Grenoble, France, Poster, 2008.

Book Chapter

- Hatem Zakaria, Eslam Yahya and Laurent Fesquet “Self Adaption in SoCs”, in: Autonomic Networking-on-Chip (Bio-inspired Specification, Development, and Verification), Part of the Embedded Multi-core Systems (EMS) Book Series Edited by: Phan Cong-Vinh. 32 pages, Taylor & Francis, CRC Press, 2011.

References

[ALA 06] Alamir M.: 'Stabilization of Nonlinear Systems Using Receding- Horizon Control Schemes: A Parametrized Approach for Fast Systems', Lecture Notes in Control and Information Sciences, Springer-Verlag, London, 2006.

[ALB 08] Albea C., Canudas de Wit C. and Gordillo F.: 'Advanced Control Design for Voltage Scaling Converters', in Proceedings of the 34th IEEE Conference (IECON'08), pp. 79-84, 2008.

[ALB 09] Albea C., Canudas de Wit C. and Gordillo F.: 'Control and Stability Analysis for the V_{dd}-Hopping Mechanism', in Proceedings of the IEEE Conference on Control and Applications, pp. 320-325, 2009.

[AYD 01] Aydin H., Melhem R., Mosse D. and Alvarez P.: 'Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems', *In Proceedings of IEEE Real-Time Systems Symposium*, 2001.

[BEE 02] Beerel P.: 'Asynchronous Circuits: An Increasingly Practical Design Solution'. *Proceedings of ISQED'02*, IEEE Computer Society, pp. 367-372, 2002.

[BEI 06] Beigne E. and Vivet P.: 'Design of on-chip and off-chip interfaces for a GALS NoC architecture'. in Proceedings of the 12th IEEE International Symposium on Asynchronous Circuits and Systems, ASYNC '06, pp. 172-183, 2006.

[BER 05] Bertozzi D., et al.: 'NoC synthesis flow for customized domain specific multiprocessor systems-on-chip'. IEEE Trans. Parallel Distribution Systems, **16(**2), pp. 113-129, 2005.

[BER 92] Berkel K.: 'Beware the isochronic fork Integration', *VLSI journal*, **13**(2), pp. 103-128, 1992.

[BER 94] Berkel C., Burgess R., Kessels J., Peeters A., Roncken M., and Schalij F.: 'Asynchronous circuits for low power: A DCC error corrector'. IEEE Design Test., **11**(2), pp. 22–32, 1994.

[BER 99] Berkel C., Josephs M. and Nowick S.: 'Scanning the Technology. Applications of Asynchronous Circuits'. Proceedings of the IEEE, **87**(2), pp. 223-233, 1999.

[BOR 03] Borkar S., et al.: 'Parameter Variations and Impact on Circuits and Microarchitecture', in Proceedings of DAC, pp. 238-242, 2003.

[BOR 05] Borinski E., Arnim V. and Seegebrecht P.: 'Efficiency of body biasing in 90-nm CMOS for low-power digital circuits'. *IEEE Journal of Solid-state Circuits*, **40**(7), pp. 1549-1556, 2005.

[BOR 97] Bormann D. and Cheung P.: 'Asynchronous wrapper for heterogeneous systems', in Proceedings IEEE International Conference on Computer Design: VLSI in Computers and Processors ICCD '97, pp. 307-314, 1997.

[BOY 06] Boyer F., Epassa H. and Savaria Y.: 'Embedded power-aware cycle by cycle variable speed processor', *Computers and Digital Techniques, IEE Proceedings* **153(**4), pp. 283 – 290, 2006.

[BUI 06] Bui H. and Savaria Y.: 'High speed differential pulse-width control loop based on frequency-to-voltage converters', Proceedings of the 16th ACM Great Lakes symposium on VLSI, pp. 53-56, 2006.

[CHA 03] Chakraborty A. and Greenstreet M.: 'Efficient self-timed interfaces for crossing clock domains'. In Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems ASYNC '03, pp. 78-88, 2003.

[CHA 84] Chapiro D.: 'Globally-Asynchronous Locally-Synchronous Systems', Ph.D. thesis, Stanford University, report No. STANCS-84-1026, 1984.

[CHE 00] Chelcea T. and Nowick S.: 'Low-latency asynchronous FIFO's using token rings'. in Proceedings of the 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems ASYNC '00, pp. 210-220, 2000.

[CUM 09] Cummings C.: 'Clock domain crossing (cdc) design and verification techniques using system verilog'. SNUG-2008, Boston, 2009.

[CUM 94] Cummings U., Lines A., and Martin, A.: 'An asynchronous pipelined lattice structure filter'. *Proceedings of the International Symposium on Asynchronous Circuits and Systems*, Salt Lake City, Utah, USA, pp. 126-133, 1994.

[DAL 01] Dally W. and Towles B.: 'Route packets, not wires: On-chip interconnection networks', in Proceedings DAC, pp. 684-689, 2001.

[DHA 01] Dhar S., Dhar E. and Maksimovic D.: 'Switching regulator with dynamically adjustable supply voltage for low power VLSI', In Proceedings of the International Symposium on Low Power Electronics and Design, pp. 103-107, 2001.

[DHA 02] Dhar S., Maksimovic D. and Kranzen B.: 'Closed-loop adaptive voltage scaling controller for standard-cell ASICs', In ISLPED '02: Proceedings of the 2002 international symposium on Low power electronics and design, pp. 103-107, New York, NY, USA, ACM, 2002.

[DIE 03] Dielissen J., Radulescu A., Goossensa K. and Rijpkema E.: 'Concepts and implementation of the Philips network-on-chip', presented at the IP-Based SoC Des., Grenoble, France, 2003.

[DIK 99] Dike C. and Burton E.: 'Miller and noise effects in a synchronizing flip-flop', *IEEE Journal of Solid-State Circuits*, **34**(6), pp. 849-855, 1999.

[DOB 04] Dobkin R., Ginosar R. and Sotiriou C.: 'Data synchronization issues in GALS SoCs'. in Proceedings of the 10th International Symposium on Asynchronous Circuits and Systems ASYNC '04, pp. 170-180, 2004.

[DUR 09] Durand S. and Nicolas M.: 'Fast Predictive Control of Micro Controller's Energy-Performance Tradeoff', IEEE Control Applications, (CCA) & Intelligent Control, (ISIC), pp. 314-319, 2009.

[DUR 09a] Durand S. and Nicolas M.: 'Dispositif de Commande d'alimentation d'un Calculateur', Patent, No. 09/04686, 2009.

[DUR 09b] Durand S. and Nicolas M.: 'Dispositif de Commande d'alimentation d'un Calculateur', Patent, No. 09/01576, 2009.

[DUR 11] Durand S.: 'Reduction of the Energy Consumption in Embedded Electronic Devices with Low Control Computational Cost'. PhD thesis, Institut National Polytechnique de Grenoble, Grenoble, France, 2011.

[EBE 91] Ebergen J.: 'A formal approach to designing delay-insensitive circuits'. *Distributed Computing*, **5**(3), pp. 107-119, 1991.

[EBE 98] Ebergen J., Fairbanks S., and Sutherland I.: 'Predicting performance of micropipelines using Charlie diagrams', in Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC98, pp. 238–246, 1998.

[FAI 04] Fairbanks S. and Moore S.: 'Analog micropipeline rings for high precision timing', ASYNC'04, CRETE, Greece, IEEE, pp. 41–50, 2004.

[FLA 03] Flautner K., Flynn D. and Rives M.: 'A combined hardware-software approach for low-power SoCs: applying adaptive voltage scaling and the vertigo performance-setting algorithms', In Proceedings of DesignCon (System-on-Chip and ASIC Design Conference), 2003.

[FLA 04] Flautner K., Flynn D., Roberts D. and Patel D.: 'Iem926: An energy efficient SoC with dynamic voltage scaling', Design, Automation and Test in Europe Conference and Exhibition, **3**, pp. 324-327, 2004.

[FRE 07] Frenkil J. and Venkatraman S.: 'Power-gating Design Automation', Chapter 10, 'Closing the Power Gap between ASIC & Custom: Tools and Techniques for Low Power Design', No.1, 2007.

[GAR 00] Garside J. et al.: 'AMULET3i - an Asynchronous System-on-Chip'. Proceedings ASYNC 2000, *IEEE Computer Society Press*, pp. 162-175, 2000.

[GIN 03] Ginosar R.: 'Fourteen ways to fool your synchronizer', International Symposium on Asynchronous Circuits and Systems Async'03, pp. 1-8, 2003.

[GRU 01] Gruian F.: 'Hard Real-Time Scheduling Using Stochastic Data and DVS Processors', In Proceedings of the International Symposium on Low Power Electronics and Design, pp. 46-51, 2001.

[GUN 93] Gunawardena J.: 'A generalized event structure for the Muller unfolding of a safe net'. *Proceedings of the 4th International Conference on Concurrency Theory*, pp. 278-292, 1993.

[HAM 08] Hamon J., Fesquet L., Miscopein B. and Renaudin M.: 'High-Level Time-Accurate Model for the Design of Self-Timed Ring Oscillators', ASYNC'08, Newcastle, UK, IEEE, pp. 29 – 38, 2008.

[HAU 95] Hauck S.: 'Asynchronous design methodologies: An overview'. *Proceedings of the IEEE*, **83**(1), pp. 69-93, 1995.

[HED 87] Hedenstierna N. and Jeppson K.: 'CMOS circuit speed and buffer optimization', *IEEE Transactions on CAD*, **6(**3), 1987.

[HEL 04] Hellerstein J., Diao Y., Parekh S. and Tilbury D.: *Feedback Control of Computing Systems*. Wiley-IEEE Press, ISBN: 978-0-471-26637-2, 2004.

[JOY 04] Joycee M., Supratik C. and Dinesh S.: 'Evaluation of pausible clocking for interfacing high speed IP cores in GALS framework', in proceedings of 17th International Conference on VLSI Design, pp. 559-564, 2004.

[JUA 05] Juang P., et al.: 'Coordinated, Distributed, Formal Energy Management of Chip Multiprocessors', in Proceedings of the ISLPED, pp. 127-130, 2005.

[KIM 02] Kim W., Kim J. and Min S.: 'A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis', In Proceedings of Design, Automation and Test in Europe (DATE'02), pp. 788-794, 2002.

[KIN 02] Kinniment D., Bystrov A. and Yakovlev A.: 'Synchronization circuit performance', *IEEE Journal of Solid-State Circuits*, **37**(2), pp. 202-209, 2002.

[KRS 07] Krstic M., Grass E., Gurkaynak F., Vivet P.: 'Globally Asynchronous, Locally Synchronous Circuits: Overview and Outlook', In *IEEE Design & Test of Computers*, **24**(5), pp. 430-441, 2007.

[LEE 00] Lee S. and Sakurai T.: 'Run-time Voltage Hopping for Low power Real-Time Systems', In Proceedings of the 37th Design Automation Conference, pp. 806-809, 2000.

[LEE 07] Lee H., Chang N., Ogras U. and Marculescu R.: 'On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches'. *ACM Trans. Des. Autom. Electron. Syst.*, **1**(3), pp. 1-20, 2007.

[MAG 03] Magklis G., et al.: 'Dynamic Frequency and Voltage Scaling for a Multiple-Clock-Domain Microprocessor'. *IEEE Transactions on VLSI*, **23**(6), pp. 62-68, 2003.

[MAR 02] Martin S., Flautner K., Mudge T. and Blaauw D.: 'Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads', In Proceedings of the International Conference on Computer-Aided Design, pp. 721–725, 2002.

[MAR 05] Marculescu D. and Talpes E.: 'Energy Awareness and Uncertainty in Microarchitecture-Level Design', *IEEE Micro*, **25**(5), pp. 64-76, 2005.

[MAR 86] Martin A.: 'Compiling communicating processes into delay-insensitive VLSI circuits'. *Distributed Computing*, 1(4), PP. 226–234, 1986.

[MAR 90] Martin A.: 'The limitations to delay-insensitivity in asynchronous circuits'. In William J. Dally, editor, *Advanced Research in VLSI*, MIT Press, pp. 263-278, 1990.

[MAR 97] Martin A., Lines A., Manohar R., Nystrom M., Penzes P., Southworth R., Cummings U., and Lee T.: 'The design of an asynchronous MIPS R3000'. *In Proceedings of the 17th Conference on Advanced Research in VLSI*, pp. 164–181, 1997.

[MAT 05] Matthew W. Wayne P. and Ian G.: 'Synchro-Tokens: A Deterministic GALS Methodology for Chip-Level Debug and Test', *IEEE Transactions on Computers*, **54**(12), pp. 1532-1546, 2005.

[MAY 90] Mayne D. and Michalska H.: 'Receding Horizon Control of Nonlinear Systems'. *IEEE Transactions on Automatic Control*, **35**(7), pp. 814-824, 1990.

[MIE 07] Miermont S., Vivet P. and Renaudin M.: 'A Power Supply Selector for Energy- and Area-Efficient Local Dynamic Voltage Scaling', in Proceedings of PATMOS, pp. 556-565, 2007.

[MIE 08] Miermont S.: 'Contrôle distribué de la tension d'alimentation dans les architectures GALS et proposition d'un sélecteur dynamique d'alimentation'. PhD thesis, Institut National Polytechnique de Grenoble, Grenoble, France, 2008.

[MUL 07] Mullins R. and Moore S.: 'Demystifying Data-Driven and Pausible Clocking Schemes', ASYNC'07, Berkeley, California, USA, IEEE, pp. 175–185, 2007.

[MUL 59] Muller D. and Bartky W.: 'A theory of asynchronous circuits'. *Proceedings of International Symposium on the Theory of Switching*, pp. 204-243, 1959.

[MUT 00] Muttersbach, J. Villiger and Fichtner, W.: 'Practical design of globally-asynchronous locally-synchronous systems' In Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems ASYNC'00, pp. 52-59, 2000.

[MUT 99] Muttersbach J., Villiger T., Kaeslin H., Felber N. and Fichtner, W.: 'Globally-asynchronous locally-synchronous architectures to simplify the design of on-chip systems', in Proceedings of 12th Annual IEEE International ASIC/SOC Conference, pp. 317 – 321, 1999.

[MYE 01] Myers C.: *Asynchronous Circuit Design*, New York: John Wiley & Sons, 2001.

[NIE 94] Nielsen L., Niessen C., Sparsø J., and Berkel C.: 'Low-power operation using self-timed circuits and adaptive scaling of the supply voltage'. *IEEE Transactions on VLSI Systems*, **2**(4), pp. 391-397, 1994.

[NIY 05] Niyogi K. and Marculescu D.: 'Speed and Voltage Selection for GALS Systems based on Voltage/Frequency Islands', in Proceedings of ASP-DAC, pp. 1-6, 2005.

[OGR 07] Ogras U., et al.: 'Voltage-Frequency Island Partitioning for GALS-based Networks-on-Chip', in Proceedings of DAC, pp. 110-115, 2007.

[OGR 08] Ogras U., et al.: 'Variation-Adaptive Feedback Control for Networks-on-Chip with Multiple Clock Domains', in Proceedings of DAC, pp. 614-619, 2008.

[OLS 99] Olsson T., Torkelsson M., Nilsson P., Hemani A. and Meincke T.: 'A digitally controlled on-chip clock multiplier for globally asynchronous locally synchronous systems', *in proceedings of the IEEE 42nd Midwest Symposium on Circuits and Systems*, **1**, pp. 84-87, 1999.

[PAN 02] Panyasak D., Sicard G., and Renaudin M.: 'A Current Shaping Methodology for Low EMI Asynchronous Circuits' EMC Compo, Toulouse, France, 2002.

[PAN 05] Pangrle B. and Kapoor S.: 'Leakage power at 90nm and below', Technical report, Synopsis Inc, 2005.

[PAV 98] Paver N., Day P., Farnsworth C., Jackson D., Lien W., and Liu J.: 'A low-power, low-noise configurable self-timed DSP', in Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems, pp. 32–42, 1998.

[PFI 07] Pfitzner A., Kuzmicz W., Piwowarska E. and Kasprowicz D.: 'Static power consumption in nano-CMOS circuits: Physics and modeling'. In Proceeding of the 14th International Conference Mixed Design of Integrated Circuits and Systems, pp. 163-168, 2007.

[PIL 01] Pillai P. and Shin K.: 'Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems'. In Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP'01), pp. 89-102, 2001.

[PIN 01] Pinheiro E., Bianchini R., Carrera E. and Heath T.: 'Dynamic Cluster Reconfiguration for Power and Performance'. In Benini L., Kandemir M. and Ramanujam J., editors, *Compilers and operating systems for low power*, Kluwer Academic Publishers, Norwell, USA, pp. 75-93, 2001.

[POK 07] Pokhrel K.: 'Physical and Silicon Measures of Low Power Clock Gating Success: An Apple to Apple Case Study', SNUG, http://www.snug-universal.org, 2007.

[POU 01] Pouwelse J., Langendoen K. and Sips H.: 'Dynamic voltage scaling on a low-power microprocessor', In MobiCom'01 Proceedings of the 7th annual international conference on Mobile computing and networking, New York, NY, USA, ACM, pp. 251-259, 2001.

[REB 09] Rebaud B.: 'Evolution vers des Architectures de Systèmes Intégrés Auto-Adaptatives et Tolérantes aux Variations Technologiques et Environnementales'. PhD thesis, LIRMM, France, 2009.

[REB 10] Rebaud B., et al.: 'Digital Timing Slack Monitors and Their Specific Insertion Flow for Adaptive Compensation of Variabilities'. In Monteiro J. and Leuken R., editors, *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, Lecture Notes in Computer Science, Springer-Verlag, **5953**, pp. 266-275, 2010.

[REN 03] Renaudin M. and Fragoso J.: 'Asynchronous Circuits Design: An Architectural Approach'. In: Guntzel J. and Reis R. *V Escola de Microeletrônica da SBC-Sul*, Rio Grande, Brazil, pp. 1-43, 2003.

[REN 96] Renaudin M., Elhassan B., and Guyot A.: 'A New Asynchronous Pipeline Scheme: Application To The Design Of A Self-Timed Ring Divider'. *IEEE Journal of Solid-State Circuits*, **31**(7), pp. 1001-1013, 1996.

[RIG 02] Rigaud J.: 'Spécification de bibliothèques pour la synthèse de circuits asynchrones'. PhD thesis, Institut National Polytechnique de Grenoble, Grenoble, France, 2002.

[RIU 06] Rius J, Meijer M. and Pineda de Gyvez J.: 'An Activity Monitor for Power/Performance Tuning of CMOS Digital Circuits'. *Journal of Low Power Electronics*, **2**(1), pp. 80-86, 2006.

[SEM 02] Semeraro G., et al.: 'Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling', in HPCA '02: Proceedings of the 8th International Symposium on High-Performance Computer Architecture, pp. 29-40, 2002.

[SEM 06] Semiconductor Association: 'The International Technology Roadmap for Semiconductors', 2006.

[SHA 03] Shang L., Peh L. and Jha N.: 'PowerHerd: Dynamically Satisfying Peak Power Constraints in Interconnection Networks', in Proceedings of the International. Symposium On Supercomputing, 2003.

[SHA 98] Shams M., Ebergen J. and Elmasry M.: 'Modeling and comparing CMOS implementations of the c-element'. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **6**(4), pp. 563–567, 1998.

[SHI 00] Shin Y., Choi K. and Sakurai T.: 'Power Optimization of Real-Time Embedded Systems on Variable Speed Processors', In Proceedings of the International Conference on Computer-Aided Design, pp. 365-368, 2000.

[SHI 01] Shin D., Kim J. and Lee S.: 'Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications', *IEEE Design and Test of Computers*, **18**(2), pp. 20-30, 2001.

[SIM 04] Simunic T., Boyd S. and Glynn P.: 'Managing Power Consumption in Networks on Chips', *in Transactions on VLSI*, **12**(1), pp. 96-107, 2004.

[SPA 01] SparsØ J. and Furber S.: 'Principles of Asynchronous Circuit Design. A System Perspective'. Kluwer Academic Publishers, 2001.

[STO 03] Stork M.: 'Digital building block for frequency synthesizer and fractional phase locked loops', SympoTIC, IEEE, pp. 126 – 129, 2003.

[SUT 89] Sutherland I.: 'Micropipelines' Communications of the ACM (Association of Computing Machinery), **32**(6), pp. 720–738, 1989.

[SVI 01] Svilan S., Burr J. and Tyler G.: 'Effects of elevated temperature on tunable near-zero threshold CMOS', In Proceeding of the International Symposium on Low-Power Electron and Design, pp. 255-258, 2001.

[VAR 03] Varma A., Ganesh B., Sen M., Choudhury S., Srinivasan L. and Jacob B.: 'A control-theoretic approach to dynamic voltage scheduling', In CASES'03: Proceedings of the international conference on Compilers, architecture and synthesis for embedded systems, New York, USA, ACM, pp. 255-266, 2003.

[VIL 03] Villiger T., Kaslin H., Gürkaynak F., Oetiker S. and Fichtner W.: 'Self-timed Ring for Globally-Asynchronous Locally-Synchronous Systems', in Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems ASYNC'03, pp. 141-150, 2003.

[WIL 91] Williams T. and Horowitz M.A.: 'A zero-overhead self-timed 160 ns. 54 bit CMOS divider'. *IEEE Journal of Solid State Circuits*, **26**(11), pp. 1651-1661, 1991.

[WIL 94] Williams T.: 'Performance of Iterative Computation in self-timed rings'. *Journal of VLSI signal processing*, **7**, pp. 17-31, 1994.

[WIL 95] Williams T., Patkar N., and Shen G.: 'SPARC64: A 64-b 64-activeinstruction out-of-order-execution MCM processor'. *IEEE Journal of Solid-State Circuits*, **30**(11), pp. 1215-1226, 1995.

[WIN 01] Winstanley A. and Greenstreet M.: 'Temporal properties of selftimed rings', in Proc. 11th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME), 2001.

[WIN 02] Winstanley A., Garivier A., and Greenstreet M.: ‘An event spacing experiment’, in Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC 02, pp. 47–56, 2002.

[WU 04] Wu Q., et al.: ‘Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors’, in Proceedings of ASPLOS, pp. 248-259, 2004.

[YAH 09] Yahya E., Elissati O., Zakaria H., Fesquet L. and Renaudin M. ‘Programmable/Stoppable Oscillator Based on Self-Timed Rings’, ASYNC 09 (15th IEEE International Symposium on Asynchronous Circuits and Systems), Chapel Hill, North Carolina, USA, pp. 3-12, 2009.

[YUN 96] Yun K., Beerel P., and Arceo J.: ‘High-Performance Asynchronous Pipeline Circuits’. IEEE Computer Society, ASYNC96, Aizu-Wakamatsu, Japan, pp. 17-28, 1996.

[YUN 96a] Yun K. and Donohue R.: ‘Pausible Clocking: A First Step Toward Heterogeneous Systems’, In Procedings of International Conference on Computer Design ICCD, pp. 118-123, 1996.

[ZAK 10a] Zakaria H., Durand S., Marchand N. and Fesquet L.: ‘Integrated Asynchronous Regulation for Nanometric Technologies’, in VARI Workshop, Montpellier, France, 2010.

[ZAK 10b] Zakaria H., Yahya E. and Fesquet L.: ‘Self Adaption in SoCs’, Book Chapter, In Cong-Vinh P., editor, *Autonomic Networking-on-Chip (Bio-inspired Specification, Development, and Verification)*, Embedded Multi-Core Systems, CRC Press, 2011.

[ZEB 05] Zebilis V. and Sotiriou C.: ‘Controlling event spacing in self-timed rings’, ASYNC’05, New York, USA, IEEE, pp. 109 – 115, 2005.

[ZHU 02] Zhuang S., Li W., Carlsson J., Palmkvist K. and Wanhammar L.: 'Asynchronous Data Communication with Low Power for GLAS Systems’, in Proceedings of the 9th international conference on Electronics, Circuits and Systems, **2**, pp. 753-756, 2002.

[ZHU 03] Zhu Y. and Mueller F.: ‘Feedback dynamic voltage scaling DVS-EDF scheduling: Correctness and PID-feedback’. Technical report, Workshop on Compilers and Operating Systems for Low Power, 2003.

Architecture Asynchrone pour L'Efficacité Energétique et L'Amélioration du Rendement en Fabrication dans les Technologies Décananométriques: Application à un Système sur Puce Multi-Cœurs

Résumé - La réduction continuelle des dimensions dans les technologies CMOS a ouvert la porte à la conception de circuits complexes multi-cœurs (SoC). Malheureusement dans les technologies nanométriques, les performances des systèmes intégrés après fabrication ne sont pas complètement prédictibles. En effet, les variations des procédés de fabrication sont très importantes aux échelles des puces. Par conséquent, la conception de tels systèmes dans les technologies nanométriques est désormais contrainte par de nombreux paramètres tels que la robustesse aux variations des procédés de fabrication et la consommation d'énergie. Ceci implique de disposer d'algorithmes efficaces, intégrés dans la puce, susceptibles d'adapter le comportement du système aux variations des charges des processeurs tout en faisant face simultanément aux variations des paramètres qui ne peuvent pas être prédits ou modélisées avec précision au moment de la conception. Dans ce contexte, ce travail de thèse porte sur la conception de systèmes dit « GALS » (Globally Asynchronous Locally Synchronous) conçus autour d'un réseau de communication intégré à la puce (Network-on-Chip ou NoC) exploitant les nouvelles générations de technologie CMOS. Une nouvelle méthode permettant de contrôler dynamiquement la vitesse des différents îlots du NoC grâce à un contrôle de la tension et de la fréquence en fonction de la qualité locale des procédés de fabrication sur chaque îlot est proposée. Cette technique de contrôle permet d'améliorer les performances du système en consommation, et d'augmenter son rendement en fabrication grâce à l'utilisation des synergies au sein du système intégré. La méthode de contrôle est basée sur l'utilisation d'un anneau asynchrone programmable capable de prendre en compte la charge de travail dynamique et les effets de la variabilité des procédés de fabrication. Le contrôleur évalue en particulier la limite supérieure de fréquence de fonctionnement pour chaque domaine d'horloge. Ainsi, il n'est plus nécessaire de garantir les performances temporelles de chaque nœud au moment de la conception. Cela relâche considérablement les contraintes de fabrication et permet du même coup l'amélioration du rendement.

Mots Clés: Logique Asynchrone, Technologies Nanométriques, Variabilité des Procédés de Fabrication, GALS-NoC, DVFS Mécanismes de Contrôle, Conceptions pour le Rendement.

Asynchronous architecture for power efficiency and yield enhancement in the decananometric technologies: Application to a multi-core system-on-chip

Abstract - Continuous scaling of CMOS technology push circuit designs towards multi-core complex SoCs. Moreover, with the nanometric technologies, the integrated system performances after fabrication will not be fully predictable. Indeed, the process variations really become huge at the chip scale. Therefore the design of such complex SoCs in the nanoscale technologies is now constrained by many parameters such as the energy consumption and the robustness to process variability. This implies the need of efficient algorithms and built-in circuitry able to adapt the system behavior to the workload variations and, at the same time, to cope with the parameter variations which cannot be predicted or accurately modeled at design time. In this context, this thesis work addresses the design of GALS-based NoC architectures in the upcoming CMOS technologies. A novel methodology to dynamically control the speed of different voltage-frequency NoC islands according to the process variability impact on each domain is proposed. This control technique can improve the performances, the energy consumption, and the yield of future SoC architectures in a synergistic manner. The control methodology is based on the design of an asynchronous programmable self-timed ring where the controller takes into account the dynamic workload and the process variability effects. The controller especially considers the operating frequency limit which does not exceed the maximum locally allowed value for a given clock domain. With such an approach, it is no more required to separately guaranty the performance for each node. This drastically relaxes the fabrication constraints and helps the yield enhancement.

Keywords: Asynchronous Logic, Nanometric Technologies, Process Variations, GALS-NoC, DVFS Control Schemes, Design for Yield.

Thèse préparée au laboratoire TIMA (Techniques de l'Informatique et de la Microélectronique pour l'Architecture des ordinateurs), 46 Avenue Félix Viallet, 38031, Grenoble Cedex, France

ISBN: 978-2-84813-164-1