Distributed clock generator for globally and locally synchronous chips with a large size

Chuan Shan

THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

École Doctorale Informatique, Télécommunications et Électronique
(EDITE)

Présentée par :
Chuan SHAN

Pour obtenir le grade de :
DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

GÉNÉRATEUR DISTRIBUÉ D'HORLOGE POUR PUCES GLOBALEMENT ET LOCALEMENT SYNCHRONES DE GRANDE TAILLE

Présentée le :
14.11.2014
Le jury est composé de :

M. Patrick GIRARD	LIRMM Montpellier	Rapporteur
M. Laurent FESQUET	TIMA Grenoble	Rapporteur
M. Gérard BERRY	Collège de France	Examinateur
M. Daniel ETIEMBLE	Université Paris Sud, LRI	Examinateur
M. Alain GREINER	UPMC, LIP6	Examinateur
M. François ANCEAU	CNAM	Directeur de Thèse
M. Dimitri GALAYKO	UPMC, LIP6	Co-directeur de Thèse

DOCTORAL DISSERTATION
PIERRE AND MARIE CURIE UNIVERSITY

Doctoral School of Informatics, Telecommunications and Electronics (EDITE)

Presented by:
Chuan SHAN

To obtain the degree of :
DOCTOR OF PHILOSOPHY AT UNIVERSITY OF PIERRE AND MARIE CURIE

Thesis title :

DISTRIBUTED CLOCK GENERATOR FOR GLOBALLY AND LOCALLY SYNCHRONOUS CHIPS WITH A LARGE SIZE

Presented on :
11.14.2014

Members of jury :

M. Patrick GIRARD	LIRMM Montpellier	Reviewer
M. Laurent FESQUET	TIMA Grenoble	Reviewer
M. Gérard BERRY	Collège de France	Examinator
M. Daniel ETIEMBLE	Université Paris Sud, LRI	Examinator
M. Alain GREINER	UPMC, LIP6	Examinator
M. François ANCEAU	CNAM	Supervisor
M. Dimitri GALAYKO	UPMC, LIP6	Co-Supervisor

Abstract

This thesis addresses the problem of global synchronization of large system on chip (SoC) in the context of deep submicron technologies.

With the development of the silicon microtechnology and with the increase of density of integration, conventional clock distribution systems become more and more difficult to be implemented in modern chips. Some designers turn toward the asynchronous communications protocols, in order to remove the need of a global clock, however, the cost of this choice is a lower verification level and a high complexity of design. This research focuses on the study of an alternative clock generation technique, allowing implementation of highly reliable synchronous digital circuit in deep submicron CMOS technologies.

My PhD work is subsequent to the project HODISS funded by ANR, in which a distributed network of all digital phase-locked loop (ADPLL) was presented for the first time. The basic idea of this approach consists in generating clock signals locally by oscillators in each synchronous clocking area (SCA), and making these oscillators coupled with their neighboring ones in phase by using ADPLL technique. If well designed, all the locally generated clock signals should be synchronized with the reference clock both in frequency and in phase.

My PhD project aims study and implementation on silicon of a large network of ADPLLs (10×10), containing 100 nodes generating each a clock for the local digital circuitry. Comparing to the study curried out previously in the HODISS project, the design of blocks composing the network was adapted to the constraints related to the network scale. For instance, the resolution of the phase measurement has been increased (20 ps), the power consumption was reduced (1 mW per node). The modeling of the ADPLL networks at several abstraction levels and by different tools (VHDL, Spice, FPGA) allowed a response to study the quality of the global synchronization between the oscillators in function of different parameters of the network, and in particular, on the network size.

During my PhD project, the prototype 10×10 was implemented on silicon generating clocks in the range 903-1161 MHz. It highlights a maximum phase error of less than 40 ps between two clocks in any neighboring zones. The implemented prototype is under fabrication at the period of editing of this manuscript.

Another important result of this study is the analysis of phase error between two non-neighboring oscillators in distance. By studying an FPGA prototype of the network, we obtained that maximum phase error at steady state between any observed clock signal and the reference signal is less than three steps of the PFD quantification steps.

In order to validate the performances of the clock synchronization in the implemented ASIC, we designed an on-chip clocking error measurement circuit, whose operation is based

on a measure of the integrity of a periodic sequence transmitted between two clocking domains. This circuit has a low rate for the off-chip readout (several MHz), and a high resolution ($\pm$2.5 ps).

Reconfigurability is another attractive feature of the distributed network of ADPLLs. In this research, we have explored this feature and proposed a novel topology with different configurations for nodes on the border and in the kernel of the network. This topology has an advantage in prohibiting phase error propagation and reflection.

Thesis title: Distributed clocking for synchronous SoC

Key words: synchronous clocking, multioscillator architecture, all-digital phase locked loop, on-chip clock error characterization

Thesis Supervisor: François ANCEAU, Professor at Conservatoire national des arts et métiers

Thesis Co-Supervisor: Dimitri GALAYKO, Associate Professor at Pierre and Marie Curie University (Paris VI)

It is possible to own too much: a man with one watch knows what time it is, a man with two watches is never quite sure

Lee Segall.

Acknowledgements

I would like to take this opportunity to express my sincere thanks to all those who participated directly or indirectly to the success of my thesis. I would never have been able to finish my dissertation without the guidance of my committee members, help from friends, and support from my family.

I would like to express my special appreciation and thanks to my supervisor Dr. Dimitri Galayko, for his excellent guidance, patience, giving my this opportunity and providing me with an excellent atmosphere for doing research. I would also like to thank my supervisor Dr. François Anceau for his brilliant ideas and suggestions, which inspired me a lot. Without their supervision and constant help this dissertation would not have been possible.

I would like to thank our collaborators Olivier Billoint and Sylvain Féruglio for their effort spent on this project. Their great experience and competence make our research team more complete.

I would like also to thank all my colleagues at SoC-LIP6. They created a warm environment in the SoC department. A special thanks to Eldar Zianbetov and Mohammad Javidan. They were always willing to help and give their best suggestions. Without their help, I might spend much more time on the chip implementation.

Many thank to my parents for all of the sacrifices that they have made on my behalf. They were always supporting me and encouraging me with their best wishes.

Finally I would like to thank all my friends in France and in China. I am grateful for their constant support and encouragement.

Chuan SHAN

Contents

Introduction: motivation de l'étude xxiii
Défis de génération d'une horloge globale dans les SoCs xxiii
Défis de caractériser l'erreur de phase entre deux signaux d'horloge xxv
Réseau de PLLs tout numériques pour la génération d'horloge xxvi
Résumé de contribution xxviii
Conception des composants xxx
Convertisseur temps-numérique (TDC) xxx
Filtre numérique xxx
Le réseau d'ADPLLs tour numériques réalisé xxxii
Caractérisation sur puce de l'erreur de phase entre deux signaux d'horloge xxxv
Conclusion xxxvii
Perspectives et travail futur xli
Modélisation d'ADPLL pour l'étude de l'erreur de phase résiduelle dans l'état d'équilibre xli
Exploration de la propriété de tolérance de faute xli
La distribution d'horloge pour les circuits en 3-D xlii

1 Introduction **1**
1.1 Area of focus 1
1.1.1 Problem of clocking in large digital circuits 2
1.2 Environment of the PhD project: the starting point and motivations 5
1.2.1 Network of coupled PLLs for clocking: history of the concept . . . 5
1.2.2 Digital phase synthesis 7
1.2.3 Presentation of an ADPLL network prototype designed at LIP6 prior to my PhD thesis 9
1.2.4 Phase frequency detector (PFD) 9
1.2.5 Digital loop control of ADPLL network node 16
1.2.6 Digitally controlled oscillator (DCO) 17
1.2.7 Modeling of ADPLL and of ADPLL network 21
1.2.8 Stability of the PLL networks 23
1.2.9 Multiplicity of synchronization modes 24
1.2.10 Discussion of test results of the implemented prototype 26
1.3 Original contribution of my PhD project 29
1.4 Thesis outline 31

2 Network of distributed ADPLLs **33**
2.1 Introduction . . . 33
2.2 The architecture of clocking network proposed in this PhD project . . . 35
2.3 Impact of quantization in ADPLL on its operation in steady state . . . 36
2.3.1 Step 1: Impact of PFD and DCO quantization steps on the residual error . . . 36
2.3.2 Step 2: impact of rounding in digital filter on the correction of residual phase error . . . 40
2.3.3 Step 3: validation of block parameters by transient simulations . . . 43
2.4 Specification of the network . . . 46
2.5 Conclusion . . . 48

3 ADPLL blocks design **49**
3.1 Phase frequency detector (PFD) . . . 50
3.1.1 The digital PFD architecture . . . 50
3.1.2 Improvement of time-to-digital converter . . . 50
3.1.3 Implementation of PFD . . . 55
3.2 Digital filter in the ADPLL network . . . 57
3.2.1 Architecture of digital filter . . . 57
3.2.2 Implementation of the filter . . . 59
3.3 Digitally controlled oscillator (DCO) . . . 60
3.3.1 DCO Architecture . . . 60
3.3.2 Control algorithm . . . 61
3.3.3 Implementation . . . 63
3.3.4 Serial programming interface (SPI) . . . 68
3.3.5 Simulation results . . . 69
3.4 Conclusion . . . 75

4 Built-In Clock Error Characterization Circuit **77**
4.1 Introduction . . . 77
4.2 State of art . . . 79
4.3 Test methodology . . . 82
4.3.1 Measurement theory . . . 82
4.3.2 Architecture of measurement circuit . . . 85
4.4 Low frequency discrete circuit prototype . . . 87
4.5 High frequency on-chip prototype . . . 90
4.5.1 Voltage-controlled delay (VCD) . . . 90
4.5.2 Physical design of test circuit on silicon . . . 93
4.5.3 Modeling of clock generator for system verification . . . 93
4.5.4 Simulation results . . . 97
4.6 Procedure of measurement . . . 104
4.7 Conclusion . . . 106

5 Clock network FPGA prototyping **107**
5.1 Introduction . . . 107
5.2 Implementation of FPGA based blocks . . . 110
5.2.1 Synthesizable DCO . . . 110
5.2.2 Synthesizable TDC . . . 113
5.3 Experimental results . . . 116
5.3.1 Stability and prevention of mode-lock . . . 116
5.3.2 Phase error between two remote local clocks . . . 122
5.4 Conclusion . . . 128

6 Clock network silicon implementation **129**
6.1 Introduction . . . 129
6.2 Methodology of chip design . . . 130
6.3 Implementation of local clock generator (NODE) . . . 134
6.4 Floorplan of the chip . . . 138
6.5 Design for test(DFT) . . . 139
6.5.1 Chip programming . . . 140
6.5.2 Built-in test circuits placement . . . 140
6.5.3 Definition of the input/outputs of the chip . . . 141
6.6 Chip layout . . . 145
6.7 Simulation results . . . 146
6.8 Conclusion . . . 148

7 "Swimming pool"-like distributed architecture **149**
7.1 Introduction . . . 149
7.2 Modeling of infinite ADPLL network by a continuous wave propagation medium . . . 150
7.2.1 From a discrete network to a continuous medium . . . 151
7.2.2 An analogy with damped wave equation . . . 153
7.3 ADPLL network with limited surface . . . 154
7.4 Simulation results . . . 156
7.5 Conclusion . . . 160

8 ADPLL with sliding-window for wide range frequency tracking **161**
8.1 Introduction . . . 161
8.2 State of art . . . 163
8.3 "Sliding window" architecture . . . 166
8.3.1 Reference frequency indicator (RFI) . . . 166
8.3.2 Coarse frequency adjustment . . . 167
8.3.3 Phase error correction . . . 172
8.4 Comparison with conventional PLL . . . 174
8.4.1 Functional Simulation results . . . 174

8.4.2 Power consumption comparison 175
8.5 Clock distribution network using "sliding window" ADPLL 176
8.5.1 Network structure 176
8.5.2 Evaluation of functional performance of system 177
8.6 Conclusion 182

9 Conclusion and Perspectives **183**
9.1 Thesis summary and conclusions 183
9.2 Future work 186
9.2.1 Modelling of ADPLL for the study of residual phase error in steady state 186
9.2.2 Exploration of fault-tolerance property 187
9.2.3 Clock distribution for 3-D chip 187

Appendices **191**

A VHDL models of the ADPLL blocks **193**

B VHDL models for built-in test circuit **219**

C Matlab scripts **225**

D FPGA prototyping of the clocking network **233**

E Tcl script for automatic floorplan of network **239**

Bibliography **247**

List of Figures

1 Synchronisation des circuits numériques complexes xxiv
2 **Definition d'erreur d'horloge** Δt_i xxvi
3 Topology of the proposed clock network and architecture of the network node . xxvii
4 Block diagram of proposed time-to-digital converter xxx
5 **Filtre de boucle pour le traitement du signal d'erreur**: quatre contrôleurs de gain d'entrée suivi par l'addition à quatre entrées, filtre PI et trois décodeurs B2T. xxxi
6 **Proposed architecture for PLL** xxxii
7 **Clocking network architecture** xxxiii
8 Local clock signals together with reference xxxiv
9 Maximal value of phase error in function of distance to the reference clock . . . xxxv
10 Layout of the test chip of the clock network xxxvi
11 **Analyse théorique**: (a) PDF d'origine de l'incertitude d'horloge; (b) PDF avec un décalage $\Delta = \Delta_x$; (c) a vs. Δ; (d) ER vs. Δ xxxvii
12 **Architecture of test circuit** xxxviii
13 **Différents réseaux de distribution d'horloge 3-D au sein du circuit de test de [38]**: (a) H-arbres, (b) H-arbre et anneaux/mailles locales, (c) H-arbre et anneaux globaux xliii
14 **L'approche proposée de distribution d'horloge 3-D en utilisant le réseau d'ADPLL** xliv

1.1 Clock domains in a SoC 1
1.2 Examples of conventional clock distribution tree structures 2
1.3 Basic idea of multioscillator clocking approach 5
1.4 Topology of the proposed clock network and architecture of the network node . 6
1.5 Phase coupling between two oscillators 7
1.6 Block diagram of the ADPLL 8
1.7 **Structure of the first ADPLL network** 10
1.8 **Structure of a node in ADPLL network** 10
1.9 The phase/frequency detector 11
1.10 Proposed phase/frequency detector for clock network 11
1.11 Principle of operation of proposed PFD 12
1.12 Schematic diagram of the bang-bang phase/frequency detector 13
1.13 Proposed in [59] arbiter circuit 13
1.14 Block diagram of proposed time-to-digital converter 15

1.15 Error signal processing block . . . 16
1.16 **DCO architecture:** (a) structure, (b) main inverters, (c) circuit diagram of the coarse tuning cell, (d) circuit diagram of the additional coarse tuning cell, (e) circuit diagram of the fine tuning tuning cell . . . 19
1.17 Virtual extension of the 8^{th} stage of the oscillator principle . . . 19
1.18 **LTI model of ADPLL for Z-domain transfer function calculation** . . . 23
1.19 Representation of the PLL network for stability study in [47] . . . 24
1.20 Cyclic nature of the conventional analog linear phase comparator . . . 25
1.21 Illustration of the mode-locking phenomenon in a 2×2 mesh network . . . 25
1.22 Dynamic reconfiguration of the network from uni- to bidirectional . . . 26
1.23 Synchronous clocks in the bidirectional configuration . . . 27
1.24 Outputs of the PFDs in bidirectional configuration . . . 28
1.25 **Structure of work contribution** . . . 30

2.1 **Clocking network architecture** . . . 35
2.2 **Phase evolution of reference clock and divided oscillator clock**: $\omega_1 < \omega_0 < \omega_2$ 38
2.3 **Bode diagram of the LTI model of the system (Fig. 1.18)**: $K = 1.2e7$ rad/s and $\beta/\alpha = 0.012$ . . . 42
2.4 **Time simulations with different TDC resolution and same filter/DCO parameters (VHDL model)**: reference clock frequency: 249.5 MHz, DCO 3 in Tab. 2.1 used . . . 44
2.5 **Time simulations with the same loop gain and β/α ratio (VHDL model)**: reference clock frequency: 249.5 MHz, DCO 3 in Tab. 2.1 used . . . 45
2.6 **The maximum values of residual phase errors of an ADPLL in the steady state with different PFD and filter coefficients (VHDL model)** . . . 45

3.1 Proposed phase/frequency detector for clock network . . . 50
3.2 Time-to-digital converter . . . 51
3.3 Block diagram of proposed time-to-digital converter . . . 52
3.4 Schematic diagram of the delay cells in Vernier TDC . . . 53
3.5 Simulated transfer function of the designed flash time-to-digital converter . . . 54
3.6 Layout of the proposed PFD . . . 56
3.7 **Loop filter for error signal processing**: four input gain controllers followed by the four-input adder, PI filter and three B2T decoders. . . . 57
3.8 **Programming sequence of parameters** . . . 59
3.9 Core of the proposed oscillator . . . 61
3.10 Ring oscillator cell control table . . . 63
3.11 Schematic of a main inverter of oscillator . . . 64
3.12 Schematic diagram of the coarse tuning cells . . . 64
3.13 Schematic diagram of the additional coarse tuning cells . . . 65
3.14 Schematic diagram of the fine tuning cells . . . 65
3.15 Schematic diagram of the feedback frequency divider . . . 65

3.16 Floorplan of the designed oscillator . . . 66
3.17 Interdigital multi-finger power routing . . . 67
3.18 Layout of the designed oscillator . . . 67
3.19 Schematic of the programming interface . . . 69
3.20 Cascading the programming interfaces of several blocks . . . 69
3.21 **Simulated output frequency versus frequency control word (FCW)**: typical condition . . . 70
3.22 **Frequency step vs. FCW**: typical condition . . . 70
3.23 **Simulated output frequency versus frequency control word (FCW) in different process corners**: process variations: TT, FF and SS . . . 71
3.24 **Simulated output frequency versus frequency control word (FCW) with different supply voltages**: 1 V, 1.1 V and 1.2 V . . . 72
3.25 **Simulated output frequency versus frequency control word (FCW) at different temperatures**: 0°C, 27°C, 85°C and 125°C . . . 73
3.26 **Monte-Carlo simulation at center frequency** . . . 73
3.27 **Power consumption vs. FCW in different process corners**: process variations: TT, FF and SS . . . 74

4.1 **Definition of the clock error Δt_i** . . . 79
4.2 **Gaussian distribution of clock phase error** . . . 80
4.3 **Delay chain based "Skitter" circuit proposed in [13]** . . . 81
4.4 **Basic architecture of measurement circuit** . . . 82
4.5 **Received data integrity**: fixed static error . . . 83
4.6 **Received data integrity**: dynamic error . . . 83
4.7 **Theory analysis**: (a) Original PDF of clock uncertainty; (b) PDF with a shift $\Delta = \Delta_x$; (c) a vs. Δ; (d) ER vs. Δ . . . 85
4.8 **ER distribution with respect to single positive delay** . . . 86
4.9 **Architecture of test circuit** . . . 86
4.10 **Measurement environment of prototype** . . . 87
4.11 **Generation of clk2 with static and dynamic errors** . . . 87
4.12 **Variable delay circuit in discrete circuit prototype** . . . 88
4.13 **Test of prototype without skew** . . . 88
4.14 **Test of prototype with a skew of -40 ns** . . . 89
4.15 **Test of prototype with a skew of -40 ns** . . . 89
4.16 **ASIC prototype architecture** . . . 90
4.17 **Voltage-controlled delay element[24]** . . . 91
4.18 **Voltage-controlled delay with 2 stages** . . . 91
4.19 Cell layout template . . . 92
4.20 Delay element layout template . . . 93
4.21 **layout of VCD and calibration oscillator** . . . 94
4.22 **Variable delay in function of control voltage** . . . 95
4.23 **Place of VCD and calibration oscillator** . . . 95

4.24 **Architecture of implemented built-in test circuit** . . . 95
4.25 **Layout of built-in test circuit** . . . 96
4.26 **VHDL model for clock generation**: (a) block diagram and (b) generation of a random variable with normal distribution using Box-Muller transform. . . . 96
4.27 **Histogram of clock errors between clk1 and clk2** . . . 97
4.28 **ASIC prototype error rate (T = 27 °C)** . . . 98
4.29 Simulated variable delay value versus control voltage at different temperatures: 99
4.30 **ASIC prototype error rate measurement (T = 100 °C)** . . . 100
4.31 Simulated variable delay value versus control voltage in different process corners: 100
4.32 **ASIC prototype error rate measurement (Corner SS)** . . . 101
4.33 **ASIC prototype error rate measurement (Corner FF)** . . . 101
4.34 Simulated variable delay value versus control voltage with power supply variation: 102
4.35 **ASIC prototype error rate measurement (Vdd=1.2 V + 5%)** . . . 103
4.36 **ASIC prototype error rate measurement (Vdd=1.2 V - 5%)** . . . 103
4.37 **The integration of proposed test circuit on the chip** . . . 104

5.1 Structure of the implemented clock network . . . 109
5.2 Repeating discrete ramp function in the DDFS . . . 111
5.3 Schematic diagram of the proposed FPGA implementation of the oscillator . . . 112
5.4 Conventional phase detector . . . 114
5.5 Block diagram of the node in a FPGA prototype with observation points . . . 116
5.6 Local clock signals together with reference . . . 117
5.7 Local clock signals together with reference . . . 118
5.8 Local clock signals around Node 10 together with reference and integer sum of the node errors (Total_Err) . . . 118
5.9 Local clock signals together with reference . . . 119
5.10 Local clock signals around Node 10 together with reference and integer sum of the node errors (Total_Err) . . . 119
5.11 Local clock signals together with reference . . . 120
5.12 Local clock signals around Node 10 together with reference and integer sum of the errors . . . 120
5.13 Local clock signals together with reference . . . 121
5.14 Unidirectional mode topology . . . 122
5.15 Clocks in unidirectional mode at steady state . . . 122
5.16 **Experiment principle diagram (Prototype-2)**: bidirectional configuration, initial frequencies of nodes are different . . . 123
5.17 **Histogram of phase errors between local clock signals and the reference in** 10×10 **prototype**: unidirectional . . . 124
5.18 **Histogram of phase errors between local clock signals and the reference in** 10×10 **prototype**: bidirectional . . . 125
5.19 Maximal value of phase error in function of distance to the reference clock . . . 126
5.20 RMS value of phase error in function of distance to the reference clock . . . 127

6.1 **Design hierarchy** . . . 130
6.2 **Design environment** . . . 131
6.3 **Top-down design flow** . . . 132
6.4 **Top-down verification flow** . . . 133
6.5 **Structure of a NODE**: one DCO, two PFDs and digital processing block (SPI and loop filter) . . . 134
6.6 **Layout of the block NODE**: one DCO, two PFDs and digital processing block (SPI and loop filter) . . . 135
6.7 **Post-layout simulation of one NODE** . . . 137
6.8 Preliminary floorplan of the test chip . . . 139
6.9 The connection sequence of the programmable blocks of the network . . . 141
6.10 Layout of the test chip of the clock network . . . 145
6.11 **Clock signals in the network in steady mode** . . . 146
6.12 **PFD outputs between NODE5-3 and its neighboring nodes:** A) programming mode; B) unidirectional mode; C) bidirectional mode . . . 147

7.1 **Interconnection and structure of the ADPLLs** . . . 150
7.2 **(a) Interconnection of nodes at the border of network**: ($\rightarrow$: unidirectional; $\leftrightarrow$, $-$: bidirectional) **(b) overflow channel of a swimming pool** . . . 154
7.3 **Proposed network topology** ($\rightarrow$: unidirectional; $\leftrightarrow$, $-$: bidirectional) . . . 155
7.4 **The surface of phase error between each local clock signal and reference:** (a) in frequency/phase acquisition mode; (b) a perturbation happened; (c) the propagation of phase error; (d) the stable state is re-established . . . 157
7.5 **Histogram of absolute phase errors at steady state:** (a) inner clock signals; (b) border clock signals . . . 157
7.6 **Transient response of perturbation:** (a) conventional 10×10 network (b) "Swimming pool"-like 10×10 network . . . 158
7.7 **Absolute phase errors of clock signals in proposed network with different parameters:** (a) overdamped; (b) underdamped . . . 159

8.1 **Conventional PLL structure** . . . 163
8.2 **Tierno PLL structure[30]** . . . 163
8.3 **Tierno PLL structure[60]** . . . 164
8.4 **Proposed architecture for PLL** . . . 166
8.5 **Sliding window algorithm** . . . 167
8.6 **Mean filter implementation** . . . 168
8.7 **State diagram of controller** . . . 169
8.8 **Implementation of memory in mean filter** . . . 169
8.9 **Chronograph of mean filter** . . . 170
8.10 **Data flow diagram in the mean filter** . . . 171
8.11 **CFA structure** . . . 171
8.12 **Transfer function of 3-bits PFD** . . . 172

8.13 Adaptive loop filter structure 173
8.14 **Simulation of conventional architecture** $Kp = 1, Ki = 15/2^{11}$ 174
8.15 **Simulation of new architecture with regular PI filter** $Kp = 1, Ki = 15/2^{11}$. 175
8.16 **Simulation of new architecture with adaptive filter** $Kp = 1$ **or** 3**,** $Ki = 15/2^{11}$ 175
8.17 PLL network topology 176
8.18 **PLL with sliding window** 177
8.19 Error entries of node 16 mutiplied by *kw* (reference code=theoretical code) . . . 179
8.20 Error entries of node 16 mutiplied by *kw* (reference code=theoretical code + 1) 179
8.21 Error entries of node 16 mutiplied by *kw* (reference code=theoretical code + 2) 179
8.22 Error entries of node 16 mutiplied by *kw* (reference code=theoretical code + 3) 180
8.23 Error entries of node 16 mutiplied by *kw* (reference code=theoretical code + 4) 180
8.24 Error entries of node 16 mutiplied by *kw* (static offset = 5) 180
8.25 Error entries of node 16 mutiplied by *kw* (dynamic offset -5 to 5, only the first phase from 0 to 90 us is displayed) 181

9.1 **Various 3-D clock distribution approaches within the test circuit of [38]**: (a) H-trees, (b) H-tree and local rings/meshes, (c) H-tree and global rings 188
9.2 **Proposed 3-D clock distribution approach using the network of ADPLL** . . 189

D.1 Functional diagram of Cyclone II DSP Development Board 237
D.2 Top view of Cyclone II DSP Development Board 237
D.3 FPGA prototyping platform 238

List of Tables

1 Parameters of FPGA and VLSI implementations xxxiii
2 Caractéristiques du réseau de PLLs tout numériques conçu xxxv

1.1 C-element truth table 14

2.1 **Characteristics of DCOs and maximum residual phase errors due to DCO quantization** 40
2.2 **Maximum residual phase errors due to PFD quantization** 40
2.3 1^{st} **network test chip characteristics summary** 46
2.4 2^{nd} **network test chip specification** 47

3.1 Values of τ_1 and τ_2 with different N_{on} 54
3.2 DCO chip performance summary 72

4.1 **Post-layout simulation results (T = 27** °C) 98

5.1 Parameters of the 1^{st} generation FPGA and VLSI implementations 115
5.2 Parameters of the 2^{nd} generation FPGA and VLSI implementations 115
5.3 **Minimum/Maximum values of phase errors between local clock signals and the reference in** 10×10 **prototype** (*ns*) 126
5.4 **RMS values of phase errors between local clock signals and the reference in** 10×10 **prototype** (*ns*) 126

6.1 Simulation parameters and conditions 136
6.2 Summary of IO pads 143
6.3 Summary of power supply pads 144

8.1 **Control table** 168
8.2 DCO chip performance summary 174
8.3 **Power consumption of two architectures** 175
8.4 Simulation procedure 178

Génération distribuée d'horloge pour SoCs synchrones

Cette thèse aborde le problème de la synchronisation globale de grand système sur puce (SoC), dans le cadre des technologies submicroniques profonds.

Ce travail de thèse est déployé dans le cadre d'un projet de recherche au laboratoire LIP6 sur les techniques de synchronisation du circuit. Il a commencé à 2007 et financé par deux subventions de l'ANR consécutifs (HODISS, 2008-20012 et HERODOTOS, 2011-2014). Le projet a porté sur l'étude d'une architecture particulière de génération d'horloge, basé sur un réseau d'oscillateurs couplés par un réseau de tout-numérique Phase Locked Loops. L'objectif du projet est le développement d'une nouvelle technique de synchronisation et de sa validation par la conception de deux prototypes de circuits intégrés.

Ma thèse est directement liée à cet objectif et le point de départ de mon projet était les résultats obtenus par les travaux antérieurs. Mon projet de thèse vise l'étude et la mise en œuvre de silicium d'un vaste réseau de ADPLLs (10 $\times$10), contenant 100 nœuds générant chacun une horloge pour le circuit numérique local. En comparaison avec l'étude effecturée précédemment (un réseau 4×4), la conception de blocs du réseau a été adapté aux contraintes liées à l'échelle du réseau. En plus, afin de valider les performances de la synchronisation d'horloge dans le circuit ASIC, nous avons conçu un circuit de mesure d'erreur d'horloge sur puce.

Le manuscript de thèse présente toutes les étapes de conception du prototype, en détaillant la méthodologie de conception, la modélisation et les techniques de prototypage utilisées.

Introduction: motivation de l'étude

Défis de génération d'une horloge globale dans les SoCs

Les progrès de la technologie CMOS ont conduit à une réduction exponentielle de la taille des circuits numériques et une augmentation du nombre de transistors par mm^2. Le SoC moderne peut être considéré comme des micro-réseaux permettant à différentes parties des

systèmes de travailler ensemble et de communiquer. Synchronisation de la communication devient un sujet de la plus haute importance de la recherche. Cette étude porte sur le problème de la génération d'horloge global et la distribution à l'intérieur de SoC complexe et large, de manière à permettre une communication complètement synchrones sur la puce. Elle est motivée par les inconvénients des approches de génération d'horloge classiques telles que l'arbre d'horloge dans le contexte des technologies CMOS submicroniques profonds.

Génération d'horloge traditionnelle dans des circuits complexes utilise des structures d'arbres ou grille [8, 5, 1]. Le *matching* entre les différents chemins d'horloge est le point clé de la conception du réseau d'horloge. Dans un grand circuit, un tel mis en *matching* global est difficile à réaliser. Le signal d'horloge global doit être acheminé sur toute la surface de la puce, alors que le délai correspondant doit être garantie avec une résolution picoseconde. Pour résoudre le problème de *mismatch*, la taille des buffers doit être augmentée pour être moins sensible aux erreurs de fabrication, ce qui rend cependant la solution très coûteuse, principalement en termes de consommation d'énergie.

Une autre façon de penser consiste dans la partition d'une grande puce dans les domaines d'horloge locale (domaines) [71, 42, 26, 41]. Ces domaines sont également connus comme des zones isochrones [4] ou zones synchrones d'horloge (SCA) [21] (cf. Fig. 1). Ces zones sont suffisamment petits que la distribution d'horloge à l'intérieur peut être réalisée sans difficulté par des techniques classiques. La taille de la zone est déterminée par le délai maximal de propagation à intérieur de la zone qui ne violerait pas les contraintes de temps du circuit. Ce délai dépend de la technologie utilisée et le routage des fils, où la zone n'a pas une dimension exacte. Un nombre empirique des portes / verrouillage à l'intérieur de la zone est d'environ 200-300 milliers.

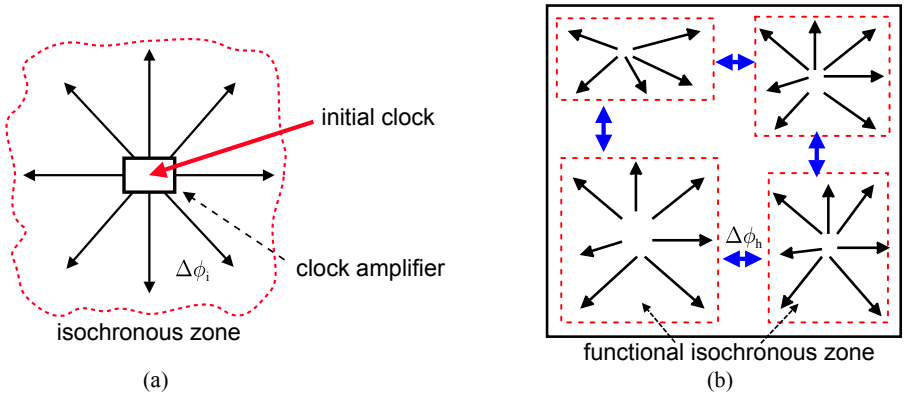


Figure 1: **Synchronisation des circuits numériques complexes**

La communication à l'intérieur du SCA est synchrone, donc le problème de la communication globale est réduite à la communication à travers les frontières des zones . La communication entre les blocs situés dans différentes SCA peut être synchrone ou asynchrone. Dans le premier cas, la puce est GSLS (globalement synchrone localement synchrone), dans ce dernier cas la puce est GALS(globalement asynchrone localement synchrone).

En raison des difficultés de synchronisation globale, les ingénieurs SoC orientent leur choix vers les GALS. Ceci est réalisé en utilisant des interfaces bi - synchrone (ex. FIFO) garantissant l'intégrité du signal au prix d'une complexité de conception et d' augmentation de la latence. Par ailleurs, dans les circuits asynchrones, la fiabilité est difficile à garantir, à l'étape de conception, pour deux raisons. Tout d'abord, la vérification exhaustive d'un système asynchrone est impossible, puisque le temps est continu. Deuxièmement, le risque de métastabilité peut être pire que d'habitude prévu, parce que les horloges de SCA ne sont pas totalement indépendant. En effet, ces horloges sont dérivées d'une horloge d'entrée unique et distribués aux différents modules. Leurs déphasages relatifs dépendent de paramètres changeantes rapides et lents comme le vieillissement (lent), la température (vitesse moyenne), et de la tension (rapide). Métastabilité peut se produire pour certaines valeurs de ces paramètres. Pendant une courte période de temps , les paramètres lents ne changent pas, ce qui entraîne la métastabilité de se reproduire de manière répétitive, augmentant le risque de défaillance du système.

Défis de caractériser l'erreur de phase entre deux signaux d'horloge

Une communication synchrone nécessite les horloges de l'émetteur et du récepteur être parfaitement synchronisés. Dans la pratique, une synchronisation parfaite est impossible: les événements d'horloge doit-être-simultanées sont séparés par des intervalles de temps (erreur de l'horloge). La prise de conscience de l'erreur d'horloge maximale est nécessaire pour l'établissement de budget de temps pour la communication et le traitement des données.

Dans un circuit synchrone, le signal d'horloge est distribué à différents endroits de la puce par une certaine méthode, ex. arbre d'horloge. Tous les signaux d'horloge doivent être synchronisées avec presque la même période moyenne T. L'incertitude de l'horloge ou erreur de phase (Δt_i) entre deux horloges clk1 et clk2 avec la même période moyenne T est définie comme clk2 i^{th} front montant (descendant) t_i^2 moins clk1 i^{th} front montant (descendant) t_i^1 (Fig. 4.1). Ici i est l'indice de cycle d'horloge. Dans cette étude, l'erreur de phase entre les horloges est censé être beaucoup plus petite que la période d'horloge moyenne (nominale), et aucune dépendance existe entre Δt_i. Ce sont des hypothèses raisonnables dans le contexte de l'horloge sur puce. Par conséquent, Δt_i peut être exprimée par les équations suivantes:

$$\Delta t_i = t_i^2 - t_i^1 \tag{1}$$

$$\|\Delta t_i\| \ll T \tag{2}$$

$\{\Delta t_i\}_{i\in\mathbb{N}}$ est un processus aléatoire à temps discret caractérisé par le temps en moyenne $S_{in} = \overline{\Delta t}$ appelé *inclinaison* et la composante dynamique de l'erreur de phase $\{\Delta t_i - S_{in}\}_{i\in\mathbb{N}}$. Ce processus est généralement considéré comme ergodique. Les valeurs d'une réalisation du

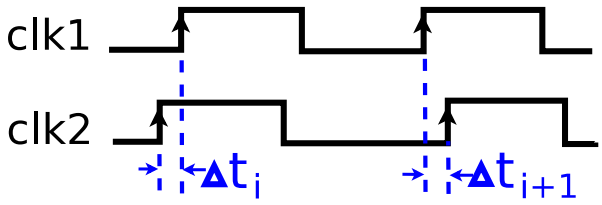


Figure 2: **Definition d'erreur d'horloge** Δt_i

processus $\{\Delta t_i\}_{i\in\mathbb{N}}$ sont caractérisés par une fonction de distribution, qui est aussi une fonction de densité de probabilité (PDF) puisque le processus est ergodique. Dans la pratique, une fonction de densité de probabilité d'une erreur d'horloge est une fonction définie sur un domaine limité à des valeurs non nulles minimales et maximales.

La mesure de l'erreur d'horloge peut être nécessaire dans beaucoup de cas. Les erreurs typiques d'horloge pour les horloges de gigahertz sont des dizaines de picosecondes ou moins. Une mesure hors puce nécessite la transmission de signaux d'horloge de la puce, introduisant ainsi des retards supplémentaires dont les valeurs sont difficiles à contrôler [68, 65, 65]. Le taux d'échantillonnage et le gain vertical des outils de mesure sont également des enjeux majeurs.

«On-chip» solutions sont généralement basées sur des techniques de mesure du temps à l'aide d'une chaîne de délai en cascade. Lorsque la valeur absolue de l'erreur d'horloge doit être connu, plusieurs travaux visant à une mesure directe de l'erreur d'horloge utilisent un convertisseur temps-numérique basée sur une chaîne de délai avec un grand nombre d'étapes (par exemple, 129 dans [13]). Les sorties des étages sont ensuite traitées par des circuits numériques à haute vitesse. Bien qu'un tel procédé fournit une valeur précise de l'erreur à chaque période, il est cher dans le sens de surface et d'énergie, puisque dans la plupart des cas, seulement des statistiques de l'erreur d'horloge sont suffisants (la moyenne, les valeurs minimale et maximale). En outre, le processus CMOS est très sensible à la température, donc une calibration fréquente avec une référence est nécessaire.

Réseau de PLLs tout numériques pour la génération d'horloge

Le générateur d'horloge proposé appartient à la famille des architectures multi-oscillateur sur la base d'un réseau d'oscillateurs couplés. Dans un tel schéma d'horloge, une puce est divisée en zones d'horloge locale, chacun d'eux ayant son propre générateur d'horloge (oscillateur) qui doit être synchronisé avec ses voisins dans le domaine de phase. Le but du réseau distribué de PLL est afin de synchroniser chaque oscillateur en phase et en fréquence. Dans un état stable, un tel réseau est une source d'horloges locales réparties entièrement synchrones.

L'architecture proposée par Pratt et Nguyen[39] est un réseau de maillage cartésien avec deux dimensions, dans lequel les nœuds sont les générateurs d'horloge locale et les arcs

représentent les liaisons de couplage entre les générateurs locaux. Chaque générateur local est liée uniquement à ses voisins immédiats cartésiennes. Une telle topologie nécessite les chemins les plus courts pour la transmission d'information - ce qui est un avantage principal d'une telle architecture comparé avec des méthodes de génération d'horloge centralisée.

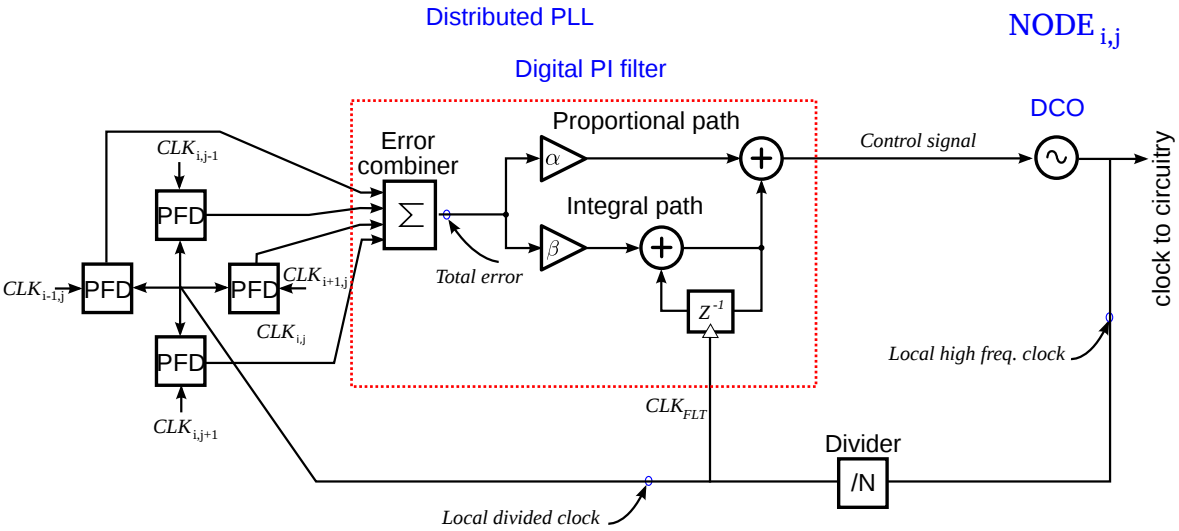


Figure 3: **La topologie du réseau d'horloge proposé et l'architecture du nœud**

Le couplage entre les oscillateurs est mis en œuvre dans le domaine de phase via *des comparateurs de phase*, Fig. 3(a). Chaque comparateur de phase fournit une mesure de l'erreur de phase entre les deux oscillateurs. Cette mesure est ensuite utilisée par le circuit de commande associé à l'oscillateur pour fournir un signal de commande forçant l'oscillateur pour synchroniser avec ses voisins. Le signal de contrôle détermine directement la fréquence de l'oscillateur - ce qui est un dérivé de la phase de l'oscillateur.

La nature analogique de ce système mis en place par Gutnik et Chandrakasan [15] est son principal inconvénient pour la génération d'horloge. Le générateur d'horloge est généralement intégré avec les blocs numériques qui utilisent le signal d'horloge. La performance d'une PLL analogique dans un environnement numérique peut être considérablement dégradé par les perturbations à cause de la commutation dans les circuits numériques. En outre, utilisant le réseau de PLL analogique pour la génération d'horloge rend la migration de la technologie plus difficile et réduit la portabilité de la conception globale du système sur puce.

Pour ces raisons, notre recherche porte sur l'étude du réseau de PLLs tout numérique (ADPLL). Dans un réseau d'ADPLL, un comparateur de phase analogique à générer un signal proportionnel à l'erreur de phase est remplacé par un comparateur de phase et fréquence (PFD) numérique, qui génère un code numérique proportionnel à l'erreur. Ce code est traité par un filtre de boucle numérique. Par la suite, le signal provenant de la sortie du filtre est utilisé pour commander un oscillateur commandé numériquement (DCO).

Basé sur les principes présentés ci-dessus, un réseau d'ADPLLs couplés a été étudié et

mis en œuvre au cours du projet HODISS. L'architecture est un réseau 4×4 avec 16 nœuds. La fonction numérique a fait l'objet d'études. Il a démontré la faisabilité de la mise en œuvre d'un tel réseau de PLL tout numérique.

La première puce de test est fabriquée et mesurée. Les résultats de test montrent que un générateur d'horloge numérique distribué est réalisable. Et les résultats de mesure correspondent bien à des études théoriques. Cependant, au cours du processus de conception et de test, nous trouvons quelques problèmes à résoudre et des points à améliorer:

1. *Preuve de l'évolutivité*. D'après les résultats de mesure du premier prototype on constate que, bien que l'erreur de phase est plus grand entre deux nœuds dans la distance qu'entre deux nœuds voisins, il ne s'accumule pas de façon linéaire en fonction de la distance. Mais puisqu'il n'y a que quatre nœuds sur la diagonale d'un réseau 4×4, le premier prototype ne peut pas bien démontrer la relation entre l'erreur de phase et la distance. Pour mieux observer cette relation et aussi d'étudier l'évolutivité du réseau ADPLL, un vaste réseau (ex. 10×10) est nécessaire. Une mise en œvre d'un plus grand réseau nécessite une optimisation des blocs existants, afin de minimiser la consommation d'énergie du circuit. Par exemple, le DCO mis en œuvre dans le premier prototype consomme 16 mA de courant d'alimentation, qui, multiplié par 100, les rend de 1,6 A: c'est prohibitif pour une mise en œuvre IC.
2. *Amélioration de la qualité de synchronisation*. La résolution du convertisseur temps-numérique (TDC) and le PFD du premier prototype est limitée à 30 ps en raison de la contrainte de sa structure et le retard minimum d'un porte en technologie 65 nm. Pour améliorer cette valeur, une autre architecture TDC doit être utilisé.
3. *Puce caractérisation: mesure de l'erreur de phase sur puce.* Comme expliqué précédemment, les résultats de mesure hors puce fournissent des informations pessimistes quant à l'erreur de phase. Une méthode sur puce est nécessaire pour caractériser l'erreur d'horloge et de suivi de la performance à grande SoC. Un exemple de ce type de circuit est le circuit «Skitter» utilisé dans les processeurs IBM[13].

Le paragraphe suivant présente un résumé de notre contribution.

Résumé de contribution

Le objectif principal de ce projet de thèse est la conception d'un réseau de ADPLL 10×10, complété par un outil de mesure de l'erreur de synchronisation sur puce. La conception est basée sur l'architecture du premier prototype de taille réduite, mais corrige des inconvénients de la puce mentionnée dans le dernier paragraphe.

Tout d'abord, une méthode visant à réduire l'erreur résiduelle dans l'état d'équilibre a été proposé. Il permet de choisir le pas de quantification de TDC et les paramètres appropriés

de DCO afin de atteindre une bonne performance avec faible effort de mise en œuvre. Sur la base de cette étude, nous avons conçu les blocs d'ADPLL pour le réseau d'horloge. En particulier, le pas de quantification optimale pour le DCO conçu est 20 ps, ce qui est inférieur au retard minimum d'une porte logique à la technologie utilisée. Pour atteindre cette valeur, un TDC avec une nouvelle structure est conçue.

Deuxièmement, en raison de la forte consommation d'énergie du premier prototype (186,2 mA), les 4×4 réseau était difficile d'être étendu à une dimension de 10×10. Pour réduire la consommation d'énergie du réseau de ADPLL, nous avons étudié la source de consommation d'énergie. Les blocs affamés d'énergie dans ce système sont le DCO et le filtre de boucle. Le DCO peut être optimisé en réduisant le nombre de cellules de réglage ainsi le nombre de pas de réglage (256 pas au lieu de 1024 pas). Moins de pas signifie moins de bits pour le mot de commande (8 bits au lieu de 10 bits), ce qui simplifie également le filtre de boucle.

En fait, la forte consommation d'énergie de filtre de boucle résulte principalement de l'opération arithmétique à haute vitesse. Dans mon projet de thèse, nous avons conçu deux types de filtres de boucle, qui non seulement réduisent la puissance, mais aussi d'accélérer la vitesse de correction d'erreur. Le premier filtre est basé sur l'architecture classique d'un filtre proportionnel-intégral (PI) avec un retard réduit au chemin proportionnel. Et la taille des opérateurs arithmétiques est plus petite. Le second filtre est de séparer l'acquisition de fréquence et le suivi de phase. A l'état stationnaire, seulment quelques bits de poids faible du mot de contrôle sont mises à jour pour le suivi de phase. Il n'est pas nécessaire de recalculer l'ensemble du mot de contrôle de chaque cycle. Les deux architectures ont été conçus au cours du projet, mais seulement la première a été mis en œuvre dans le prototype conçu du réseau. Ce choix est motivé par un risque élevé qui serait présente si une nouvelle architecture d'un filtre a été choisi.

Ces blocs conçus sont assemblés dans un réseau 10×10. Cette architecture est mise en œuvre dans un premier temps sur FPGA puis sur silicium. Topologies différentes sont étudiées sur la base de cette architecture.

Pour des raisons expliquées précédemment, un circuit de mesure d'erreur d'horloge sur puce est nécessaire pour la caractérisation de l'erreur de phase entre deux signaux d'horloge dans le réseau (générée par deux nœuds, soit à côté de l'autre ou en éloignement). Le circuit conçu dans ce projet de thèse est basée sur une méthode indirecte facile à mettre en œuvre: au lieu de mesurer la valeur de l'erreur de phase de chaque cycle, il permet de calculer les valeurs minimum/maximum et moyenne d'erreur pendant une période. En utilisant ce circuit, une caractérisation précise de l'erreur de phase sur puce est possible, donc il a une grande importance dans la preuve de l'évolutivité du réseau d'ADPLL. Il a été conçu comme un IP analogique et placé à quatre emplacements différents de la puce.

Conception des composants

Convertisseur temps-numérique (TDC)

En fait, la résolution TDC de la première puce d'essai est limitée à 30 ps à cause de la contrainte de sa structure et le retard minimum de porte logique dans la technologie de 65 nm. Pour améliorer cette valeur sans évoluer la technologie, une autre architecture doit être appliqué. L'idée de base consiste à utiliser la valeur différentielle des deux unités de délai pour atteindre une valeur inférieure à la chacun des unités de délai (technique Vernier).

Comme montré dans la Fig. 4, l'entrée de la chaîne de délai lente (τ_1) est un signal appelé *START*, dont le front descendant signifie le début de l'intervalle entre deux signaux d'horloge dans la comparaison. L'autre chaîne de délai (τ_2, $\tau_2 < \tau_1$) a une entrée *STOP*, dont le front descendant marque la fin de l'intervalle.

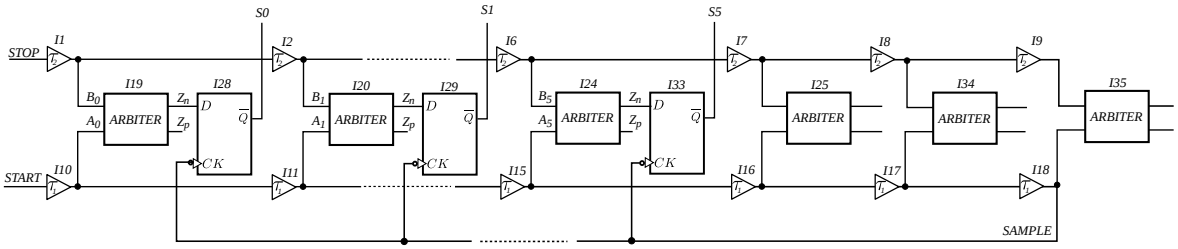


Figure 4: **Schéma de principe du convertisseur temps-numérique**

Dans chaque étage, les deux signaux retardés sont comparés en utilisant un arbitre. Comme les événements qui arrivent sont définis par les fronts descendants de *START* et *STOP*, si à un certain étage i, le front descendant retardé de *START* arrive à A_i encore plus tôt que l'événement arrive à b_i, A_i est égal à '0 ' et b_i est égal à '1'. Par conséquent, Z_p est égale à '1' et Z_n est égal à 0. Comme τ_1 est plus grande que τ_2, après chaque étage, l'intervalle de temps entre deux signaux est réduit par $\tau_{TDC} = \tau_1 - \tau_2$. La position de la ligne de délai, à laquelle le signal *STOP* retardé rattrape le signal *START* retardé définit la différence de temps entre les signaux d'origine avec une résolution de τ_{TDC}. Les registres produisent l'instantané de l'état de la chaine de délai au moment où l'événement *SAMPLE* arrive, donc contiennent un code thermomètre représentant l'intervalle à mesurer.

A l'aide de cette technique, nous pouvons obtenir une résolution de 20ps, ce qui est inférieur au retard minimum d'un délai en technologie CMOS 065 (30 ps). La résolution peut être encore améliorée, mais dans ce cas, l'erreur de phase résiduelle à l'état d'équilibre ne peut pas être réduit davantage. Ceci est prouvé par l'étude théorique et les résultats de simulation. Par conséquent, 20 ps est la valeur optimale pour ce système.

Filtre numérique

La puissance de la première puce de test est principalement consommé par DCO et le filtre de boucle. L'objectif de la conception du prototype n'atteint pas une grande dynamique de

réglage de la fréquence, donc le DCO peut être optimisée par la réduction du nombre de cellules de réglage ainsi les étapes de réglage (256 pas au lieu de 1024 pas). Moins d'étapes signifie moins de bits pour le mot de commande (8 bits au lieu de 10 bits), ce qui simplifie également le filtre de boucle peut être simplifié. En fait, la forte consommation d'énergie du filtre de boucle résulte principalement de son opération arithmétique à grande vitesse. En plus de réduire le mot de commande de sortie à 8 bits, il existe deux autres méthodes. La première méthode consiste à réduire le nombre de bits réservés aux cœfficients du filtre, ainsi la taille des opérateurs. Le schéma est affiché sur la Fig. 5. Le nouveau bloc de contrôle de boucle comporte quatre entrées permettant de recevoir au plus quatre entrées binaires de 4 bits des mots signés et génère un mot de 8 bits non signé pour le contrôle DCO (la sortie de l'additionneur ADD5). Ce signal est codé à A, B et C, qui sont appliquées à l'entrée de la DCO.

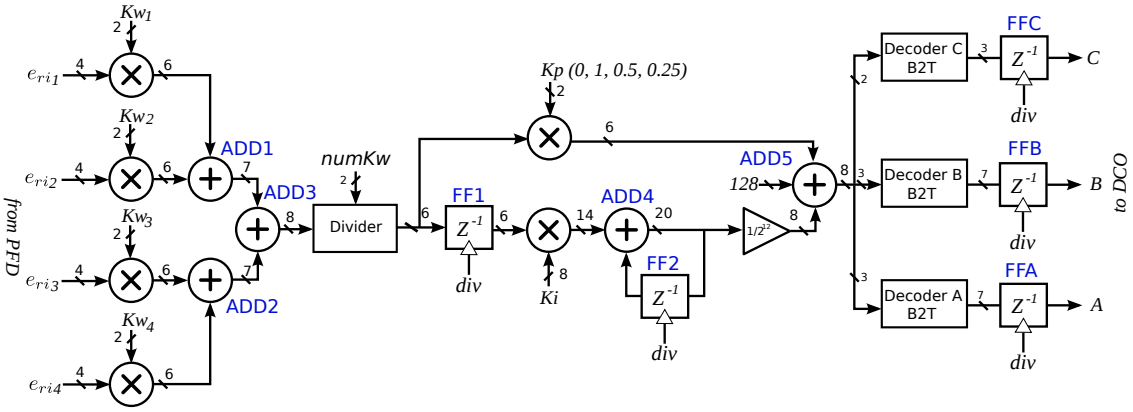


Figure 5: **Filtre de boucle pour le traitement du signal d'erreur**: quatre contrôleurs de gain d'entrée suivi par l'addition à quatre entrées, filtre PI et trois décodeurs B2T.

La deuxième méthode consiste à séparer l'acquisition de fréquence à partir de la poursuite de phase. La raison est qu'à l'état d'équilibre, seulement quelques bits de poids faible du mot de commande sont mis à jour pour le suivi de phase. Il n'est pas nécessaire de recalculer l'ensemble du mot de commande de chaque cycle. Cette architecture de PLL est montré dans Fig. 6. Un indicateur de la fréquence de référence (RFI), en dehors du circuit principal, donne un code ($code_{ref}$) correspondant à la fréquence d'horloge de référence. Ce code est ensuite transmis aux PLLs dans le réseau de distribution d'horloge. Poursuite de phase est réalisée par un PFD à 3 bits et un filtre de 6 bits signé générant un code de 6 bits en fonction de la différence de phase entre deux horloges. La poursuite de fréquence est réalisé par un filtre moyen, qui reçoit un flux de codes à 6 bits et calcule la moyenne des huit les plus récemment reçus. Le signe de la valeur moyenne ($+1$, -1 ou 0) estime le rapport de fréquence entre deux horloges, et est utilisé par le bloc du réglage grossier de la fréquence (*CFA*) pour le réglage du code. Le code de sortie de *CFA* plus la sortie du filtre forme le mot de commande du DCO.

L'avantage de cette structure est que le bloc *CFA* peut toujours mettre à jour la fréquence grossier (fc) de la DCO en fonction de la relation référence-l'horloge locale, et cet ajuste-

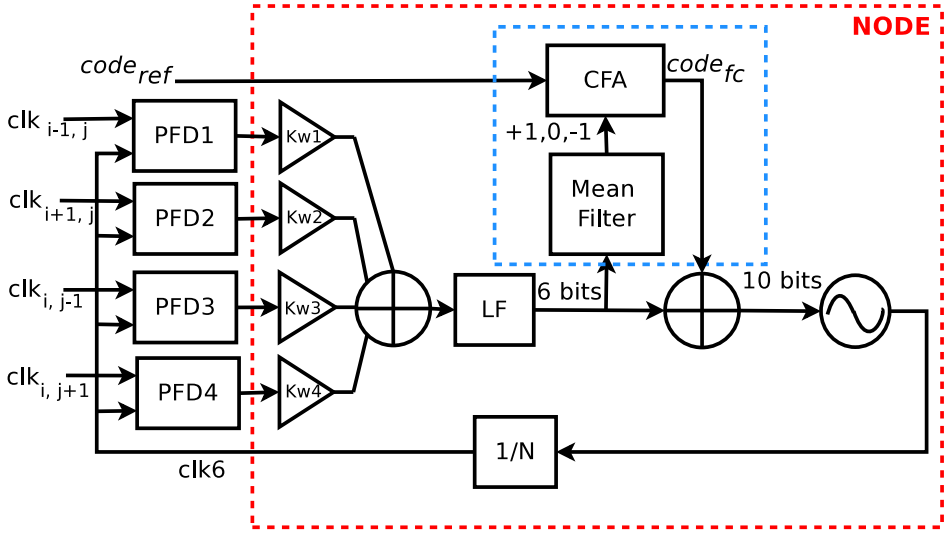


Figure 6: **Proposed architecture for PLL**

ment est effectué à une fréquence inférieure à la fréquence d'échantillonnage du système (*div*). PFD et filtre, travaillant à la fréquence *div*, réglent la fréquence de DCO autour de fc. Ce processus de réglage corrige l'erreur de phase et en même temps travaille en collaboration avec *CFA* pour assurer que la fréquence de référence est toujours dans l'intervalle de reglage. Comme l'intervalle est relativement faible, un grand nombre de bits pour PFD et le filtre n'est plus nécessaire.

Le réseau d'ADPLLs tour numériques réalisé

La puce de prototypage a été conçu et mis en œuvre pendant ma thèse. Il possède une topologie étendue par rapport à la première. Il a une dimension de 10 x 10 fournissant des signaux d'horloge pour autant de 100 domaines d'horloge synchrones locales (de SCA1-1 à SCA10-10 dans la Fig. 7).

Dans Fig. 7 nous pouvons observer le bloc FO (filtre + oscillateur) dans chaque SCA, la génération du signal d'horloge pour la zone isochrone locale. Un PFD à la frontière entre deux zones voisines compare deux signaux d'horloge et renvoie l'erreur de phase entre eux avec la même valeur absolue mais de signe opposé (e and $\overline{e}$). Toutefois, en raison de la consommation d'énergie élevée (186.2 mA), 4 le réseau 4×4 est difficile d'être étendu à une dimension de 10×10.

Avant la fabrication de la puce de test utilisant ces blocs améliorés, un prototype FPGA d'un réseau de 10×10 de ADPLLs. Ce prototype a deux objectifs: d'abord, il permet de valider la fonctionnalité d'un tel grand réseau avant la mise en œuvre en silicium; D'autre part, avec ce prototype, on peut étudier la relation entre la valeur d'erreur de phase et de la distance. Ce prototype a la même dimension et topologie que la puce de test (Fig. 7). Les principales limites de prototypage FPGA concernent l'impossibilité de mettre en œuvre correctement (a) les blocs de signaux mixtes: le PFD et le DCO et (b) la fréquence de

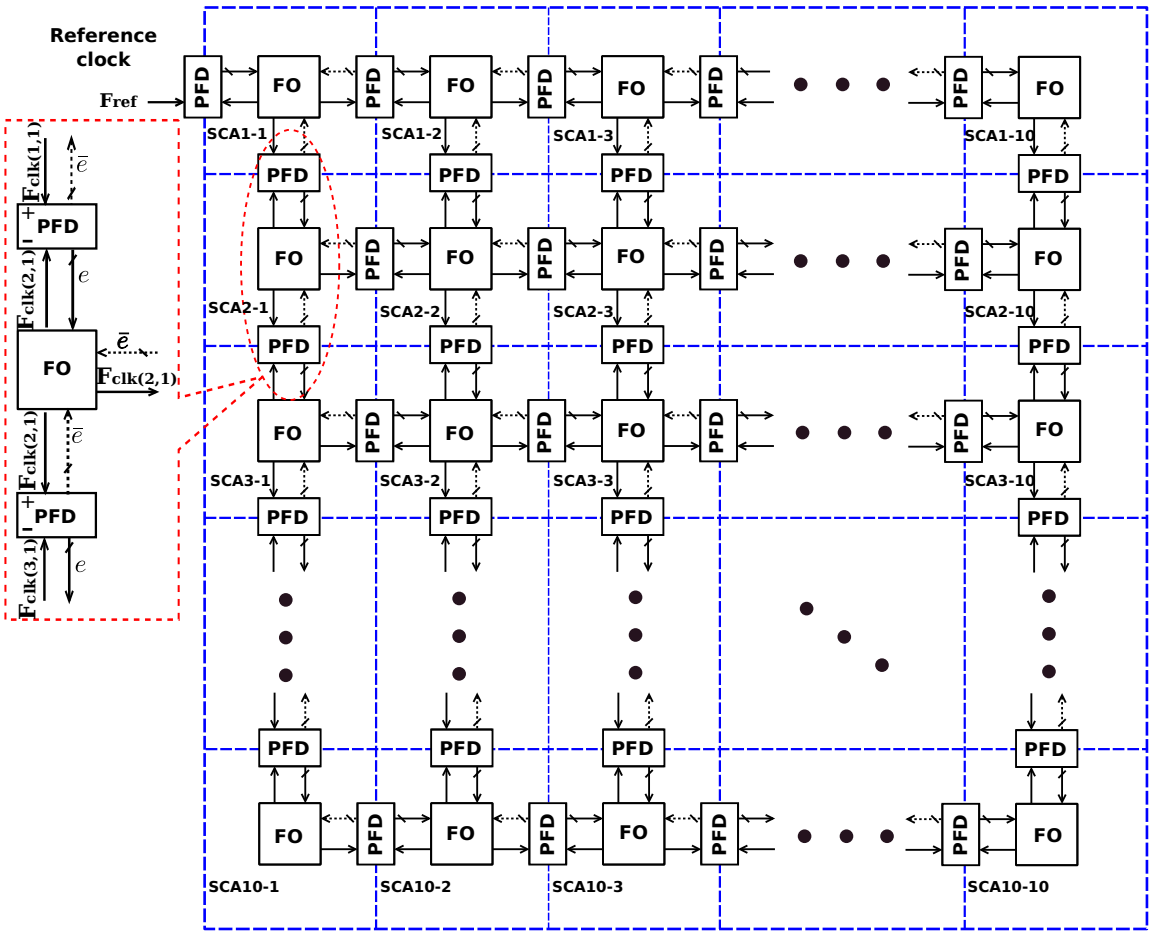


Figure 7: **Clocking network architecture**

Table 1: **Les paramètres de la mise en œuvre de FPGA**

Parameter	FPGA
Fréquence de sortie nominale	77.93 kHz
Pas de fréquence du DCO	97.05 Hz
Plage de réglage de la fréquence de sortie	67.28 kHz∼92.73 kHz
Résolution de TDC	68.065 ns

fonctionnement élevée. Un modèle numérique commun des modules basé sur l'étage de temps est un chronomètre; Pour le deuxième problème, il suffit de réduire l'ensemble des paramètres de fréquence avec le même facteur d'échelle. Tab. 1 résume les paramètres du TDC et du DCO mis en œuvre pour le prototype FPGA du réseau 10x10 d'ADPLL.

Fig. 8 montre les dix signaux d'horloge dans les nœuds diagonale du réseau. Dans cette expérience, les nœuds du réseau ont diverses fréquences initiales, mais avec l'aide de la configuration dynamique, tous les signaux d'horloge locaux ont la même fréquence et sont bien alignées en phase, ce qui démontre une bonne performance de l'architecture proposée dans un grand réseau.

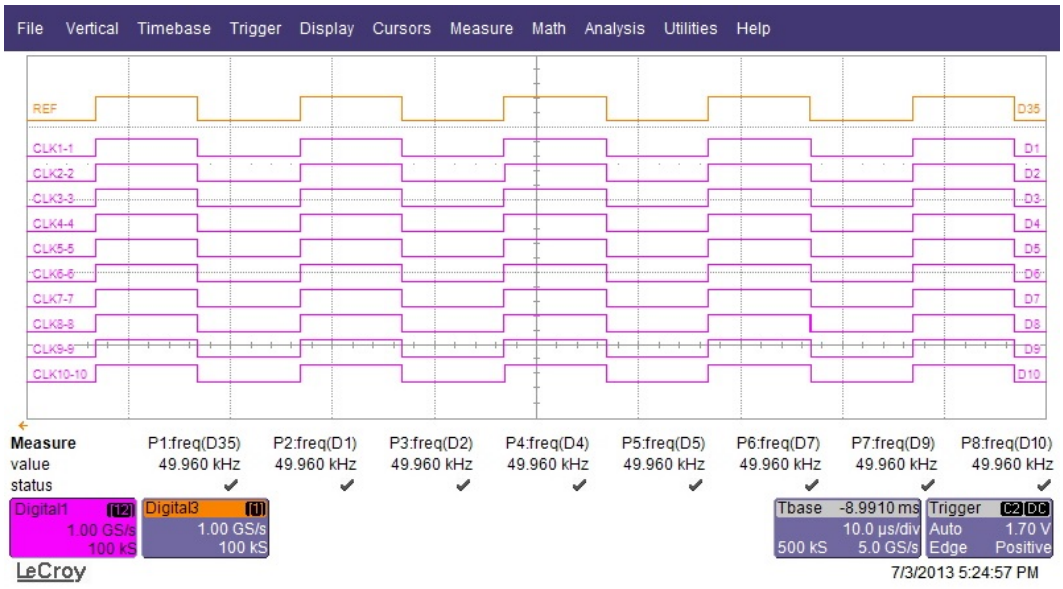


Figure 8: **Horloges signaux locaux avec la référence**: configuration bidirectionnelle, différentes fréquences initiales des nœuds

Afin de valider le fait que dans un tel réseau d'ADPLL couplés, les erreurs de phase ne sont pas accumulés comme dans un arbre d'horloge, on a mesuré sur chaque cycle d'horloge pendant certain temps, la différence de phase entre les signaux d'horloge locaux dans les nœuds diagonaux et l'horloge de référence. Ensuite, nous pouvons tracer une courbe d'erreur de phase maximale en fonction de la distance à l'horloge de référence. Il faut noter que la distance mentionnée ici n'est pas la distance physique en microns, mais la distance de Manhattan l'information de phase de référence doit parcourir avant d'arriver à un nœud local.

De Fig. 9 nous pouvons observer que en mode unidirectionnel, l'erreur de phase s'accumule comme l'information de phase de référence se déplace plus loin. En mode bidirectionnel, qui est le mode à laquelle le réseau fonctionne à l'état d'équilibre, les erreurs de phase entre tous les nœuds du réseau et la référence sont bien limitées à $\pm$ 3 fois le pas de quantification du PFD.

le prototype AISC avec la même dimension que le prototype FPGA a été conçue dans une technologie CMOS 65 nm, en utilisant le PFD, DCO et le bloc numérique de traitement d'erreur présentés précédemment. La conception de puces applique une stratégie de réutilisation de l'IP et un flot de conception à signal mixte.

L'assemblage des blocs de réseau d'horloge a été fait dans un flot standard de conception numérique avec l'aide d'outils de CAO. La disposition réelle de la puce est donné dans Fig. 10. La taille de la puce est $2734 \times 2756,8\ \mu m^2$ dans laquelle le cœur du réseau occupe $1805 \times 1771\ \mu m^2$. Les paramètres du réseau sont résumés dans la Tab. 2.

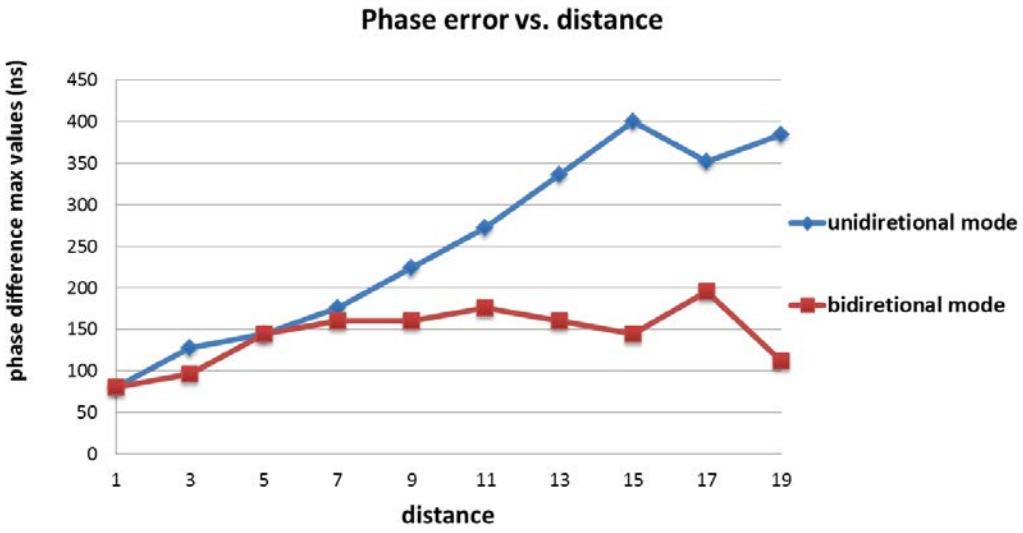


Figure 9: **La valeur maximale de l'erreur de phase en fonction de la distance par rapport à l'horloge de référence**

Table 2: **Caractéristiques du réseau de PLLs tout numériques conçu**

Paramètre	***Valeur***
Fréquence de sortie nominale F_{CLK}	1,036 GHz
Plage de réglage de la fréquence de sortie	903∼1161 MHz
Pas de fréquence du DCO $\Delta\omega/(2\pi)$	1,01 MHz
Erreur de timing	$<$ 40 ps
Taux de convergence	$\approx$ 5 MHz/μs
Tension d'alimentation	1,2 V
Consommation	600 mW @ F_{clk} = 800 MHz
Taille des SCA	$\sim 0,022 mm^2$
Taille de puce	$\sim 7,54 mm^2$

Caractérisation sur puce de l'erreur de phase entre deux signaux d'horloge

Comme présenté précédemment, il est difficile de mesurer l'erreur de phase entre les signaux d'horloge à haute fréquence. Dans cette étude, nous avons étudié une stratégie de test pour la caractérisation des statistiques d'erreur d'horloge entre deux domaines d'horloge dans les systèmes d'horloge à grande vitesse (gigahertz et plus). Le procédé permet une mesure indirecte (pas basé sur la mesure de l'intervalle de temps) de la distribution de l'erreur d'horloge par l'observation de l'intégrité d'une séquence périodique transmis entre deux domaines d'horloge. La méthode est compatible avec l'implémentation sur puce, et la lecture des résultats aux signaux hors puce est cadencée à faible taux. La stratégie vise à la résolution de picosecondes sans calibration complexe.

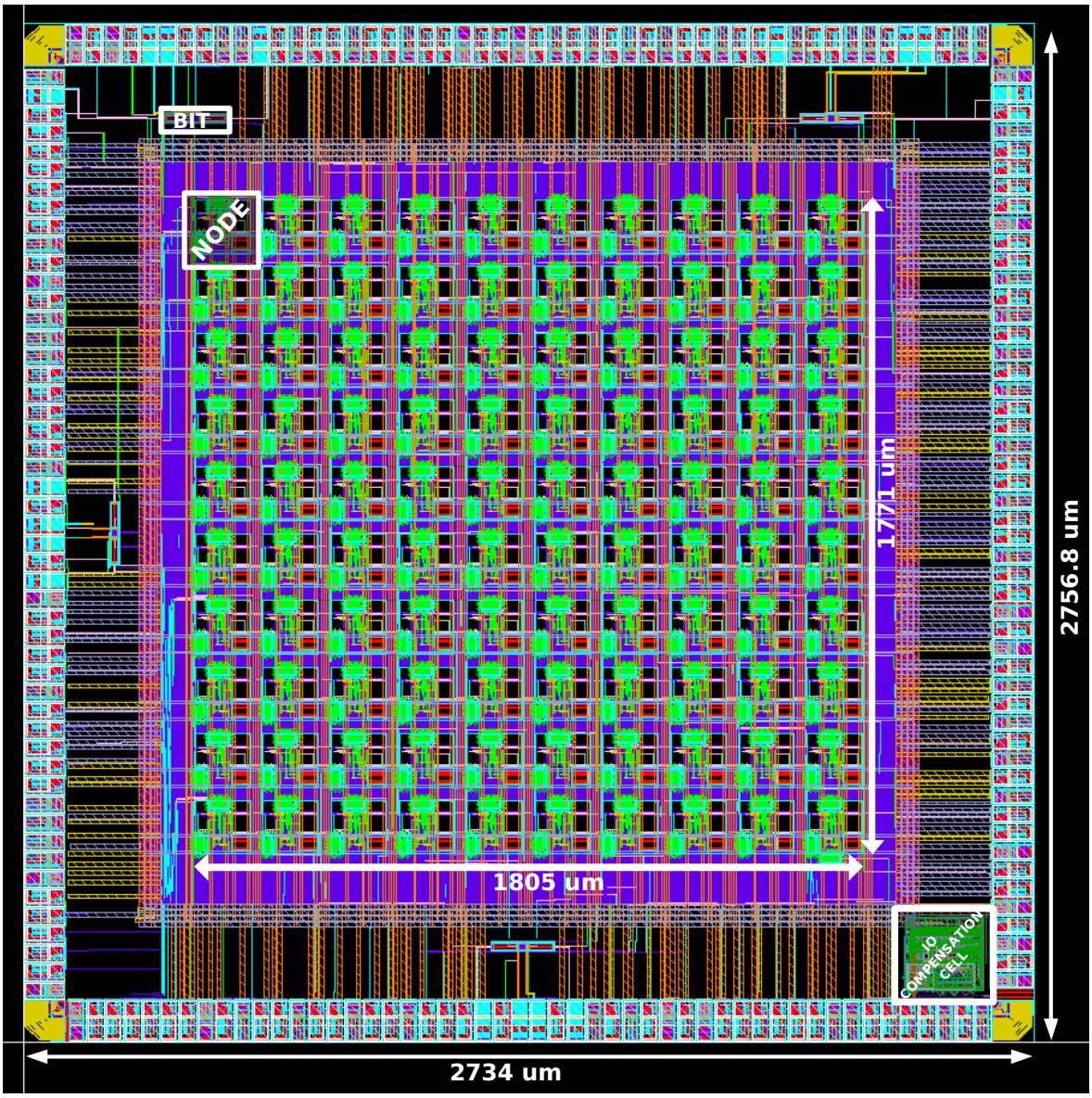


Figure 10: **Layout de la puce de test du réseau d'horloge**

Le circuit de mesure est mis en œuvre comme représenté sur la Fig. 12. Une séquence binaire «... 1010 ...» est généré sur la puce dans le domaine de clk1 par une bascule de type D synchronisée avec l'horloge CLK1 (Fig. 12). La séquence binaire est envoyé au domaine d'horloge clk2 après un délai contrôlable (Δ). Ce délai est équivalent à un *skew* supplémentaire entre les deux horloges. En faisant varier le délai Δ, nous changeons le *skew* effectif, par conséquent, le taux d'erreur (ER) mesuré, qui est maintenant une fonction de Δ.

De l'autre côté, l'introduction du délai Δ (Fig. 12) modifie le *skew* entre les horloges, et donc la fonction de densité de probabilité (PDF) effective de la distribution des erreurs d'horloge, par le *mapping* de $\Delta t \rightarrow \Delta t + \Delta$. Nous pouvons trouver une relation mathématique entre l'ER et la PDF d'origine de l'erreur d'horloge par l'observation de deux cycles d'horloge voisins. Comme illustré dans Fig. 11, le délai $\Delta = \Delta t_B$ ou $\Delta = \Delta T_C$ déplace la PDF d'origine à droite/à gauche jusqu'à $a = 0$ ou $a = 1$, où $a(\Delta) = P(\Delta t_i < 0)$ est la probabilité

que l'erreur de phase du cycle i^e est négative. Il a été prouvé que Δ_B et Δ_C correspondent respectivement aux valeurs de Δtd aux points B et C de PDF d'origine (Fig. 11(a)), qui sont les valeurs min/max d'erreur d'horloge.

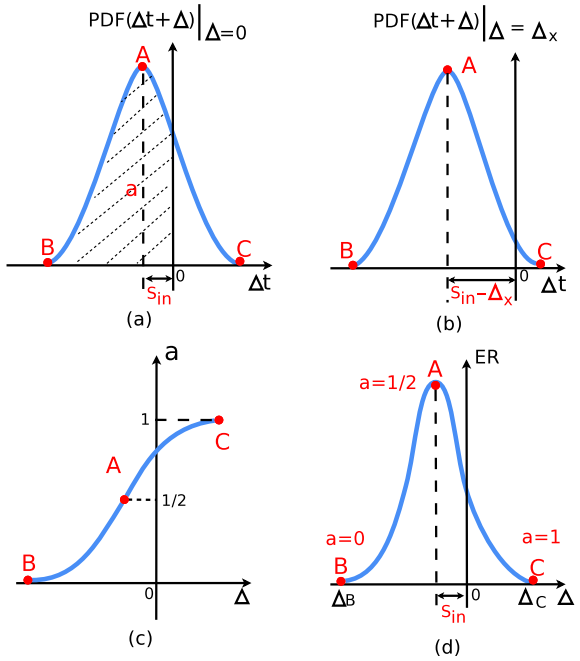


Figure 11: **Analyse théorique**: (a) PDF d'origine de l'incertitude d'horloge; (b) PDF avec un décalage $\Delta = \Delta_x$; (c) a vs. Δ; (d) ER vs. Δ

Pour obtenir l'ER, la séquence de données est échantillonnée dans le domaine de clk2. Pour éviter métastabilité dans le bascule[18], un registre à décalage avec 4-étage échantillonne le signal d'entrée (D2) du domaine de clk2. Une porte NXOR avec 2 entrées détecte les erreurs de transmission en comparant les sorties R3 et R5; chaque événement de détection est alors prise en compte par le compteur de n bits C1 cadencée par clk2. Le rôle du compteur C2 est de générer un événement de tous les $2^n - 1$ cycles de CLK2: ceci fournit un intervalle de temps pendant lequel C1 compte les erreurs. La valeur de C1 est écrit dans le registre de sortie Rs quand C2 déborde. Les registres Rs stocke la valeur signal de Nerr. Il représente le nombre d'erreurs lors de $2^n - 1$ cycles de CLK2 et peut être facilement transmis hors de la puce, car l'affichage est à une fréquence 2^n inférieure à la fréquence d'horloge.

L'idée a d'abord été validé par un prototype discret avec des fréquences à échelle réduite, puis un prototype avec la fréquence élevée sur puce a été conçu en utilisant la technologie CMOS 65 nm.

Conclusion

La génération et la distribution d'horloge sont des techniques importantes dans la domaine des circuits VLSI. La qualité du signal d'horloge a un grand impact sur les performances du circuit et de la consommation d'énergie. De nos jours, une grande variété de solutions

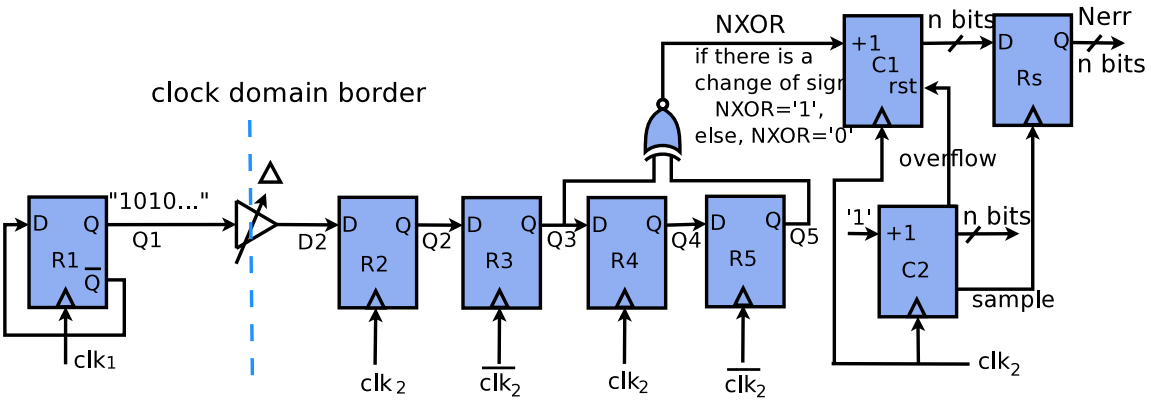


Figure 12: **Architecture of test circuit**

d'horloge comme l'arbre d'horloge, le maillage hybride d'horloge, la communication asynchrone sont utilisées. Ces techniques présentent une limite supérieure de la taille de circuit qui peut être synchronisé. Dans les technologies submicroniques profonds, c'est difficile de générer d'une horloge globale utilisable dans une grande puce par ces techniques classiques d'horloge.

L'objectif de cette recherche est une étude d'une solution alternatif de synchronisation, qui utilise un réseau des oscillateurs couplés par PLL tout numérique(ADPLL). Un tel système est appelé «réseau d'ADPLL». Ce projet de doctorat inclus une étude théorique, la modélisation, prototypage CMOS et FPGA des réseaux d'ADPLL. En comparaison avec les études précédentes, qui ont montré la possibilité principale de génération d'horloge par un réseau d'ADPLL, cette étude a démontré une faisabilité d'un *grand* réseau d'ADPLL, et résolu des problèmes spécifiques liés à la grande taille du réseau. Ce rapport de thèse décrit en détail la conception d'une puce contenant un réseau cartésien de 10x10 nœuds, chacun d'eux générant une horloge locale synchronisée en phase et en fréquence avec les horloges voisines. La modélisation du réseau à différents niveaux d'abstraction était une partie importante du projet. Le réseau d'ADPLL est un système complexe: seulement son modèle réduit peut être décrit par des équations, et des effets réalistes ne peut être prise en compte qu'avec une modélisation numérique. Cependant, nous avons proposé un modèle réduit utilisé pour la conception, se rapportant aux paramètres de réseau à la qualité de la synchronisation des signaux d'horloge locaux.

La conception d'un prototype VLSI du réseau utilisée était basée sur l'étude précédemment réalisée dans le projet HODISS, où un réseau de prototype de faible taille a été mis en œuvre. Cependant, dans cette thèse, nous avons adapté le modèle de plusieurs blocs du réseau aux contraintes liées à la taille du réseau, et nous avons modifié le détecteur de phase de manière à améliorer la résolution (2O ps au lieu de 32 ps dans le travail précédent) . Pour réaliser cette résolution, l'architecture Vernier a été utilisé. Le bloc DCO a été simplifié (la taille et la consommation d'énergie réduites). Nous avons conçu deux versions du filtre: une architecture nouvelle avec l'optimisation de la vitesse de convergence et la consommation de

la puissance, et l'autre étant une version simplifiée de l'architecture utilisée dans le premier prototype de faible taille de réseau. La puce contenant un 10×10 réseau d'ADPLL est en cours de fabrication en technologie 65 nm.

Manycore circuits sont probablement des applications potentielles les plus importantes de la solution d'horloge proposée. Pour de tels systèmes, la question de l'évolutivité est primordiale: comment la performance du système change si la taille du système augmente? Pour notre solution de synchronisation, la question peut être formulée comme suit: comment la qualité de synchronisation (l'erreur de l'horloge) changer avec la taille du réseau (le nombre de nœuds)? Notre étude fournit la réponse suivante à cette question:

- Si l'erreur de phase entre les horloges voisines est considéré, la réponse est «l'erreur d'horloge exprimée en unités de temps ne change pas lorsque la taille du réseau augmente» . L'erreur d'horloge entre les nœuds voisins ne dépasse pas deux étapes de résolution de la mesure d'erreur de phase. La résolution de la mesure d'erreur de phase s'améliore à la réduction de la taille des transistors CMOS (de la même manière que les retards de portes numériques).

- Si l'erreur de phase entre deux nœuds du réseau est pris en compte: on a montré que, bien que l'erreur de phase augmente avec la distance de Manhattan entre les nœuds (de façon équivalente, avec la taille du réseau), une saturation est observée, et l'erreur est limitée à trois étapes de la résolution de mesure d'erreur de phase.

Ces faits ont été vérifiés expérimentalement, sur un réseau de 10x10 nœuds mis en œuvre sur une plate-forme FPGA; limitations matérielles nous ont empêchés de test sur des réseaux de taille plus grande.

L'un des avantages principaux du réseau d'oscillateurs couplés par ADPLL est la possibilité de reconfigurer dynamiquement sa topologie, le connectivité, les paramètres de blocs de traitement, etc. Ceci permet une mise en œuvre des comportements différents du réseau, avec la même plate-forme matérielle. Cette propriété a été utilisée avec succès pour la sélection du mode synchronisé souhaitable. En outre, nous avons exploré une technique originale inspirée de l'hydrodynamique, ce qui permet à l'élimination des vagues qui pourraient apparaître dans le réseau debout. En configurant le bord du réseau d'une façon différente du noyau du réseau, le bord du réseau peut être considéré comme un anneau indépendant et synchrone. Les nœuds qui composent cet anneau excitent le noyau interne et absorbent les ondes d'erreur qui reviennent du noyau, comme les canaux d'évacuation d'une piscine. Cette «piscine» topologie a été étudié et conçu. Son avantage dans la prévention de la propagation d'onde et de réflexion d'erreurs a été démontrée par les résultats de la simulation.

La mesure de l'erreur d'horloge peut être nécessaire dans de nombreux cas: pour la caractérisation, ou pour l'auto-test du système d'horloge. Comme le signal d'horloge est continue dans le temps et il est très sensible au bruit, La mesure hors puce n'est pas appro-

priée pour les signaux de haute fréquence. Dans ce projet, nous avons conçu un circuit de mesure d'erreur d'horloge sur puce, qui mesure les statistiques d'erreur d'horloge entre deux domaines d'horloge dans les systèmes d'horloge à grande vitesse (gigahertz et plus). Le résultat de la mesure peut être lu hors puce à faible taux. La stratégie vise à la résolution des picosecondes sans calibration complexe. Après validation de la technique sur un prototype discret, un bloc de circuit intégré a été conçu en utilisant la technologie de 65 nm. Cette IP avec signal mixte a été utilisé dans la puce contenant un grand réseau de ADPLL, pour la mesure des erreurs entre des horloges dans quatre endroits du réseau.

Le circuit de génération et de distribution d'horloge, avec le circuit de test intégré constitue un système de synchronisation complet, qui propose une alternative aux techniques d'horloge classiques et existants pour les grands systèmes sur puce.

Perspectives et travail futur

La puce de test est toujours en cours de fabrication lorsque ce document est écrit. Par conséquent, le premier travail à l'avenir devrait être tester la puce et analyser les résultats de mesure. Si les résultats des tests sont satisfaisants, la prochaine étape sera l'intégration du système d'horloge proposé dans un SoC multiprocesseur réel.

En outre, au cours de la recherche, on a trouvé certaines difficultés ou des points d'amélioration par rapport à la modélisation d'un tel système non-linéaire, la robustesse de distribution d'horloge et le déploiement dans des circuits intégrés 3D.

Modélisation d'ADPLL pour l'étude de l'erreur de phase résiduelle dans l'état d'équilibre

Un défi dans l'analyse du comportement de correction d'erreur de phase d'un ADPLL ou un réseau d'ADPLLs consiste en les fonctions non-linéaires. La non-linéarité apparaît pour les raisons suivantes:

- La caractéristique du détecteur de phase est mod 2π.
- L'interfaces analogique/numériques entre le filtre numérique et le DCO, et entre le DCO et le PFD.
- L'échantillonnage de la bascule dans le filtre est commandé par la sortie du DCO: un auto-échantillonnage

Les techniques les plus courantes et les plus efficaces sont ceux qui proviennent de la théorie du contrôle pour les systèmes LTI. Toutefois, la suppression de quantification conduit aux prévisions inexactes pour les erreurs résiduelles (c'est à dire quand un réseau est très proche de l'état synchronisé). Par exemple, un modèle LTI continu d'un PLL à deux intégrateurs résiduel prédit l'erreur de phase nulle. Cependant, dans un réseau de ADPLL réaliste, l'erreur résiduelle est au moins égale au pas de quantification du PFD, et, est un processus stochastique avec une forte corrélation. Le cas où le système reste la plupart du temps aux petites erreurs de phase est «nominale» pour un réseau d'ADPLL et, par conséquent, la défaillance du modèle LTI pour décrire ce mode de fonctionnement est un inconvénient sérieux.

Exploration de la propriété de tolérance de faute

Un problème très grave avec des arbres d'horloge ou d'autres approches centralisées consiste à le manque de robustesse. Si en raison de certains défauts de fabrication, un fil interne ou un buffer dans un arbre d'horloge se brise, l'ensemble de la sous-arbre du point cassé cesseront de fonctionner correctement. Le système d'horloge proposé, une approche distribuée, c'est mieux dans ce sens. Mais comme l'ensemble du réseau est couplé, si un lien entre deux domaines d'horloge est cassé ou un oscillateur fonctionne anormalement, les nœuds voisins et

même l'ensemble du réseau seront également touchés. Cependant, étant donné que les PFDs observent toujours la fonction de génération d'horloge locale, et le système est numérique, il est possible de développer un mécanisme pour détecter la position de défaut et pour contourner le point de défaut automatiquement.

En général, il existe deux types de défauts dans le réseau: un lien cassé et un nœud défaillant. Si un lien se brise, le PFD ne peut ainsi pas détecter l'erreur de phase entre les deux horloges ou ne peut pas renvoyer des résultats corrects. Dans ce cas, ce lien ne doit pas être pris en considération lors du calcul du mot de commande DCO. Sachant que toutes les liens ont leurs cœfficients de poids, il est facile de désactiver un lien s'il est cassé. Un nœud défaillant est plus difficile à traiter. Une possibilité consiste à contourner ce nœud et utiliser le signal d'horloge voisine provisoirement.

De cette façon, même s'il y a une erreur quelque part dans le système, la puce ne sera pas brisé rapidement. Il faut noter qu'il existe un compromis entre la complexité et la robustesse du système.

La distribution d'horloge pour les circuits en 3-D

Trois-dimension (3D) IC est une technologie émergente poursuivie par l'industrie et les laboratoires. Il offre de nouveaux niveaux d'efficacité, de puissance, et d'autres avantages. La distribution d'horloge dans le circuit doit être dans trois dimensions spatiales, ce qui augmente la complexité du générateur d'horloge. Plusieurs recherches portent sur ces problèmes [34, 35, 38, 6]. Trois topologies typiques ont été résumées dans Fig. 9.1[38]. En général, ils combinent différentes topologies utilisés dans les circuits 2-D, comme arbre d'horloge, l'anneau et la maille. Le point commun dans les trois approches présentées dans Fig. 9.1 est qu'un arbre d'horloge est mis en œuvre dans le 2ème plan. La racine de l'arbre d'horloge est reliée à l'horloge de référence. Dans Fig. 9.1(a), deux autres arbres d'horloge sont utilisés dans les deux autres plans. Les nœuds racines dans les deux plans sont reliés à la racine du 2ème plan par TSV. Il en résulte un arbre d'horloge 3-D avec un seul nœud racine réel, et le déphasage entre deux nœuds de feuille est plus grande que dans un arbre d'horloge 2-D. Dans Fig. 9.1(b), le 1er et 3ème plans utilisent structure maillée à la place de la structure de l'arbre. Chaque nœud dans les mailles sont connectés avec un nœud dans le 2ème plan par TSV, donc un grand nombre de TSV sont nécessaires. Il en résulte une difficulté de mise en œuvre au niveau physique. Dans Fig. 9.1(c), les anneaux d'horloge sont utilisés dans le 1er et 3ème plans, et que les quatre nœuds de coin sont reliés aux nœuds de coin du 2ème plan. Cependant, si le plan est grand, la synchronisation entre les nœuds centraux dans les plans différents est difficile à garantir. Nous pouvons voir les problèmes de ces solutions 2-D classiques sont multipliés par la mise en œuvre 3-D. En conséquence, ces solutions sont d'efficacité limitée.

Nous proposons une méthode basé sur l'étude de ce projet de thèse. Cette méthode (cf. Fig. 9.2) est inspiré par l'approche affichée dans Fig. 9.1(c). Au lieu d'utiliser un arbre

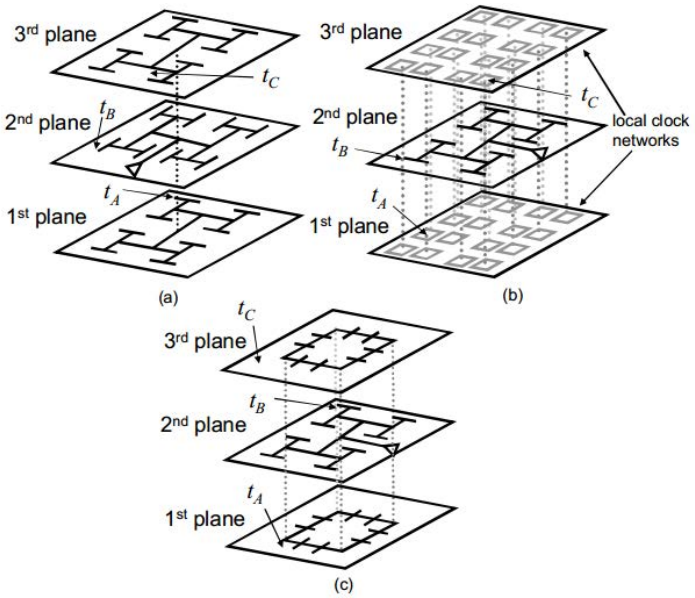


Figure 13: **Différents réseaux de distribution d'horloge 3-D au sein du circuit de test de [38]**: (a) H-arbres, (b) H-arbre et anneaux/mailles locales, (c) H-arbre et anneaux globaux

d'horloge dans le 2ème plan, une architecture distribuée du réseau ADPLL présenté dans ce document (cf. Fig. 2.1) peuvent être mettre en œuvre. Un coin du réseau est relié à l'horloge de référence. Deux autres réseaux d'ADPLLs sont construits dans le 1er et le 3ème plans. Les quatre nœuds de coin dans chacun de ces deux plans sont couplés avec les quatre nœuds de coin du 2ème avion aussi en utilisant la technique d'ADPLL. Par ailleurs, puisque les réseaux d'horloge dans le 1er et le 3ème plans ont quatre signaux d'horloge de référence, ces deux plans peuvent être configurés comme une architecture «piscine» proposée dans Chapter 7. Cette approche proposée exploite la fiabilité et l'évolutivité de réseau d'ADPLL par rapport aux approches classiques utilisés dans Fig. 9.1.

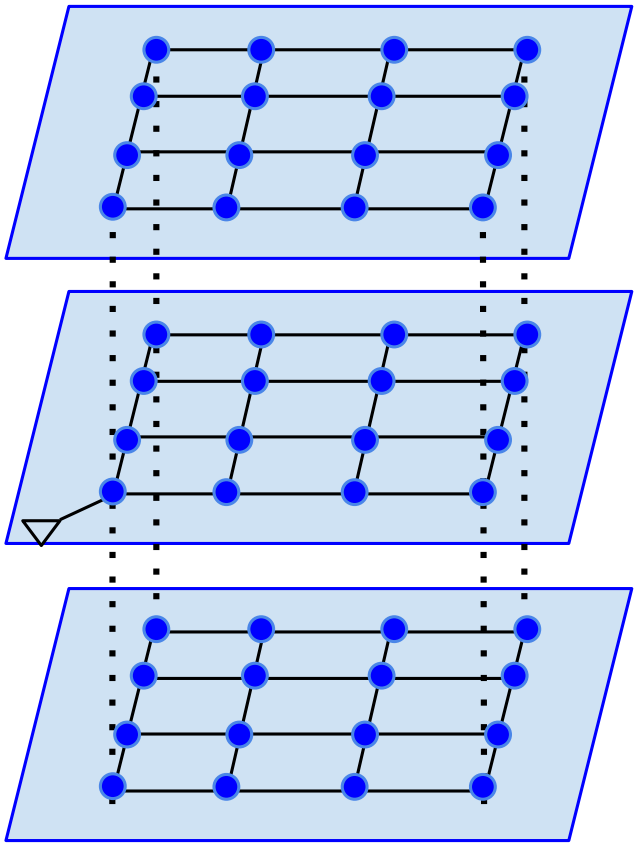


Figure 14: **L'approche proposée de distribution d'horloge 3-D en utilisant le réseau d'ADPLL**

Chapter 1

Introduction

Contents

1.1 Area of focus 1
1.2 Environment of the PhD project: the starting point and motivations . . . 5
1.3 Original contribution of my PhD project 29
1.4 Thesis outline 31

1.1 Area of focus

Advances in CMOS technology have led to an exponential reduction in size of digital circuits (logic cells, transistors). The modern SoC can be regarded as micro-networks allowing different parts of the systems to work together and communicate. Synchronization of the communication becomes a research subject of utmost importance. This study addresses the problem of global clock generation and distribution inside complex and large SoC, so as to allow a full-synchronous communication on the chip. It is motivated by deficiencies of conventional clock generation approaches such as clock tree in the context of deep sub-micron CMOS technologies. The developed clock generation technique is based on a network of coupled oscillators distributed over the chip.

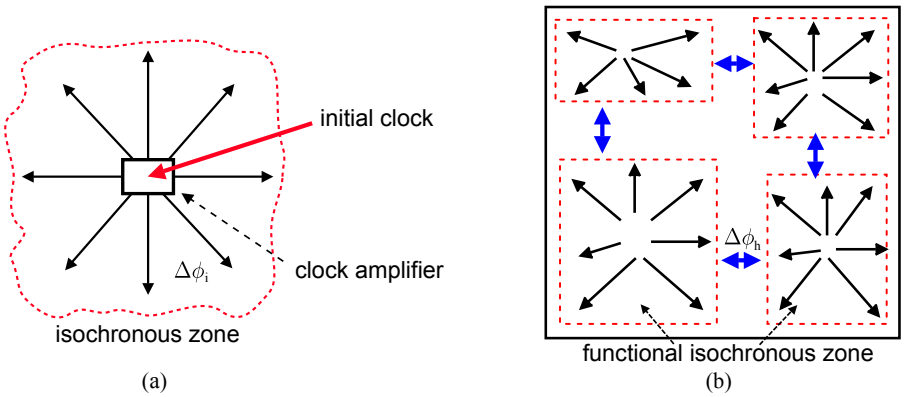


Figure 1.1: **Clock domains in a SoC**: (a) isochronous zone and (b) their placement in SoC

1.1.1 Problem of clocking in large digital circuits

Traditional clock generation in complex circuits uses tree or grid structures [8, 5, 1]. The matching between different clock paths is the key point of the clock network design. In large chips designed in advanced CMOS technology, such a global matching is difficult to achieve. Because even if positions of the leaf nodes are symmetric with regard to the source point of the clock, mismatches between the buffers and the lines introduce uncontrollable skew. In practice, the local clock areas have different sizes and positions, making difficult a perfect clock tree equilibrium. To solve the problem of mismatch, the size of the lines and buffers must be increased so to become less sensitive to fabrication errors, which however makes the solution very expensive, mainly in terms of energy consumption and area[61, 5].

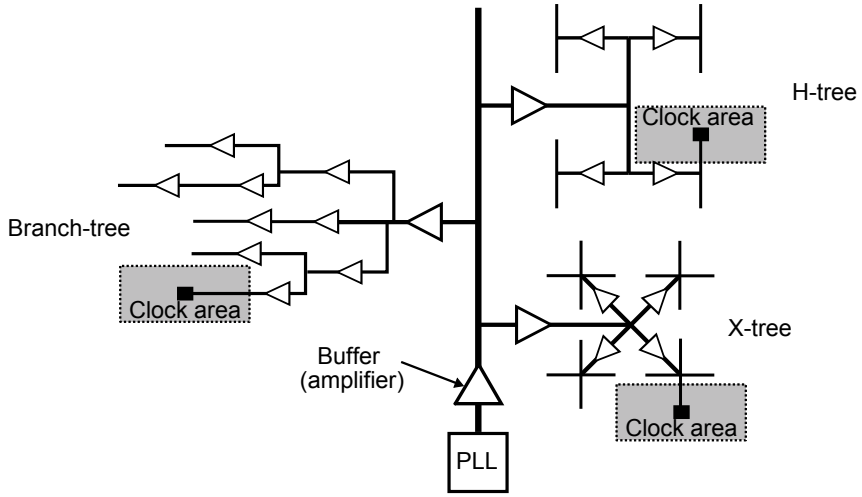


Figure 1.2: **Examples of conventional clock distribution tree structures**: Branch-tree, H-tree and X-tree

Possible improvements of the original tree distribution system consist in providing the clock generator with a skew compensation mechanism [46, 62, 9, 25, 12, 63, 69, 20, 48, 7]. The skew monitoring and control are performed by a dedicated controller called skew compensator gathering information about the skew in all characteristic points of the chip. Its role is to ensure that the clock ticks sent to all clock domains arrive at the same time. It is difficult to implement this strategy because of the requirements imposed on the interconnections of the various regions. In addition, this strategy does not solve the problem of high power consumption associated with buffering of high frequency clock signals.

Even though the cost is high, clock tree or grid is still attractive for industry for now because of it is a mature method for synchronizing the chip and is well supported by current EDA tools. According to the recent published papers of industry, Sumsung's new 28 nm mobile application processor[54], Applied Micro's 3GHz 64b ARM v8 Processor [70] uses a hybrid structure of H-tree and clock mesh.

However, it is obvious that this method is not scalable. With the increase of circuit complexity and number of cores in a processor, centralized clock generation will be too expensive

to realize. A different way of thinking consists in partition a large chip into local clock areas (domains) [71, 42, 26, 41]. These domains are also known as isochronous zones [4] or Synchronous Clocking Areas (SCA) [21](cf. Fig. 1.1). These zones are small enough so that clock distribution inside can be achieved without any difficulty by conventional techniques. The size of the zone is determined by the maximal clock signal propagation delay inside the zone that would not violate timing constraints of the circuit. This delay depends on process technology and wire routing, hence the zone does not have an exact dimension. An empirical number of gates/latches inside the zone is approximately 200-300 thousands.

The communication inside the SCAs is synchronous, thus the problem of global communication is reduced to communications through the borders of the zones. The communication between the blocks situated in different SCAs can be synchronous or asynchronous. In the first case the chip is GSLS (Globally Synchronous Locally Synchronous), in the latter case the chip is GALS (Globally Asynchronous Locally Synchronous).

Because of the difficulties of global synchronization, SoC engineers orient their choice toward the GALS[37]. This is achieved by using bi-synchronous interfaces (ex. FIFO) guaranteeing the signal integrity at price of design complexity and increased latency. Moreover, in asynchronous circuits, the reliability is difficult to guarantee at design stage for two reasons. First, an asynchronous system is analog, because the time is continuous, and its exhaustive verification is impossible. Second, the metastability risk may be worse than usually expected, because the clocks of SCAs are not really fully independent. Indeed, these clocks are derived from a single input clock and distributed to different modules. Their relative phase shifts depend upon fast and slow changing parameters like aging (slow), temperature (medium speed), and voltage (fast). Metastability could occur for certain values of these parameters. During a short period of time, slow and medium speed parameters do not change, leading the metastability to reoccur repetitively, so increasing the risk of the system failure[50].

Therefore, although by using GALS there is no more worry about mismatch between clock signals in different domains, it gives away the advantage of global synchronous circuit like reliability, deterministic behavior and high communication rate. Loosely synchronous [58] arises when some bounds on the frequencies or phases of communicating blocks are known. In this style, if the clock generation method ensure that timing requirements are met, the handshaking is unnecessary for data transfer, resulting in higher circuit performance. However, synchronizers and FIFOs are still necessary, so metastability is not avoided. Moreover, to determine the optimal size of FIFO buffers, a timing analysis is necessary to bound how far the relative phase difference between the sender and receiver may drift. This kind of timing analysis is not yet common for on-chip timing and not supported by EDA tools.

This work contributes to this subject by studying an distributed clock generation method for synchronizing a large circuit. It is addressed to the designers developing GSLS circuits: the proposed solution is an alternative to traditional global clock distribution techniques.

The following section presents an overview of the distributed clock generation proposed during HODISS project.

1.2 Environment of the PhD project: the starting point and motivations

This PhD work is deployed in the framework of a research project at LIP6 laboratory focused on techniques of alternative clocking, started at 2007 and funded by two consecutive ANR grants (HODISS, 2008-20012 and HERODOTOS, 2011-2014). The project was focused on a study of a particular architecture of clock generation, based on a network of oscillators coupled by a network of All-Digital Phase Locked Loops. The goal of the project is a development of a novel clocking technique and its validation by design of two IC prototypes. My PhD thesis is directly related to this objective. Before my arrival, the project were carried out by a PhD student Eldar Zianbetov who finished his PhD when I arrived, and a post-doctoral researcher Mohammad Javidan. Since the starting point of my PhD project was the result obtained by the previous work, in this section I will summarize the state of the project at that time. In Subsection 1.2.10 I will explain the shortcoming and challenges which motivated the opening of my PhD position, and in Section 1.3 I will present the objectives of my PhD.

1.2.1 Network of coupled PLLs for clocking: history of the concept

In 1995 Gill Pratt and John Nguyen proposed a distributed clock generator based on network of coupled analog PLLs [39]. Our work is based on this kind of architecture. For this reason, this subsection provides essential information about that.

The proposed clock generator belongs to the family of multioscillator architectures based on a network of coupled oscillators. In such a clocking scheme, Fig. 1.3, a chip is partitioned into local clock areas, each of them having its own clock generator (oscillator) which must be synchronized with its neighbors in the phase domain. The goal of the distributed PLL(phase-locked loop) network is to synchronize each oscillator in phase and in frequency*. In a steady state, such a network is a source of fully synchronous distributed local clocks.

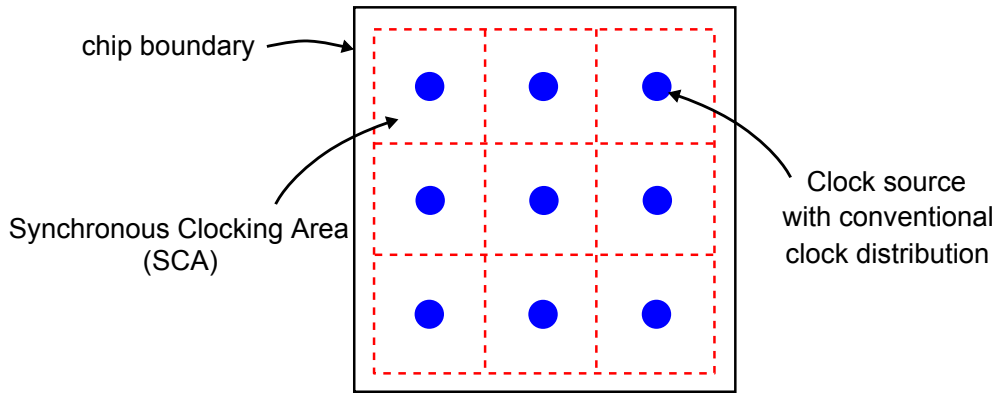


Figure 1.3: **Basic idea of multioscillator clocking approach**

*A synchronization in phase implies a synchronization in frequency

The architecture proposed by Pratt and Nguyen is a 2 dimensional Cartesian mesh network (cf. Fig. 1.4(a)), where the nodes are the local clock generators and the arcs represent the coupling links between these local generators. Each local generator is linked only with its immediate Cartesian neighbors. In such a topology, a global distribution of clock signal is replaced by local signal transmission using short coupling links. This is the main advantage of such an architecture comparing with centralized clock generation approaches.

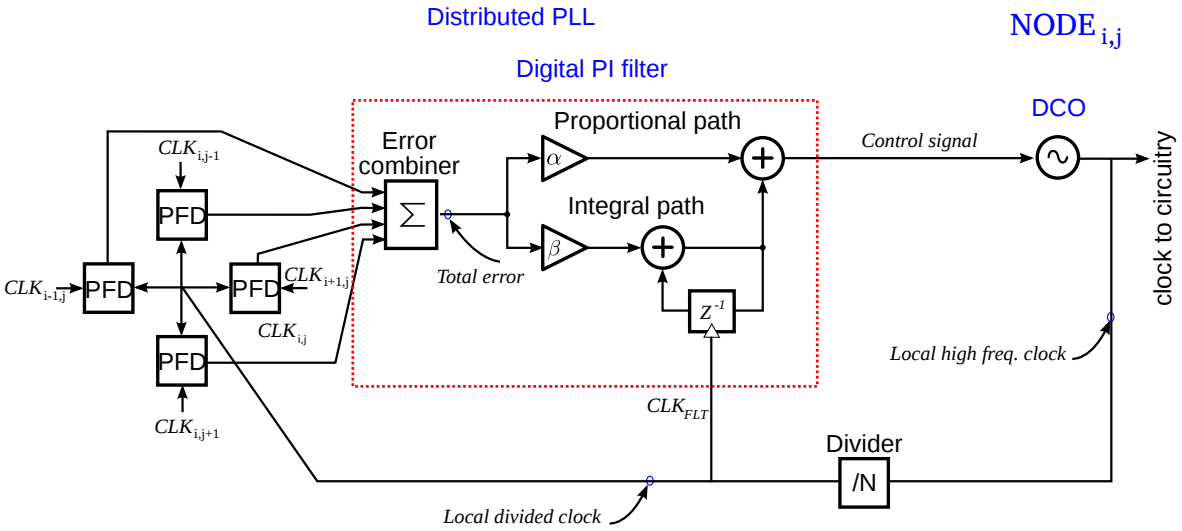


Figure 1.4: **Topology of the proposed clock network and architecture of the network node**

The coupling between the oscillators is implemented in the phase domain via *phase comparators*, Fig. 1.4(b). Each phase comparator measures the phase error between the neighboring two oscillators. This measure is then used by the control circuit associated with the oscillator in order to provide a control signal forcing the oscillator to synchronize with its neighbors. The control signal impacts directly the frequency of the oscillators – which is a derivative of the oscillator phase. For each oscillator a, the phase error $e_{a,b}$ is defined as the difference between its own phase and the phase of its neighbor b. The phase ϕ and the phase error are defined modulo 2π: the most common definition of the phase error is [39]:

$$e_{a,b} = (\pi + \phi_a - \phi_b) \bmod 2\pi - \pi. \tag{1.1}$$

According to this definition, $e_{a,b}$ can have values in the interval $[-\pi, \pi]$. The phase error "seen" by the oscillator b between b and a is $e_{b,a} = -e_{a,b}$. For this reason, each phase comparator is associated with two oscillators and generates two phase error signals e and $-e$.

Fig. 1.5 presents an example of an autonomous (without input reference signal) network composed of two oscillators. In a more complex network, each oscillator (i, j) receives $n_{i,j}$ errors with its $n_{i,j}$ neighbors (Fig. 1.4). These errors are processed by the control block including an error combiner and a loop filter. The error combiner can be in the simplest case a weighted adder. The filter processes the combined error signal called *Total error* so to

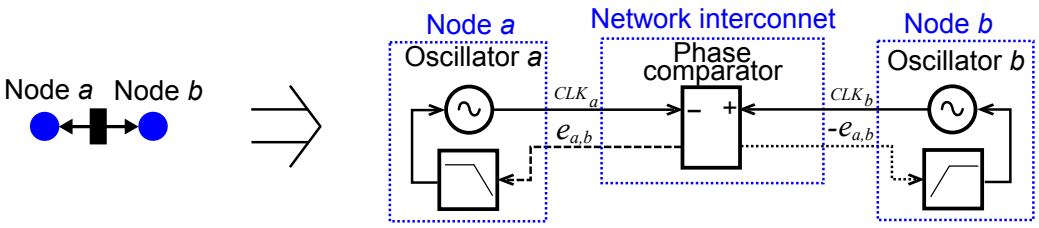


Figure 1.5: **Phase coupling between two oscillators**

generate a control signal on the oscillator input. The objective of the control is to keep at the signal *Total error* close to zero.

It has been proved that such a system has a stable operation mode in which all oscillators have the same phase. The existence of this mode is conditioned by a right choice of the network parameters, in particular, the parameters of the control blocks of the nodes. It has been shown that apart from the mode in which all oscillators have the same phase, there are several modes in which a fixed phase shifts between the oscillators exists. Pratt and Nguyen studied this phenomenon and indicated several solutions for the selection of the desirable stable synchronous mode. This issue is discussed in Subsection 1.2.9.

The architecture proposed by Pratt and Nguyen has been successfully implemented by Gutnik and Chandrakasan [15]. The implemented chip contains 4×4 voltage controlled oscillators (VCO) synchronized by a PLL network.

The analog nature of this system is its main drawback. The clock generator is usually integrated with the digital blocks which use the clock signal. The performance of an analog PLL in a digital environment may be drastically degraded by perturbations due to switching in the digital circuits. Moreover, using analog PLL network for clock generation makes technology migration more difficult and reduces the design portability of the overall SoC.

For this reason, the project carried out at UPMC is focused on a digital architecture of the network of PLL for local clock synchronization[43]. In the next subsection we present the principles of digital phase synthesis and its advantages for the distributed clock generation.

1.2.2 Digital phase synthesis

This subsections explains how all-digital PLLs can be used for generation of clock for large digital circuits.

The principle of digital phase synthesis can be illustrated by a single All-Digital Phase-Locked Loop (ADPLL) as an example. Known since a long time but actively used since one decade, the digital PLLs has recently gained ground on the analog PLLs [55]. The ADPLL operates following the same principle as conventional analog PLL, and functionally has the same structure (Fig. 1.6).

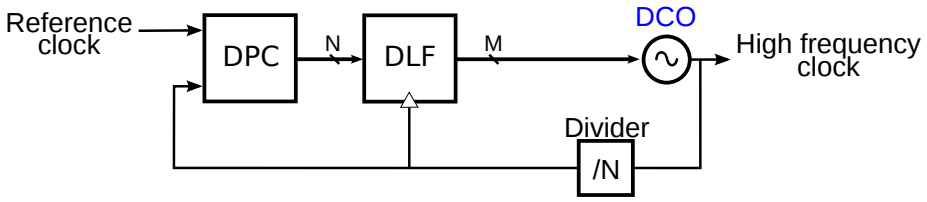


Figure 1.6: **Block diagram of the ADPLL**

An analog phase comparator generating a signal proportional to the phase error is replaced by a digital phase comparator (DPC), which generates a digital code proportional to the error. This code is processed by a digital loop filter (DLF). Thereafter, the digital signal from the filter output is used to control the Digitally-Controlled Oscillator (DCO) directly. The divider defines the ratio between the reference (input) and output frequencies of the ADPLL, i.e. the frequency multiplication factor.

A digital PLL is a system processing mixed analog/digital signals: the target control quantity is analog (the phase), thus in a broad sense, the DCO and the digital phase comparators are digital analog converter(DAC) and analog digital converter(ADC) respectively. However, as it will be shown in Chapter 3, these two blocks can be implemented with digital cells. Such a circuitry is weakly sensitive to the perturbations generated by digital circuit environment. Hence, the drawbacks usually associated with analog circuits are attenuated.

A digital PLL has the following particular property. The phase error (the quantity to be regulated) is sampled by the divided DCO output signal; the same signal is used for the digital processing block clocking. However, the DCO output signal depends on the output of digital processing block. By consequence, an ADPLL is a self-sampled system. This fact significantly complicates the system analysis; the properties of the ADPLL network related to the self-sampled operation were studied in the frame of the PhD project of Jean-Michel Akré [2]. Comparing with analog frequency synthesis, the digital phase synthesis has numerous advantages:

- Use of digital design techniques.
- Reconfigurability and programmability.
- Immunity to perturbations.

These advantages of digital phase synthesis inspired the research team at LIP6 to study the possibility of synchronization of an array of oscillators by a network of all-digital PLLs. A first prototype of such system was designed in CMOS 65 nm technology. The prototype implemented a 4×4 network with 16 nodes. It is presented in the next section.

1.2.3 Presentation of an ADPLL network prototype designed at LIP6 prior to my PhD thesis

Fig. 1.7 presents the architecture of the network and the location on different blocks on a chip. A chip is divided into zones we call "SCA" (Synchronous Clock Areas, similarly as in GALS literature). In the center of each zone there is a digital loop control filter and a DCO (Filter/oscillator block (FO) in Fig. 1.7). Each phase frequency detector (PFD) is shared by two neighboring SCAs.

The structure of one network node ($NODE_{i,j}$) in this specific ADPLL network is displayed in detail in Fig. 1.8. Compared with an analog PLL network in Fig. 1.4(b), it uses digital phase/frequency detector, a proportional-integral (PI) digital loop control filter, and a digitally controlled oscillator. Four clock signals generated by neighboring nodes around $NODE_{i,j}$ are used as references. PFDs quantize the phase differences between these reference signals and the locally generated divided clock. These quantized binary codes are summed by Error combiner and then processed by the PI filter, which updates the control signal for DCO at each cycle. The PI filter is sampled by the local divided clock, by consequence, this is a self-sampled system.

The following subsections Subsection 1.2.4 to Subsection 1.2.6 present the principles and implementations of these blocks designed in LIP6, and emphasize the features specifically related to the digital nature of this system.

1.2.4 Phase frequency detector (PFD)

PFD principle

A phase comparator is a device measuring the difference between phases of two periodic or quasi-periodic signals. Modern PLL circuits use phase comparators providing the sign of the phase error [14]. They are usually called *phase-frequency detectors* (PFD), for the reason which will be explained later. In this document, the following convention is used: the input signals of the phase comparator are named *ref* (signifies the *reference clock*) and *div* (signify the *divided* feedback clock), and we concern their rising edges. When the rising edge of *ref* is leading, the phase error is positive.

The main difference between digital and analog PFD, is that the digital PFD generates a digital value of the error immediately after the end of the measurement, and the result is stable until the next measurement. Whereas existing analog PFDs codes the measured phase difference with a PWM(pulse width modulation) signals [14]. A confusion is often made between analog PFDs composed of digital components but whose output information is contained in the widths of output pulses (sometimes called "digital") and a digital PFD defined in the present paragraph. These PFDs using PWM signals are analog because their analog pulses are continuously variable analog quantities.

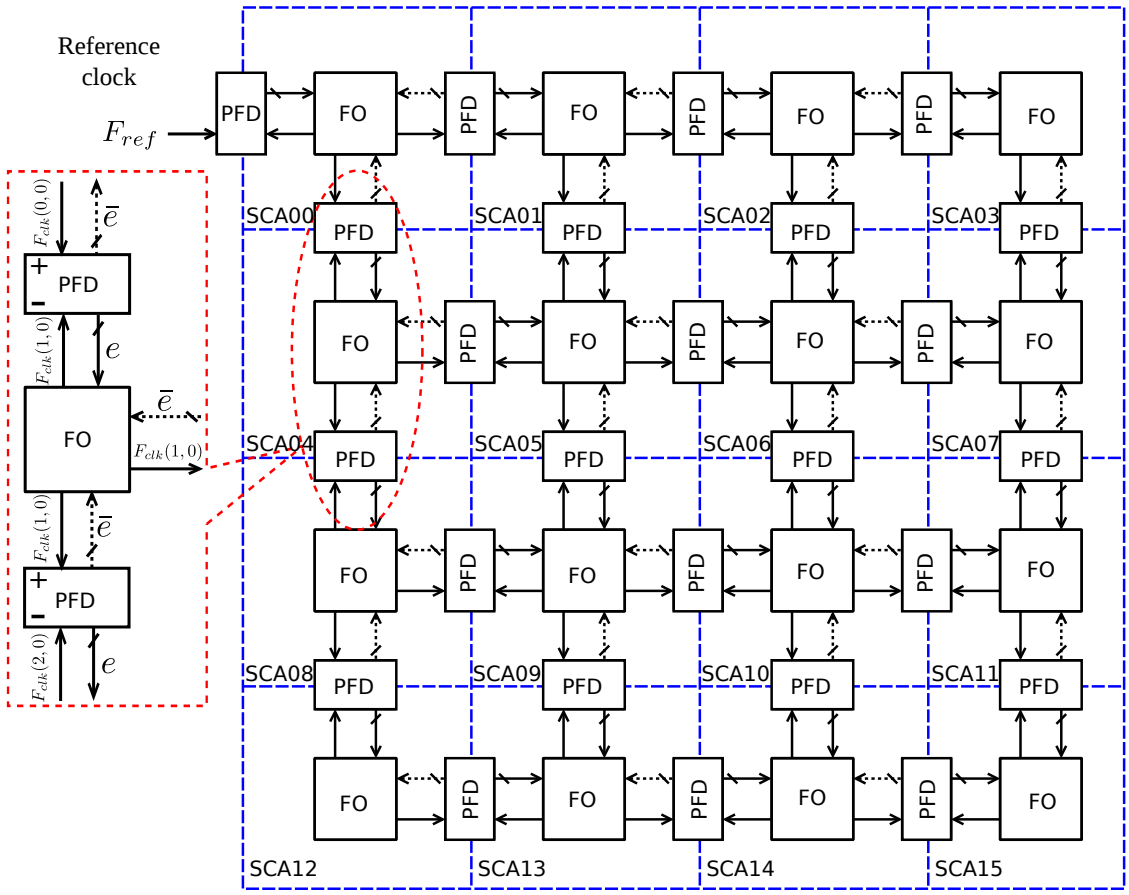


Figure 1.7: **Structure of the first ADPLL network**

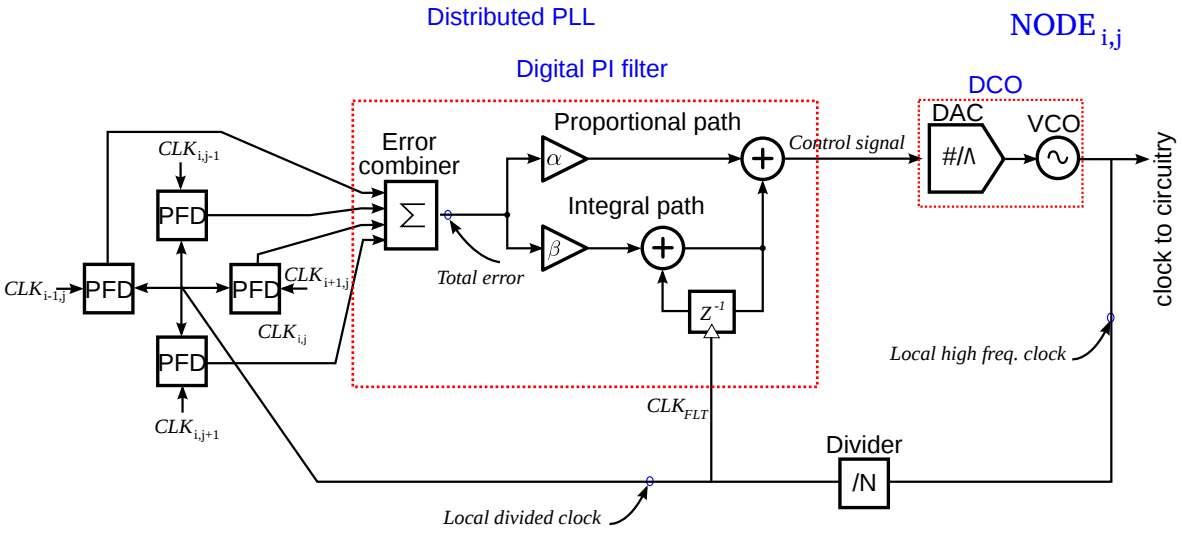


Figure 1.8: **Structure of a node in ADPLL network**

In a digital PFD, the phase error sign detection is achieved with a four states finite-state automaton. These states are described by values of two internal binary signals *MODE* and *SIGN*. A measurement cycle starts when the *MODE* signal is zero and the value of the *SIGN* signal has the value of the last measured error sign. In this state the automaton is ready for a new measurement and waits for an event on one of the inputs. When an event arrived on the

input, the *MODE* goes to '1' and the *SIGN* value takes either '0' or '1', depending on the input which registered the event. After that, the automaton waits for an event on the other input, ignoring all following events on the first input. As an event arrives on the other input, the *MODE* bit goes to '0' and a new measurement cycle can start.

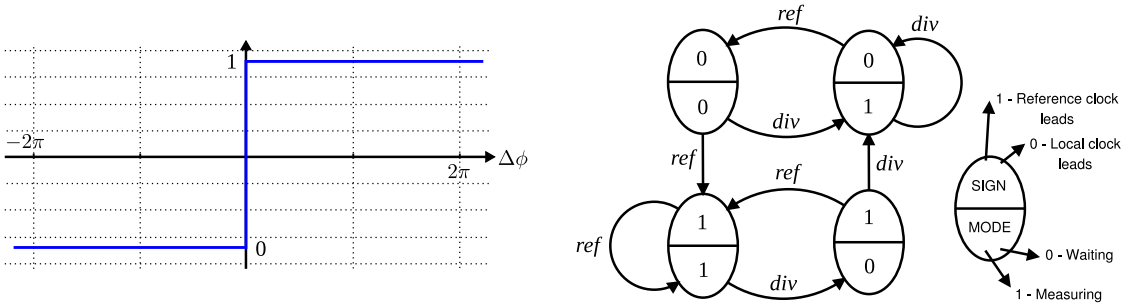


Figure 1.9: **The phase/frequency detector**:(a) transfer function and (b) state diagram

Hence, the *SIGN* signal value indicates which input receives a rising event first during the last measurement cycle. By convention, when *ref* is leading, SIGN is high (a positive phase error). Such a function is called in literature *bang-bang(BB) phase detection*, and the corresponding hardware block *bang-bang detector*.

Considering the finite-state automaton studied above, we can note that duration of the *MODE* signal provides the time equivalence of the absolute value of the phase error. This signal can be used as an input of an analog phase detector, or passed through a time-to-digital converter (TDC) so as to obtain a digital output, which is then combined with the sign bit so as to obtain a signed digital result. We used the latter approach to obtain a fully-digital PFD. Its architecture and input-output characteristic is given Fig. 1.10. Note, that the aspect of the characteristics for small phase errors is chosen so to maximize the information carried by the digital signal: The phase error less than one step of PFD is represented by +1 or -1 depending on its sign. Thus the minimum output of PFD is ± 1. The PFD has two outputs, the phase error $e_{ri}[n]$ and $\bar{e}_{ri}[n] = -e_{ri}[n]$.

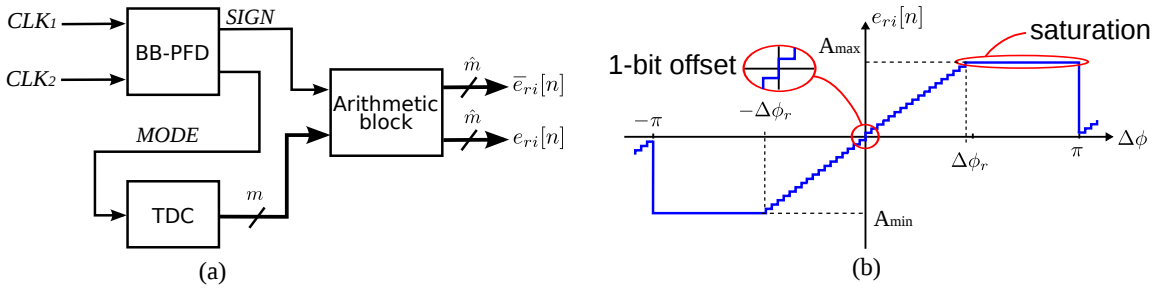


Figure 1.10: **Proposed phase/frequency detector for clock network**: (a) block diagram and (b) transfer function

The dynamic range of the PFD depends on the application context, as well as on the shape of the PFD transfer characteristic (linear or not). For this implementation, a linear

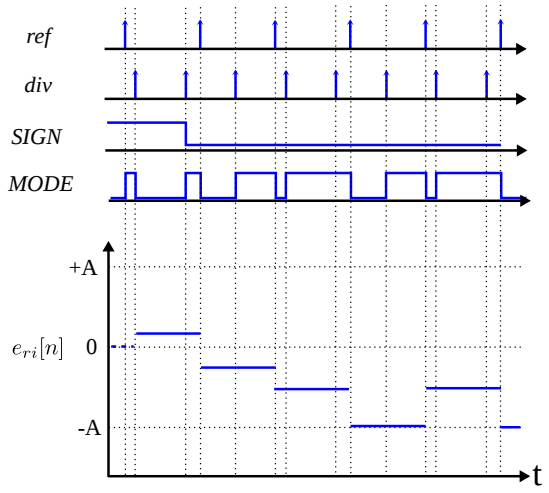


Figure 1.11: **Principle of operation of proposed PFD**: initial fault result about sign of the frequency error can be observed at the beginning; resolved at the next cycle of operation

characteristic of the digital PFD is chosen. The linear range of the PFD may not necessarily cover the range of $\pm\pi$: starting from certain value $\Delta\phi_r$ of input error, the output may saturate. This is because the essential operation mode of a PLL is when the synchronization is close to be achieved. In this mode the phase error is small and falls in the linear range of the characteristic. That allows a reduction of the number of bits of the PFD output while keeping a high measurement precision.

The waveforms in Fig. 1.11 illustrate the operation of the circuit. The initial states of the BB-PFD and TDC are unknown and *MODE* bit has low logical level. Thus, independently of the initial state of the phase error sign, when the *ref* event happens, the *SIGN* becomes '0' and TDC waits for the *div* event, counting the time elapsed from the first *ref* event. After the *div* event happens, the error code is ready, and the *MODE* bit becomes '0' again, waiting for one of the events *div* or *ref*. It is very important to note that the information at the output of the proposed multi-bit PFD is ready only at the falling edge of the *MODE* bit, as it can be observed on the diagram.

Implementation of digital PFD by the LIP6 team, prior to my PhD

The bang-bang detector architecture used for this project is inspired by [59]. As mentioned previously, the BB-detector measures the sign of the phase error and generates a time interval corresponding to the absolute phase error value.

The detector architecture consists of two input flip-flops, the arbiter circuit filtering the metastability, the output buffer latch and the reset logic (Fig. 1.12).

The input registers detect the input events and generate '0' at the outputs $\bar{Q}$ as the events arrive. The principal role of the flip-flops is to detect the first event arriving on one of the inputs *ref* or *div* and to ignore all subsequent events arriving at the same input till the first event arriving on the other input. In the waiting state, the outputs $\bar{Q}$ are '0', when events

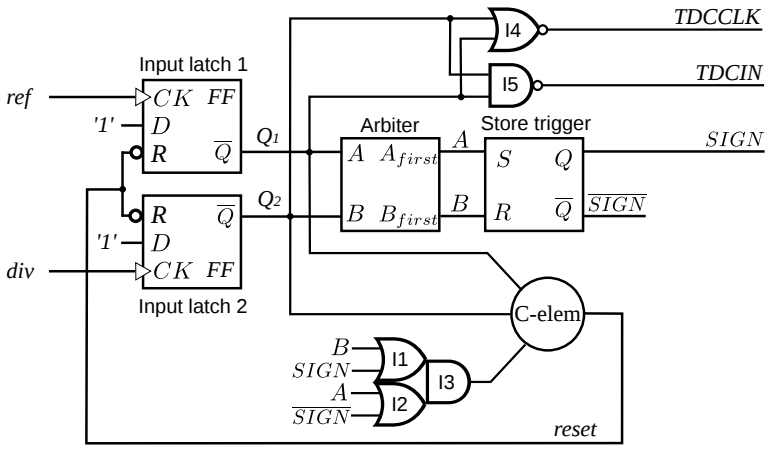
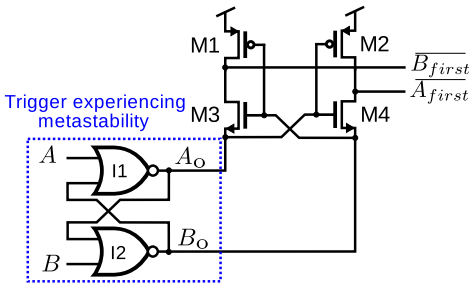


Figure 1.12: **Schematic diagram of the bang-bang phase/frequency detector**: taken from [59]

arrive, the outputs $\bar{Q}$ return to '1'.

The arbiter plays two roles. The first one is the generation of the signals at the outputs A_{first} and B_{first} which are '1' or '0' depending on which input either A or B receives an event first. This information is then stored till the end of the measurement cycle. The second role is to filter the possible metastability resulting from simultaneous arriving of events at inputs A and B.

The principle of the metastability filter is shown Fig. 1.13. If the flip-flop goes to a metastable state ($Vdd/2$, $Vdd/2$), the metastability filter composed of the transistors M1-M4 produces '1' as long as the difference between the output voltages of the flip-flop does not exceed the transistor (M3 or M4) threshold voltage. After that, the actual state of the flip-flop is propagated to the output. This circuit makes impossible a metastable state at its output, however, it generates a delay of unpredictable value, which is surely able to produce a metastability in the further stages.



A_o	B_o	$\overline{A_{first}}$	$\overline{B_{first}}$
0	0	1	1
1	0	0	1
0	1	1	0
Vdd/2	Vdd/2	1	1

Figure 1.13: **Proposed in [59] arbiter circuit**

The bang-bang detector operates as follows. In the initial (waiting) state, the arbiter inputs are at '1' level, and the arbiter has '1' at both outputs, keeping the buffer trigger in the storage mode. The buffer trigger keeps the *SIGN* value detected from the last measurement cycle. When one of the signals Q_1 and Q_2 goes low, the trigger of the arbiter is set to a well-defined state, and after the second signal goes low, the arbiter trigger is in the storage

Table 1.1: **C-element truth table**

A	B	C	Z
0	0	0	0
1	0	0	Z_{n-1}
0	1	0	Z_{n-1}
1	1	0	Z_{n-1}
0	0	1	Z_{n-1}
1	0	1	Z_{n-1}
0	1	1	Z_{n-1}
1	1	1	1

mode. If the falling edges on A and B inputs of the arbiter arrive simultaneously or with a vary small time interval, the flip-flop of the arbiter can trap into a metastable state. In this case, the arbiter outputs '1'. In this case, the output buffer trigger is in the storage state, and it outputs the sign value detected from the last measurement cycle. If the phase error is far from zero and if there is no metastability problem, the sign value is defined just after the arriving of the first event (after the flip-flop delay). Otherwise, the correct value of the phase error sign may be established on the output with a delay whose value is random.

The buffering RS trigger is needed to store the value of the phase error sign during the measurement cycles.

The truth table of the Muller C-element is given in Tab. 1.1. Its role is to generate a reset signal for the input flip-flops only when two conditions are fulfilled simultaneously:

- Events are detected on both inputs by the input registers
- The arbiter is out of the metastable state, and the output of the buffer register is set to a well-defined state. This condition is verified by the combinational logic composed of gates I1-I3

The reset of the input latches marks the end of a measurement cycle. The duration of this reset pulse is determined by the loop delay: C-element keeps the value '0' till the input registers reset their states and the arbiter's trigger comes to the state with '0' at both outputs (and '1' at the outputs of the metastability filter). This state corresponds to an initial state of the bang-bang detector.

As we said at the beginning of this subsection, the second role of the bang-bang detector is a generation of the *MEASURE* signal whose duration is equal to the time equivalent of the phase error. This time interval is then quantized by the TDC. The *MEASURE* signal is delimited by two events *TDCIN* and *TDCCLK* generated by a combinational logic from

the outputs of the input latches (Fig. 1.12). The event *TDCIN* marks the start of the time interval, *TDCCLK* marks its end.

A simple architecture using delay chains is applied for the TDC design in the first test chip. The typical resolution of a delay chain has the same order as the elementary buffer delay available in the technology. For CMOS 65 nm this is 30 ps. Hence, for the project HODISS, 30 ps is defined as TDC resolution. The dynamic range of the phase comparator $(-\Delta\phi_r, +\Delta\phi_r)$ was chosen to be $-\pi/4, \pi/4$. For the nominal output frequency 1 GHz divided by 4, $\Delta\phi_r$ corresponds to 500 ps. This range needs about 15 measurement steps, which can be coded by 4 bits. With one bit added for the sign of phase error, the output PFD word has 5 bits.

The proposed architecture of the TDC is based on a tapped delay line, and is inspired by the architecture presented in [33]. The delay line implements a discrete set of time (delay) values defining the output ADC grid. The time interval to be measured is compared with values of this set, and a corresponding digital code is produced. This architecture is similar to flash ADC, where measured signal is compared to discrete reference levels. The time interval to be measured ΔT is defined by the starting and ending events given by the rising fronts of the signals *TDCIN* and *TDCCLK*. The starting event is applied to the input of the delay line. If initially all output buffers had '0' values, after the rising front on the signal *TDCIN* there is a propagation of '1' through the line. The depth of this propagation is proportional to the elapsed time. The latches detect its propagation depth through the delay line during the measured time interval. Hence, the outputs of the buffer can be considered as a digital chronometer giving the digitized time elapsed from the starting event. The latches produce the snapshot of the state of the delay line at the moment when the ending event arrives. The obtained thermometer coded word is then converted to a binary code and is used to generate the output phase error code of the PFD.

The implemented TDC uses 14 delays which provide 15 quantization levels (0-14τ_{TDC}, τ_{TDC} is resolution of TDC, which equals to the single delay value) including zero and the saturation level (Fig. 1.14). The PFD_{out} is in the range $[-15,-1]\cup[1,15]$.

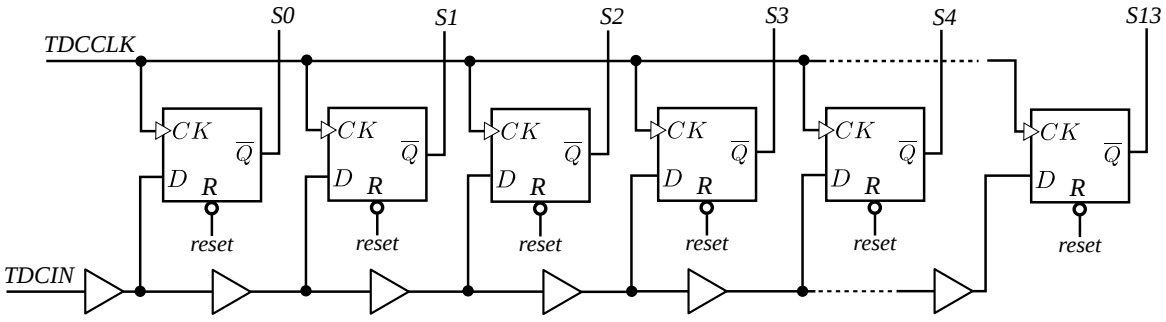


Figure 1.14: **Block diagram of proposed time-to-digital converter**

1.2.5 Digital loop control of ADPLL network node

This subsection describes the digital loop corrector of node used in the 1^{st} prototype designed at LIP6. The simplified architecture of this block is given in Fig. 1.8. This block processes the phase errors between the local clock and 2, 3 or 4 neighbors issued by the corresponding PFDs. The purpose of this block is to generate a control word for the input of the DCO. It includes two cascaded elements: an error combining block and a proportional-integral digital filter.

A detailed schematic of the implemented digital processing block is provided in Fig. 1.15. The block receives four 5-bit signed binary words as inputs, and generates a 10-bit unsigned word for the DCO control (the output of the adder ADD5). This signal is applied to the input of the DCO. The schematic in Fig. 1.15 includes the encoder converting this input binary code to the A, B and C signals necessary to control the DCO core. This encoder is described in Subsection 3.3.2.

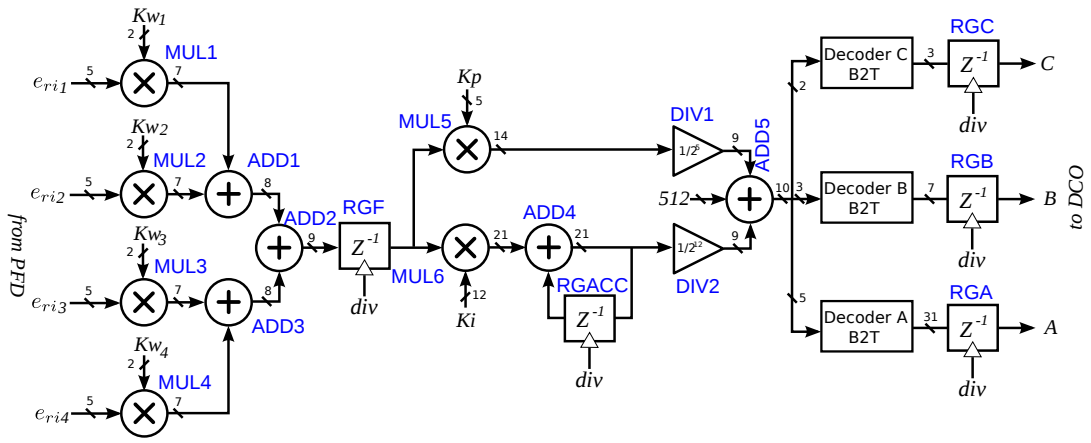


Figure 1.15: **Error signal processing block:** four input gain controllers followed by the four-input adder, PI filter and three Binary-to-Thermometer(B2T) decoders. Shown timings are maximal in a worst conditions.

The error combining block receives four 5-bit 2-complement coded words representing the phase errors with neighbors. These values are passed through four gain blocks (multipliers with a constant) MUL1-MUL4 and then summed using three two-inputs adders ADD1-ADD3. The weighting coefficients of the gain blocks $Kw_1 - Kw_4$ are programmable (cf. Subsection 3.3.4). Each gain can take independently a value in the set {0,1,2,4}. These values are powers of 2: the product is implemented as a binary shift.

The four inputs adder operates with four 7 bit operands and produces a 9 bit sum. This adder is based on the carry look-ahead (CLA) architecture. The output of the adder ADD3 is buffered with a register RGF. This is necessary to complain with the timing constraints and keep sufficient timing margin. The digital block is sampled with the local divided clock (*div* signal).

The transfer function of the PI filter given in Fig. 1.8 between the points "*Control signal*" and "*Total error*" is given by Eq. (1.2).

$$H(z) = \alpha + \beta \frac{1}{1 - z^{-1}} = \frac{(\alpha + \beta) - \alpha z^{-1}}{1 - z^{-1}} \tag{1.2}$$

where α and β are the gain coefficients of the proportional and the integral paths respectively.

In order to multiply the configurations in which the prototype can be tested, the coefficients of the filter were made programmable. Theoretical investigations [28, 27, 3, 2] provided for the coefficients the following specifications: $\alpha \in \{1 \ldots 0.03\}$, $\beta \in \{1 \ldots 0.00024\}$

The calculations inside the filter are achieved in *fixed point* arithmetic. For the proportional part, the coefficient α is represented as a ratio of a programmable integer number and a power of 2 integer number:

$$\alpha = \frac{K_p}{2^5}, \tag{1.3}$$

where K_p is integer in the range $\{0, 31\}$. The 9 bit input word is first multiplied with K_p then divided by 2^5. The fractional part of the result is then ignored, and only the integer part on 9 bits is applied on the input of the last adder ADD5.

The integral coefficient β is defined as a ratio:

$$\beta = \frac{K_i}{2^{12}}, \tag{1.4}$$

where K_i is in the range $(0, 2^{12} - 1)$.

Note that the adder ADD5 before DCO receives integer values, by consequence, the fractional parts of the results of the proportional and integral branches ($y_{p,n}$ and $y_{i,n}$) are rounded. The actual formula for the DCO input y_n is :

$$\begin{aligned} y_{i,n} &= y_{i,n-1} + \beta x_n \\ y_{p,n} &= \alpha x_n \\ y_n &= [y_{i,n}] + [y_{p,n}] \end{aligned} \tag{1.5}$$

where x_n is the input of the PI filter (output of ADD3).

1.2.6 Digitally controlled oscillator (DCO)

There is an extensive literature about CMOS implementation of Digitally Controlled Oscillators [72, 56, 31]. A deep study of DCO suitable for clock generation in the context of the ADPLL network has been carried out at LIP6 prior to my PhD. This section presents

the key information about the implemented DCO, necessary to understand the contribution I provided to this project.

The designed oscillator is a CMOS DCO with frequency range 999-2480 MHz and 1024 equal frequency steps within it, with highly linear and monotonous code-frequency characteristic. The DCO ring includes 7 inverting stages. The frequency control is achieved through a *width modulation* technique [59, 23, 36] using an array of tuning CMOS inverters. A control scheme employing a mixed thermometer/weighted encoding provides a high frequency precision and monotonicity of the DCO. This technique is a result of compromise between the DCO precision and area economy. In addition, a frequency divider produces all necessary clock signals for the digital filter, the local clocking zone and feedback.

DCO architecture

The ring oscillator is constituted with seven main inverters (MI0-6), each one associated with a column of individually controlled tuning inverters (I0-I6). The oscillation frequency is modulated by the number of simultaneously activated controllable cells. It can be roughly considered that the input and output capacitance of a tuning inverter doesn't depend on its state (active or not). In this case, the load capacitance of all seven stages is constant in time. The number of active inverters defines the driving charging/discharging current, hence, the total delay of the chain and by consequence, the oscillation frequency. The number of stages is chosen so to be closed to a power of 2, and is large enough to produce a well-shaped rectangular signal [17].

Detailed architecture of the DCO is given in Fig. 1.16(a). While the main inverters are simple CMOS inverters (Fig. 1.16(b), MI0-MI6), the tuning inverters are connected in parallel to each stage and distributed over all 7 stages of oscillator. Each tuning cell is a controllable inverter associated with a local control logic decoding the row and column actuation code (Fig. 1.16(c-e)). These tuning elements are organized in three arrays implementing a coarse and a fine frequency tuning.

Coarse tuning

The coarse tuning is achieved by two arrays of identical tuning cells (Fig. 1.16(c,d)). The first array consists of 224 (7 stages $\times$ 32 rows) coarse-tuning inverters (CTI) (Fig. 1.16(c)). The second array consists of 42 tuning cells (7 stages $\times$ 6 rows). 32 cells of second array are called "additional coarse-tuning inverters" (Fig. 1.16(d), CTIA); they are used for *virtual extension* of the number of the columns in array to 8 (a power of 2), in order to simplify the decoding of the binary input word (cf. Section 1.2.6). The 11 remaining cells of the second array are used for the oscillation frequency trimming. CTI and CTIA total 256 identical elements and are designated as #000–#255. Together, they provide $2^8 - 1$ equal frequency steps.

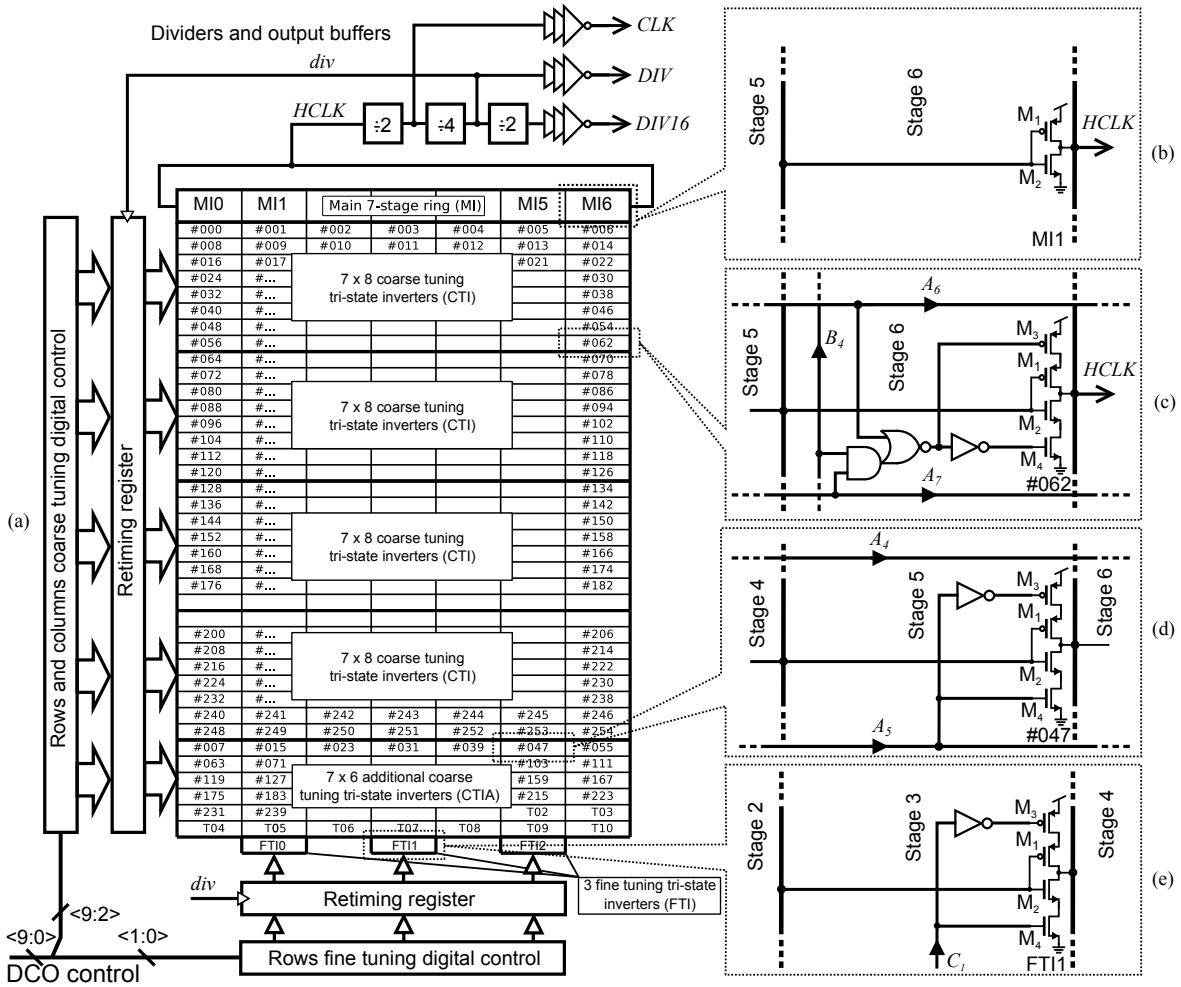


Figure 1.16: **DCO architecture:** (a) structure, (b) main inverters, (c) circuit diagram of the coarse tuning cell, (d) circuit diagram of the additional coarse tuning cell, (e) circuit diagram of the fine tuning tuning cell

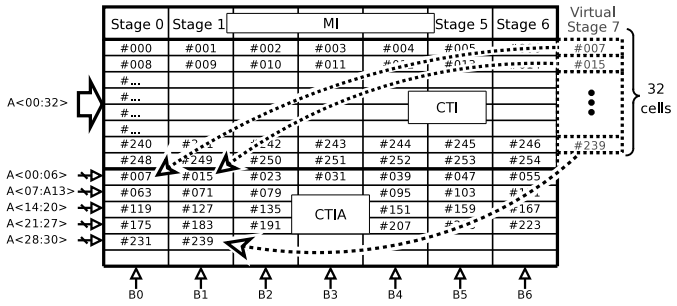


Figure 1.17: Virtual extension of the 8th stage of the oscillator principle

Fig. 1.17 presents the principle of virtual extension of stages. The CTIAs are considered by the control blocks as if they were at the 8th (or 2^3) stage of the DCO.

Fine tuning

Fine tuning is implemented with three identical fine-tuning inverters (Fig. 1.16(e), FTI0-FTI2) connected to the stages 1, 3 and 5. These cells are smaller than the CTI and CTIA cells,

and provide 4 "small" frequency tuning steps inside each coarse tuning step. Together with coarse-tuning inverters, these fine-tuning cells allow 1024 DCO frequency steps. The size of transistors in FTI cells is defined by the dimensions of transistors in CTI/CTIA divided by the number of the fine tuning steps corresponding to each coarse tuning step (size ratio 1:4). The number of fine tuning steps must be a power of two for control simplicity, and was set to 4 as a compromise between the precision (monotonicity may degrade when the number of fine tuning steps increases) and the obtained gain in the overall number of frequency steps.

Control algorithm

The goal of the digital control circuit is to generate the individual command signals for all tuning cells from the 10 bits binary input signal. The whole array is controlled with three thermometer signals A, B and C, whose integer values are derived from the input 10 bit DCO binary code W following the equations which are very easy to implement in digital circuit:

$$\begin{cases} A = W\%32 + 1 \\ B = (W \text{ modulo } 32)\%4 \\ C = W \text{ modulo } 4 \end{cases} \tag{1.6}$$

where % means the integer division. Note that the bit A_0 of the thermometer coded A bus is set as always 1 by this formula.

The logical equation for individual enable signal of the coarse tuning inverters is given by:

$$EN_{cti\,i,j} = \begin{cases} A_{i+1} \vee B_j \wedge A_i, & i \leq 30; \\ B_j \wedge A_i, & i = 31; \end{cases} \tag{1.7}$$

where $i \in \{0,1,..31\}$ is the number of the CTI row and $j \in \{0,..6\}$ are the number of the CTI column. In this formula j defines the physical columns of the array and its range doesn't include the virtual extended row.

For the additional coarse tuning cells implementing a virtual row the logical function is given by:

$$EN_{ctia\,i} = A_{i+1},\ i = 0...30. \tag{1.8}$$

Here i is the index of additional cell equal to the index of the CTI row completed by this cell.

The individual *enable* signals for FTI are:

$$EN_{fti\,i} = C_i \tag{1.9}$$

where $i = 0..2$, i is the index of the FTI cell.

All the tuning cells have the same geometrical constraints as the standard design kit cells and are controlled by digital signals, making the block compatible with digital circuits such as the filter.

Section Subsection 1.2.8 and Subsection 1.2.9 will present the stability issue of the ADPLL network.

1.2.7 Modeling of ADPLL and of ADPLL network

An ADPLL is a non-linear dynamic system. A network of coupled ADPLLs is an even more complex system with high order. To study their behavior and performance, they should be described by models. Transistor level modeling is the most precise way to model a circuit. However, for a circuit with a large amount of transistors, simulation at this level takes long time. Moreover, transistor level modeling is not flexible if we need to modify system parameters. For these reasons, some abstract models at behavior or analytical levels are necessary. By consequence, models at different levels have been created for different steps during the design procedure.

Transistor level modeling — Transistor level modeling is a precise tool for the characterization of an ADPLL. We can use this model to simulate the system as close as possible to the real circuit to be fabricated. For instance, we can verify the system performance at different process corners (TT, SS, or FF) or with variations of PVT (process, voltage and temperature). Transistor-level modeling is unavoidable for mixed signal blocks (TDC and DCO), whereas purely digital blocks may be accurately modeled by VHDL/Verilog description. Transistor-level description of the ADPLL network is presented in detail in Chapter 3.

HDL modeling — Models written in VHDL or Verilog HDL describe systems with discrete (digital) signals defined on discrete or continuous time. It allows a time simulation of an ADPLL network, and an observation of its behavior in transient and steady-state regimes.

A VHDL language provides a possibility to model a system at different abstraction levels. At the top of the hierarchy, macroscopic behavioral modeling provides only correct relations between the inputs and the outputs. A structure model represents the topology of the real digital circuit, and may be more or less detailed. The most detailed model accounts for all elementary digital gates realizing the function of the block. If the Design Kit (DK) of the used technology contains reliable models of the used gates with correct timing information, the simulation provides a result very close to the real behavior of the chip.

In the ADPLL, there are two digital blocks: the PFD and the PI filter. A PI filter can be modeled by its structure in VHDL (cf. Fig. 1.15), with simplified behavioral modeling of all sub-blocks (adders, multipliers, etc.). Otherwise, a detailed gate-level model of the filter provides a very reliable information about the filter behavior. In our project, both mentioned models of filter were used at different design stages.

A TDC can also be modeled at a high abstraction level by a macro-model, but it can also be described by the netlist (interconnection) of elementary digital gates. However, a TDC belongs to the class of asynchronous logic circuit, and the DK we used did not contain

specific gates (e.g., Muller C-element, arbiter, delays, ...), we did not used a VHDL model of the TDC for low structural level.

Since the physics of the DCO operation use techniques issued from the analog electronics (the frequency control by the transistor width modulation, cf. Section 3.3), only high-level behavioral model was used for the DCO. It is implemented with a look-up table containing the output frequency value corresponding to each value of the input control word. The output clock is expressed as:

```
clk <= not clk after period
```

Details of these models can be found in Appendix A.

In some cases, VHDL models defined at high abstraction level may be used for the synthesis of the electrical schematic of a digital block by tools such as Design Compiler of Synopsys. In our project, this was only used for the filter design. Indeed, only this block was compatible with the requirement of a standard "compiled" digital design flow available in our environment.

By consequence, we have divers models of each block in behavior level, structural level and transistor level. And we note that there is a trade-off between modeling precision and simulation speed at different levels. Thus the study requires an open platform able to integrate different levels of description. In our study, we use the AdvanceMS tool to build a mixed-level model of system (VHDL, Verilog and transistor-level Spice), and AMS (analog mixed signal) simulations can be performed using this model.

Mathematical modeling — The models presented above are complex, and can only be solved numerically. However, at the earlier stages of the design, it is desirable to have simplified analytical models of the system, providing simple relations between the system parameters and the main system performance. So, we need a mathematical (analytical) model to study the transient behavior of a single PLL and to know how the parameters of individual blocks affect the behavior of such a system.

However, an establishment of truthful simplified analytical models is difficult, because of two features of an ADPLL: the self-sampling and the discretization of the signals. An ADPLL circuit is a self-sampled system because the loop-filter operates on the irregular rising edges of the divided clock of DCO.

For this reason, in this PhD project, linear time-invariant (LTI) models with both discrete time and continuous time have been created. Fig. 1.18 shows the LTI model of ADPLL for Z-domain transfer function calculation. Quantization is not modeled and the sampling frequency is fixed at the nominal DCO output frequency divided by 4, which is the division factor used in our circuit design. The transfer function of each block in the model is expressed as:

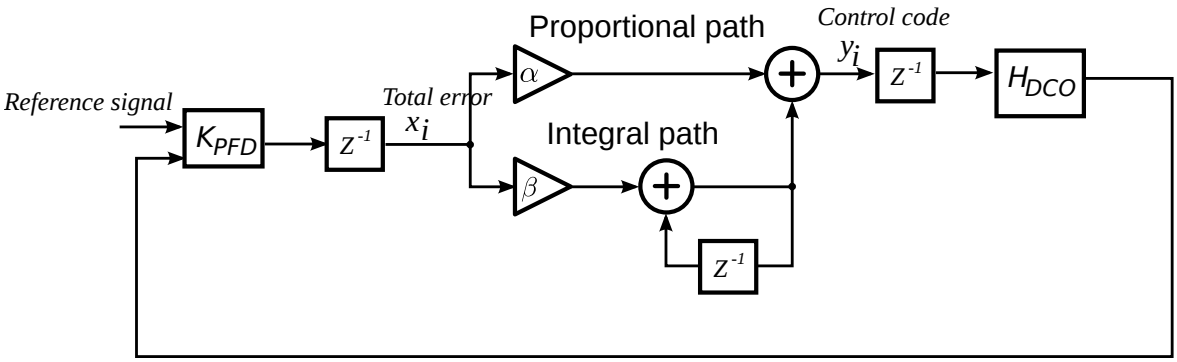


Figure 1.18: **LTI model of ADPLL for Z-domain transfer function calculation**

$$\begin{aligned} H_{\text{PFD}} &= K_{\text{PFD}} = \frac{1}{\Delta T_{TDC} \cdot 2\pi f_s} \\ H_{\text{Filter}} &= \alpha + \frac{\beta \cdot T_s(Z+1)}{2(Z-1)} \cdot Z^{-2} \\ H_{\text{DCO}} &= \frac{K_{\text{DCO}} \cdot T_s(Z+1)}{2(Z-1)} \end{aligned} \tag{1.10}$$

where K_{PFD} is the gain of PFD, which is reversely proportional to the resolution of TDC (ΔT_{TDC}). α and β are coefficients of the proportional and integral paths of PI filter (cf. Subsection 1.2.5). The existence of Z^{-2} can be explained by the two cycles delay introduced by the two registers in the loop filter (cf. Fig. 1.18). The gain of DCO (K_{DCO}) is proportional to the tuning step of DCO (Δf_{DCO}).

By consequence, the system is described in this analytical model using parameters of its blocks. Using this model we can define the generic analytical relation between the system parameters and the performance (cf. Subsection 2.3.2).

The VHDL and transistor level models presented above can easily be extended for modeling of a network of ADPLLs. Functionality and performance of the network can be verified by performing time simulations at these levels.

However, interconnecting several PLLs in a network increases the order of the system, making the analytical modeling of a network complex. A robust design procedure guaranteeing the global stability of network is presented in the next subsection.

1.2.8 Stability of the PLL networks

In the context of the *HODISS* project, our colleagues from the CEA-LETI and AMPÈRE laboratories developed a mathematical tool based on the control theory. It allows synthesizing the node loop filter for the global stability (cf. [3, 27, 28, 29, 2]). The model used in this mathematical tool is in continuous time without considering quantization (cf. the preceding subsection).

The design procedure is based on the decentralized H_∞ control technique making use of the dissipative properties of the system [47]. The system is first represented as a loop including a sub-system of a unidirectional chain of the network nodes $\widetilde{T}$, and a sub-system $\widetilde{M}$ representing interconnections in the network (Fig. 1.19). *ref* represents the input phase of the reference, ε represents the phase errors between neighbors which should converge to zero. In this way, the procedure of filter synthesis is done in two steps: firstly, the local stability (that of the matrix $\widetilde{T}$ is ensured), then the global stability is guaranteed. The procedure provides to the implementation engineers the range of stable PI filter coefficient values.

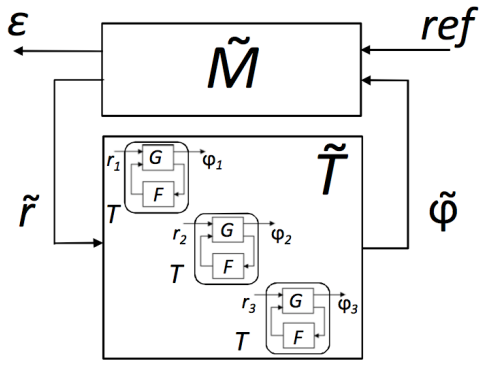


Figure 1.19: **Representation of the PLL network for stability study in [47]**

1.2.9 Multiplicity of synchronization modes

The work of Pratt and Nguyen in [39] highlighted a fundamental problem specifically related to the PLL networks. A PLL network may have several modes in which the local oscillators are synchronized in frequency and in phase, but with fixed phase errors between the oscillators. The residual errors may be zero or not. In our project, only the mode in which the phase errors are zero is suitable for the clocking application. However, when several synchronization modes exist, the actual mode depends on the initial conditions of the system – which cannot be controlled in practice. This section presents this phenomenon in details and provides a review of the solutions to this problem.

Definition of the problem

The multiplicity of synchronization modes has been called *mode-locking* in the literature. It is caused by the cyclic (modular) nature of the phase: the transfer function of the phase comparator is defined modulo 2π (Fig. 1.20).

Such a nonlinear transfer function of phase comparator and a large number of degrees of freedom of a PLL network define multiple synchronization modes. Depending on the initial conditions, the system can converge either to desired state (all oscillators has the same phase) or to undesirable state (oscillator phases misaligned).

The mode-lock phenomenon is illustrated on the example of a 4 nodes network Fig. 1.21, in which the error processing block receives the sum of the errors with two neighbors. The

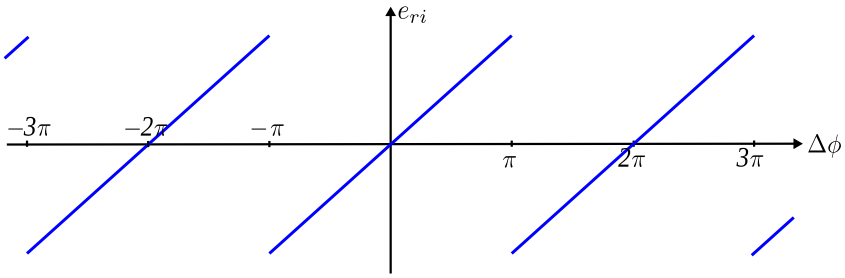


Figure 1.20: **Cyclic nature of the conventional analog linear phase comparator**

error processing block operates so to keep the *Total error* equal to zero. The *Total error* can be zero if :

1) the phases ϕ_1, ϕ_2, ϕ_3 and ϕ_4 of the oscillators are equal

2) the phases have the values given in Fig. 1.21.

In the first case, all phase errors of the network are obviously zero, their sum is zero as well. For the second case, the phase errors are non-zero, but the *Total error* value is zero. For example, for the Node 1: the phase difference with the Node 2 is $\phi_1 - \phi_2 = +\pi/2$, which is compensated by phase difference with the Node 4 $\phi_4 - \phi_1 = -\pi/2$. In this way, the sum for all nodes of this network is zero despite the unequal phases of all oscillators. It can be proven that such a state is stable; once acquiring this operation mode, the network remains in it, since the control objective (zeroing the *Total error* quantity) is fulfilled.

For a network with more complex topology (more nodes), several undesirable states can exist, with fixed phase errors of any values. Hence, in a more complex network, the probability of appearance of undesired synchronization modes increases. Therefore, it is important to take into account this phenomenon during the design of the network. In the following subsection we present the solutions we have studied to design a stable network, spontaneously converging to a state where all oscillators are synchronized in phase and frequency.

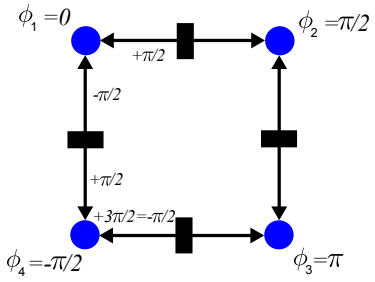


Figure 1.21: **Illustration of the mode-locking phenomenon in a 2×2 mesh network**

Synchronization mode selection: dynamic network reconfiguration

This technique is based on the fact that each stable mode of a dynamical system has a basin of attraction: if the system starts from an initial condition corresponding to a point in this basin,

the system settled up to the corresponding stable mode. The idea of the method suggested by Pratt and Nguyen[39] is to bring the system to a state close to the desired synchronization mode by configuring the network to be unidirectional as the left mesh in Fig. 1.22 [45, 44]. When the system is synchronized, the links between the oscillators are reconfigured so to set the network in the bidirectional configuration (right one in Fig. 1.22). Since in this case the bidirectional network starts in a state close to the desired synchronization, it remains in this mode.

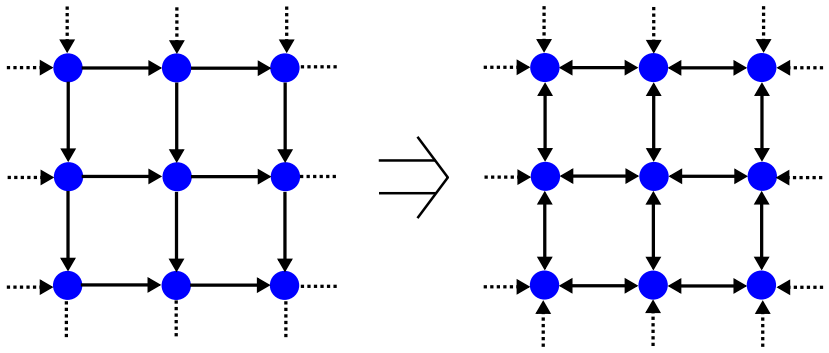


Figure 1.22: **Dynamic reconfiguration of the network from uni- to bidirectional**

This method is particularly suitable for digital implementation since it is very easy to reconfigure the network and to distribute the global signal ordering the reconfiguration. We note that in the original publication of Pratt and Nguyen this technique was judged as being inappropriate for the analog implementation.

In this technique it is not necessary for the digital phase comparator to have a large dynamic range. It can be shown that even if the phase comparators have only one bit of resolution (if they detect only the sign of the phase), the unidirectional network still converges to a state with the phase errors close to zero. However, when the network switches to the bidirectional configuration, the phase comparator characteristic should be linear with positive slope at the phase error values less than $\Delta\phi_r$. Hence, the phase comparator characteristic may have a limited linear range as in Fig. 1.10(b), as far as the residual error obtained in unidirectional mode is within this range.

This technique is selected for the desired synchronization mode selection in our project. The validation of this method has been done by simulation and on a FPGA platform. Details of this implementation can be found in Chapter 5.

1.2.10 Discussion of test results of the implemented prototype

A network of coupled ADPLLs (Fig. 1.7) was implemented in CMOS 65 nm technology. The test of the implemented chip consisted in measuring the phase errors between the local oscillators of the network. This measurement was made out of chip, on signals with divided frequency. An example of measured outputs is given in Fig. 1.23: the 16 nodes of the network have very close phases, suggesting a synchronization of the network.

Fig. 1.24 displays the measurement results of the quantized phase errors among the four clock signals the most remote from the reference clock input. The maximum error between local clocks are within ±60 ps range, which signifies that timing errors do not exceed ±2 steps of PFD quantization step. This result is the best one can achieve with an ADPLL network, according to our study. Indeed, it has been demonstrated that when such a network is synchronized, there are always some neighboring clocks whose residual errors are more that one PFD quantization step. By consequence, in the best case, the maximum phase error between any two neighboring nodes has a value between one and two quantization steps. As show the measurement, the prototype operates in this best case, where the errors don't exceed 2.

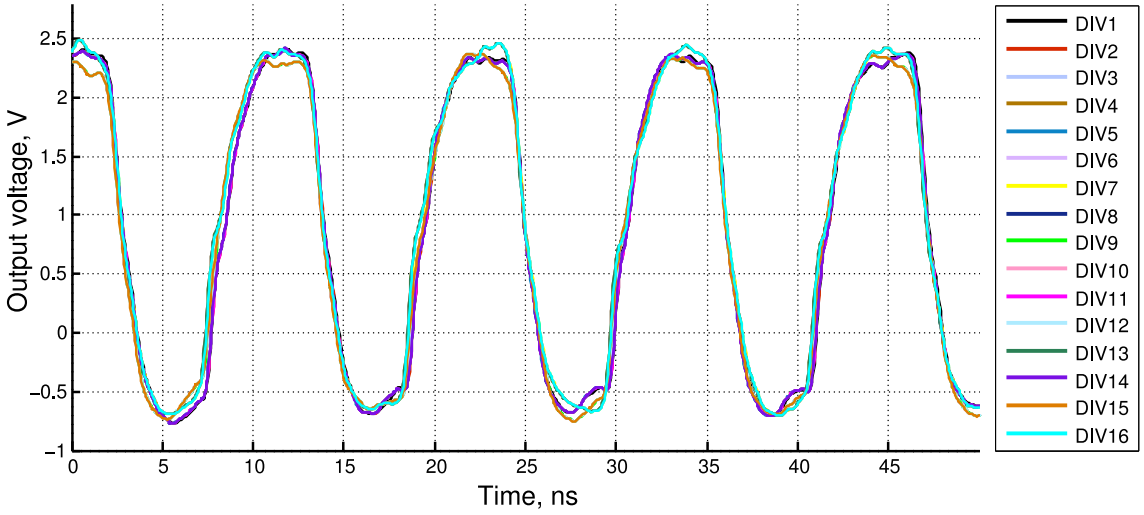


Figure 1.23: **Synchronous clocks in the bidirectional configuration**

Test results of the first prototype demonstrate the feasibility of such a digital distributed clock generator. However, during the design and test process, we find some problems to be resolved and some points to be improved:

1. *Proof of scalability*. From measurement results of the first prototype we observe that although phase error is larger between two nodes in distance than between two neighboring nodes, it does not accumulate linearly in function of distance. But since there are only four nodes on the diagonal of a 4×4 network, the first prototype cannot well demonstrate the relation between phase error and distance. To better observe this relation and also to study the scalability of the ADPLL network, a larger network (ex. 10×10) is necessary. An implementation of a larger network requires an optimization of the existing blocks, in order to minimize the power consumption of the circuit. For example, the DCO implement in the first prototype consumes 16 mA of supply current, which, multiplied by 100, yields 1.6 A: that is prohibitive for an IC implementation.

2. *Improvement of the quality of synchronization*. The TDC resolution of the first prototype is limited to 30 ps due to the constraint of its structure and the minimum gate

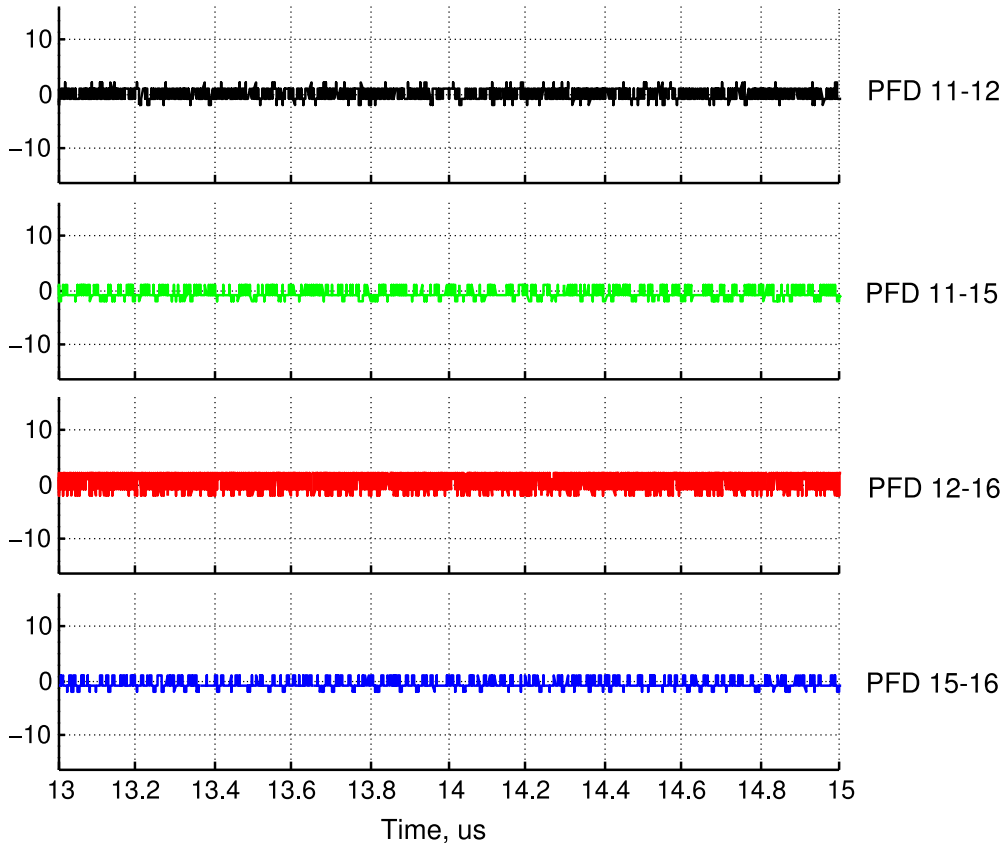


Figure 1.24: **Outputs of the PFDs in bidirectional configuration**: Nodes 11, 12, 15 and 16

delay in 65 nm technology. To improve this value, another TDC architecture should be used.

3. *Chip characterization: on-chip phase error measurement.* Last but not least, during the measurement process we found it very difficult to measure and compare output clock signals with picosecond precision in an off-chip way. The difficulties mainly come from the noise on chip pads, probe, test equipment and mismatch on routing. Actually, the off-chip measurement results provide pessimistic information about the phase error. An on-chip method is necessary for characterizing clock error and monitoring performance of large SoC. One example of this kind of circuit is the "Skitter" circuit used in IBM processors[13].

1.3 Original contribution of my PhD project

The main goal of this PhD project is the design of a 10×10 ADPLL network, complemented with an on-chip synchronization error measurement tool. The design is based on the architecture of the first prototype with reduced size, but addresses shortcomings of the chip mentioned in Subsection 1.2.10.

Firstly, a method aiming to reduce the residual error in steady state has been proposed. It allows choosing appropriate TDC quantization step and DCO frequency parameters in order to achieve a good performance with low implementation effort. Based on this study, we designed the ADPLL blocks for the clocking network. Especially, the optimal quantization step of TDC for the designed DCO is 20 ps, which is below the minimum delay of a logic gate in the used technology. To achieve this value, a TDC with a new structure is designed.

Secondly, due to the high power consumption of the first prototype (186.2 mA), the 4×4 network was difficult to be extended to a dimension of 10×10. To reduce the power consumption of the ADPLL network, we have studied the source of power dissipation. The energy hungry blocks in this system are the DCO and the loop filter. The DCO can be optimized by reducing the number of tuning cells thus the tuning steps (256 steps instead of 1024 steps). Less steps means less bits for the control word (8 bits instead of 10 bits), which also simplifies the loop filter.

In fact, the high power consumption of loop filter mainly results from arithmetic operation at high rate. In my PhD project, we have designed two different kinds of loop filters, which not only reduce the power but also accelerate the speed of error correction. The first filter is based on the conventional architecture of a proportional-integral (PI) filter with reduced delay in proportional path. And the size of arithmetic operators is smaller. The second filter is separating the frequency acquisition from phase tracking. At the steady state, only a few LSBs of the control word are updated for phase tracking. It is not necessary to recalculating the whole control word each cycle. The both architectures have been designed during the project, but only the former one has been implemented in the designed prototype of the network. This choice is motivated by a high risk which would be present if a novel architecture of a filter was chosen.

These designed blocks are assembled in a 10×10 network. This architecture is implemented at first on FPGA and then on silicium. Different topologies are explored based on this architecture.

For reasons explained in Subsection 1.2.10, an on-chip clocking error measurement circuit is necessary for the characterization of phase error between two clock signals in the network (generated in nodes either next to each other or in distance). The circuit designed in this PhD project is based on an indirect method easy to be implemented: instead of measuring the value of phase error each cycle, it allows calculating the minimum/maximum and

mean values of clocking error during a period. By using this circuit, a precise on-chip characterization of phase error is possible, thus it has a great importance in proving the scalability of ADPLL network. It has been designed as an analog IP and placed at 4 different locations of the chip.

In general, the structure of my work can be illustrated in Fig. 1.25.

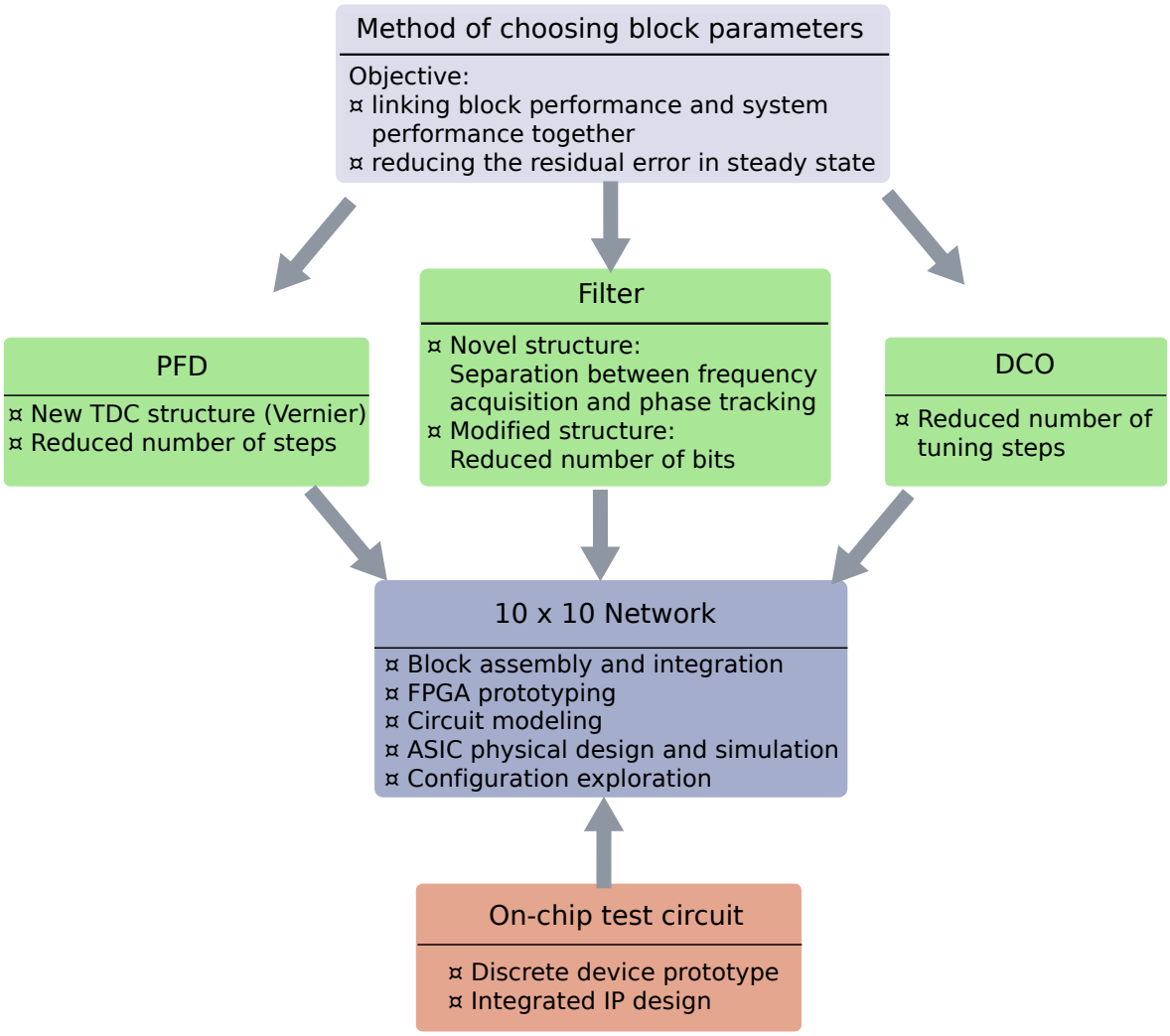


Figure 1.25: **Structure of work contribution**

1.4 Thesis outline

This thesis is composed of three parts. The first part comprising Chapter 2, Chapter 3 and Chapter 4 presents the ADPLL network from system level description to components implementation, from analytical study to physical design. This part is deployed in a top-down hierarchical way: The topology of proposed network is introduced in Chapter 2. The dependency of residual phase error on block parameters is also studied in this chapter. Then design of system components including clocking network cells and built-in test cell is detailed in Chapter 3 and Chapter 4. The second part comprising Chapter 5 to Chapter 6 presents two implementations of ADPLL network using the designed components. Implementation details and solutions to specific problems are also presented in this part. In the third part, including Chapter 7 and Chapter 8, two ADPLL networks with improved performance are explored. Both of the two circuits are based on the architecture presented previously but from two different points of view. The circuit presented in Chapter 7 proposes an innovative network topology in order to attenuate phase error propagation and reflection in the network. The method in Chapter 8 focuses on the improvement of one single ADPLL in terms of phase locking speed and power consumption, thus improving the ADPLL network performance in these two aspects.

The content of each chapter is described as follows.

Chapter 2 presents the studied system. It describes the topology of the ADPLLs network and impact of quantization in ADPLL on its operation in steady state, especially the value of residual phase error. A method is proposed to minimize the residual phase error at a lowest DCO and PFD circuit design effort. The chapter ends with a summary of the specifications for each block of the network.

Chapter 3 presents the design of the blocks measuring and processing the phase error: the digital phase frequency detector (PFD) and the digital proportional-integral filter. The local clock generation block — digitally controlled oscillator (DCO) is also presented in this chapter. The 4-bit PFD with 20 ps resolution is implemented with a Vernier delay line structure. The PI filter has a coefficient-programmable feature. The 0.9-1.16 GHz 8-bit DCO is based on a ring oscillator architecture. These three kinds of blocks are the basic components of the clocking network.

Chapter 4 explains the reason why a special circuit for clocking error characterization is useful, and how the analog built-in test circuit targeting at picosecond precision measurement is designed. This block allows giving the minimal/maximal and mean values of phase error between two clocks during a long test experience. This circuit uses differential technique so that it has a strong immunity to PVT variation.

Chapter 5 presents an FPGA prototype of a 10×10 network of ADPLLs. This prototype has two objectives: First, it allows validating the functionality of such a large network be-

fore the implementation in silicium; Second, with this prototype, we can study the relation between phase error and distance.

Chapter 6 explains in detail the implementation of the ASIC prototype in 65 nm technology. The chip consists in a distributed clock generator with a 10×10 dimension using blocks presented in Chapter 3. It also contains 4 test blocks presented in Chapter 4 for clock error characterization. Design methodology and implementation details are stated in this chapter.

Chapter 7 shows a distributed clock generator with a "swimming pool" like architecture. This architecture is based on the network structure presented in previous chapters but explores a new configuration of the connectivity between local clock generators. Simulation results have proved the improvement in attenuation of phase error propagation and reflection in the network.

Chapter 8 shows another ADPLL network designed during this PhD work. It uses the same network topology of prototypes presented in Chapter 5 and 6, but an innovative structure of digital filter. In this novel ADPLL, phase error correction functionality is separated from frequency tracking, which shortens the frequency acquisition time and decreases power consumption.

The report is finished by conclusions and perspectives for the development and research associated with proposed clocking approach. The academic contribution consists in the articles published and presented on international conferences.

Chapter 2

Network of distributed ADPLLs

Contents

2.1 Introduction 33
2.2 The architecture of clocking network proposed in this PhD project 35
2.3 Impact of quantization in ADPLL on its operation in steady state 36
2.4 Specification of the network 46
2.5 Conclusion 48

2.1 Introduction

As presented in the previous section, the objective of this PhD project consists in designing an ADPLL network with a large dimension (10×10) as well as an on-chip method of phase error characterization. From the first prototype we found that the power consumption of DCO and filter is too large so that these two blocks cannot be reused in a network with large dimension. To lower the power consumption, the DCO frequency range should be reduced, the structure of filter should be improved and the dynamic range of PFD can also be decreased so that the arithmetical calculation of the filter can be simplified. Moreover, since prior to the first prototype, no study has been carried out on the ADPLL behavior in the steady state, especially the dependency of residual phase error on block parameters, in the first prototype, the quality of synchronization in steady state is limited by the resolution of PFD.

The design of a DCO is a time consuming task. It was not the scope of my PhD. For this reason, in my work I used DCO IP cells available in my environment, designed in the frame of the HERODOTOS project (by LIP6 and by CEA-LETI). This chapter discusses the methodology of choosing the appropriate parameters of DCO, PFD and filter so to minimize the residual phase error at a low implementation cost.

This chapter is organized as follows:

In Section 2.2 we describe the topology of a 10×10 ADPLL network for clock generation.

Section 2.3 highlights the effects of quantization in the ADPLL and studies the link between block parameters and residual phase errors. A method is proposed to minimize the residual phase error at a lowest DCO and PFD circuit design effort.

A specification of the 10×10 ADPLL network is defined in Section 2.4 based on the method presented in Section 2.3 and previous stability study in Section 1.2.8.

2.2 The architecture of clocking network proposed in this PhD project

The proposed system has a dimension of 10×10 providing clock signals for as many as 100 local synchronous clocking areas (from SCA1-1 to SCA10-10).

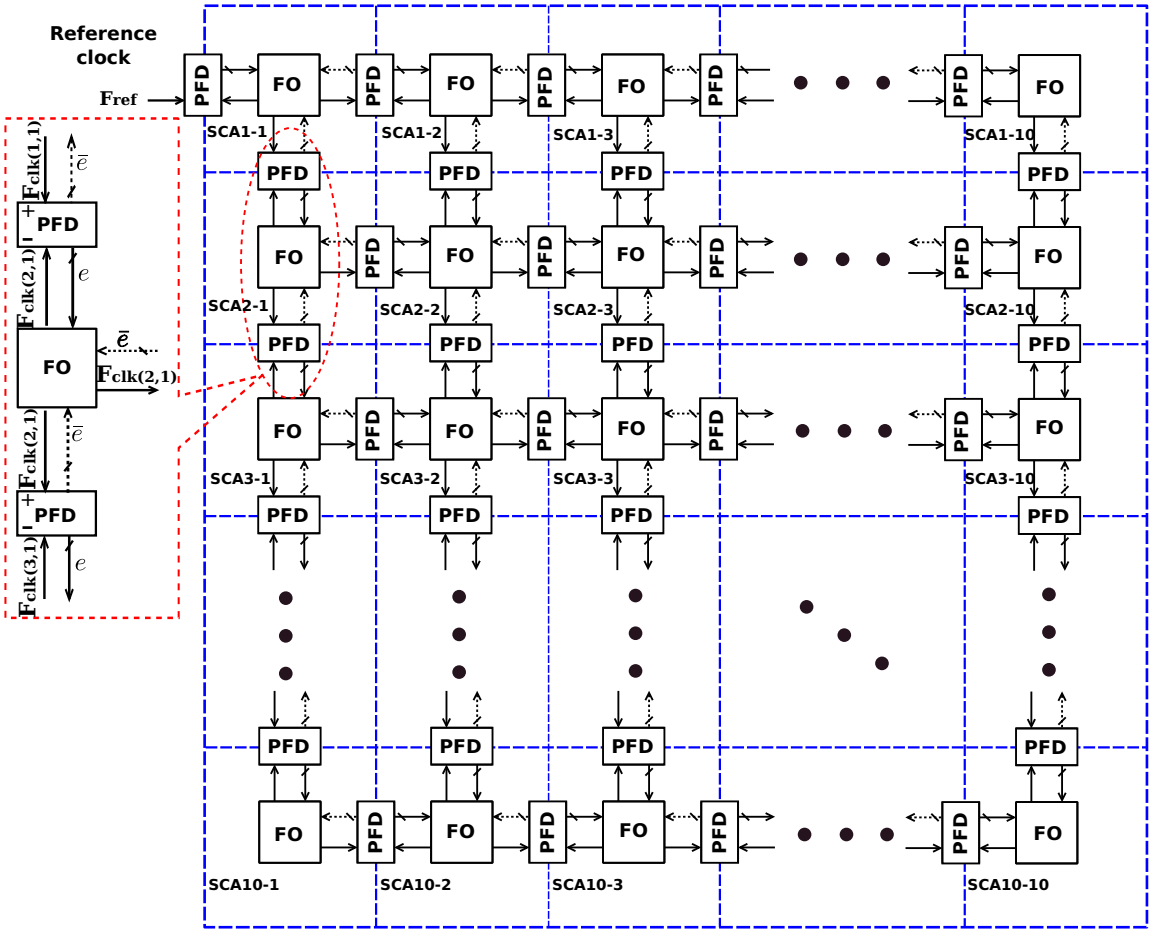


Figure 2.1: **Clocking network architecture**

From Fig. 2.1 we can observe the FO block (filter + oscillator) in each SCA, generating the clock signal for the local isochronous zone. A PFD at the border between two neighboring zones compares two clock signals and send back the phase error between them with the same absolute value but opposite signs (e and $\bar{e}$).

Before the implementation of this architecture, it is necessary to choose the appropriate values of the block parameters. The following section in this chapter provides a study on the link between the quality of synchronization of ADPLL (the residual phase error in steady state) and the parameters of its blocks.

2.3 Impact of quantization in ADPLL on its operation in steady state

Previous study presented in Subsection 1.2.8 focuses on the global stability of system, and not on the optimization of its operation in steady state when the whole system converges. Moreover, these studies were based on an analog model of the system, which did not account for quantization. Indeed, in an analog PLL network with 2 integrators in each node, in absence of noise, the residual error is zero[39]. However, it is not the case with all digital PLLs, because the PFD and DCO, which implement signal conversion between the analog and digital domains, introduce unavoidable quantization errors. Moreover, the digital filter operates at fixed point arithmetic, and rounding code applied to the DCO may also be a source of errors.

The study presented in this section provides a method aiming to minimize the absolute phase error between reference clock and divided DCO clock in steady state at a lowest DCO and PFD circuit implementation cost. Although this study is based on the model of one single PLL, it has an instructive meaning for the estimation of the quality of quantization in an ADPLL network.

This method presented in the following three subsections is summarized as follows. First, the analysis presented in Subsection 2.3.1 estimates the relation between the residual error in steady-state and the parameters of the DCO and PFD individually. This analysis allows a rough selection of DCO and PFD.

Afterwards, in Subsection 2.3.2, the effect of quantization in the digital filter is highlighted and a solution is proposed. Thanks to this solution, for a target loop gain and a given DCO, we can estimate the required PFD resolution.

Then in the third step (cf. Subsection 2.3.3), behavior VHDL model considering quantization in the ADPLL is created. Time simulations using this model can show the residual error in steady state with different block parameters. Simulations validate our study in Subsection 2.3.1 and Subsection 2.3.2 and help estimating the optimal combination of parameters, which can minimize the residual phase error.

2.3.1 Step 1: Impact of PFD and DCO quantization steps on the residual error

In this step, we study the PFD and DCO blocks separately. First, we study the limit imposed by the finite resolution of PFD. In an ADPLL system, the residual phase error in steady state does not exceed two steps of TDC resolution (cf. Subsection 1.2.10). Hence, in the best case (optimal operation of the DCO and filter), the phase error expressed in time units between neighbors $\Delta\tau$ will be in the following interval:

$$-2\Delta T_{TDC} \leq \Delta\tau \leq 2\Delta T_{TDC} \tag{2.1}$$

where ΔT_{TDC} is the TDC quantization step value in second. We define a parameter $\Delta\tau_{PFD}$ which characterizes the maximal absolute phase error expressed in time unit due to quantization of PFD in the best case:

$$\Delta\tau_{PFD} = 2\Delta T_{TDC} \tag{2.2}$$

In this analysis we express the phase error in the time units, since such a phase error is invariant with regard to the frequency division.

Now we study the limit imposed by the quantization of the DCO characteristic. The objective of a PLL is to equalize the phase of the feedback signal coming from the DCO and divider, on the signal of the input reference (Fig. 1.6). It means that the PLL feedback signal have the same frequency as the input reference signal. However, whereas the input signal can have any frequency belonging to some continuous interval, the DCO generates a signal whose frequency belongs to a discrete grid of values. When the DCO signal passes through a divider, the frequency values are downscaled, but remain discrete. Hence, if the PLL input (reference) frequency is between two neighboring values of the divided grid, the PLL closed loop generates a variable input DCO code, so that the output frequency of the DCO switches permanently between the two corresponding values of the grid. This is done with the help of feedback loop which minimizes the phase error.

In the analysis which will follow, we will estimate the maximal synchronization error introduced by the DCO quantization in the best case, with an ideal input command of the DCO, which will also be defined in the analysis.

Fig. 2.2 shows the accumulated phases of reference clock and divided DCO clock. We suppose that the frequency of reference clock is f_0 (ω_0 in radian), which corresponds to a linear phase evolution. The phase of the divided DCO output signal must approximate as close as possible this trajectory. However, the phase of the divider output can only follow the lines with discrete slopes defined by the DCO divided frequency grid. The slope (the frequency) can change in function of the input DCO command word arriving with cadence f_s, where f_s is the clock frequency of the PLL digital filter (cf. Fig. 1.6). If the input reference frequency is not exactly one of the divided DCO frequency grid values, in the steady state of convergence, an ideal control loop switches the divided output frequency of the DCO between two neighboring values f_1 (ω_1 in radian) and f_2 (ω_2 in radian) with a cadence f_s, so to minimize the tracking (quantization) error. The maximum tracking error is at the instants when DCO changes its frequency ($\Delta\phi_1$ and $\Delta\phi_2$ in Fig. 2.2).

Now we search the optimal command the DCO should receive at its input in order to minimize the tracking error. For simplicity, we limit the demonstration to the case where

$$(\omega_2 - \omega_0)/(\omega_1 - \omega_0) = P, \tag{2.3}$$

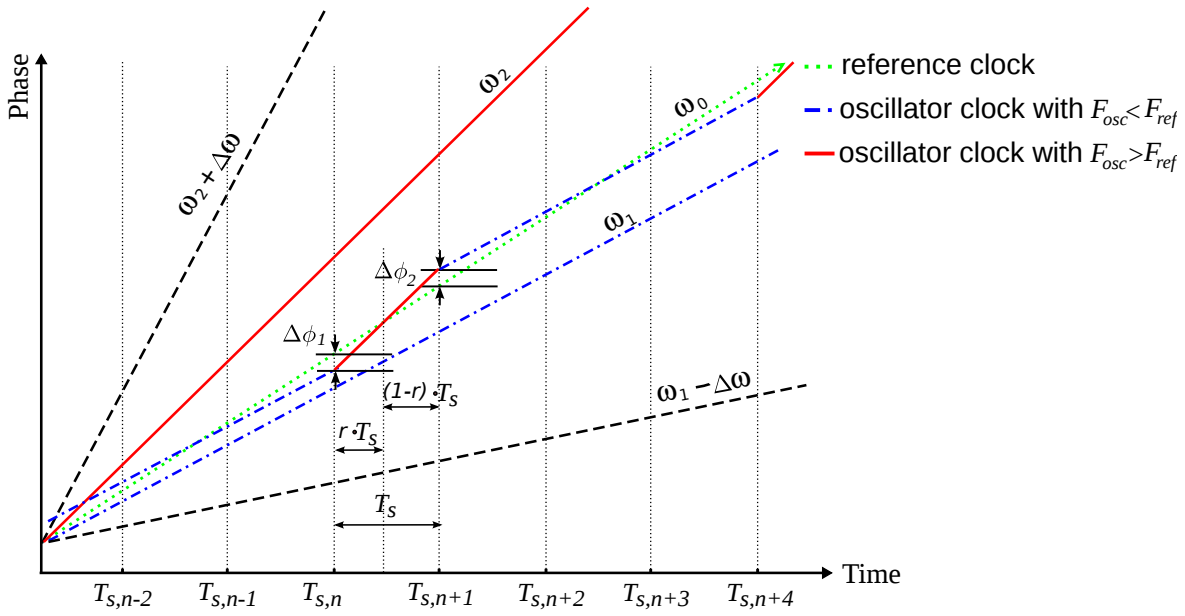


Figure 2.2: **Phase evolution of reference clock and divided oscillator clock**: $\omega_1 < \omega_0 < \omega_2$

where P is an integer superior or equal to 1.

Fig. 2.2 presents an ideal trajectory of the phase of DCO output. The DCO output stays at frequency ω2 during one clock cycle T_s (T_s is the inverse of the frequency of digital filter f_s), and at ω_1 during P clock cycles. Evidently, the phase of the DCO output will present a periodic signal with period $(P+1)T_s$.

We focus on a segment of T_s during which the DCO output frequency is ω_2. The crosspoint of the the segment and the trajectory of the reference divides the time interval into two parts: $r \cdot T_s$ and $(1-r) \cdot T_s$, where $0 < r < 1$. The phase error between the DCO output and the reference signal is the largest at the beginning and the end of the segment. We define the two values as $\Delta\phi_1$ and $\Delta\phi_2$ as shown in Fig. 2.2, and we look for the optimal r for any ω_1, ω_2 and ω_0 complying with Eq. (2.3). The position of crosspoint and the difference between two frequencies decide the phase error absolute values in radians:

$$\begin{aligned} \Delta\phi_1 &= (\omega_2 - \omega_0) \cdot r \cdot T_s \\ \Delta\phi_2 &= (\omega_2 - \omega_0) \cdot (1-r) \cdot T_s \end{aligned} \tag{2.4}$$

The maximum among these two values is the smallest if $r = 0.5$. Hence, we have for the maximal phase error in the ideal case:

$$\Delta\phi = \frac{1}{2}\frac{\omega_2 - \omega_0}{f_s} \tag{2.5}$$

During the operation of a PLL, the value of ω_0 can be any between ω_1 and ω_2, by conse-

quence, the error Eq. (2.5) is the largest when ω_0 is close to ω_1. In this way, in the ideal case, a DCO is able to synchronize its output with a reference signal with precision given by:

$$\Delta\phi_{DCO} = \frac{1}{2}\frac{\Delta\omega_{DCO}}{f_s} \tag{2.6}$$

where $\Delta\omega_{DCO}$ is the tuning step of DCO in radian/second.

This quantity can be expressed as a relative phase error normalized to the period:

$$\mathrm{err}_{DCO} = \frac{1}{2}\frac{\Delta f_{DCO}}{f_s} \tag{2.7}$$

where Δf_{DCO} is the tuning step of DCO in Hertz. It is also interesting to express the maximal error in the time units :

$$\Delta\tau_{DCO} = \frac{1}{2}\frac{\Delta f_{DCO}}{f_s}\frac{1}{f_{DCO}} = \frac{1}{2}\frac{\Delta f_{DCO}}{f_{DCO}}T_s \tag{2.8}$$

where f_{DCO} is frequency of DCO output. The parameter $\Delta\tau_{DCO}$ is useful, because it demonstrates that the timing error is independent of frequency divider of the DCO output. Indeed, in this case, both f_{DCO} and Δf_{DCO} are divided.

Eq. (2.2) and Eq. (2.8) provide the *optimistic* estimation of the largest synchronization error which an ideal ADPLL may achieve. In practice, the error will obviously be greater, but the knowledge of this principal limit is important for the selection of PFD and DCO.

The equations show that the maximum phase error is function of filter clock frequency, in other words, the speed at which the DCO input refreshes. Evidently, if we raise the filter frequency, the maximum phase error will be smaller. However, it demands high processing speed of the filter, which makes the timing constraint more stringent. In practice, the output signal of the local DCO (divided by a power of 2 ratio) is used for the filter clocking. We keep the ratio 4 between the output DCO frequency and the filter clock frequency, as in the previous prototype of the network. Moreover, the above analysis highlights that to minimize the phase error, the DCO input word should switch between two neighboring values (cf. Subsection 2.3.2).

Three available DCOs were designed in my environment (cf. Tab. 2.1): one of them issued from CEA-LETI (the partner of our project), the second one used in the first prototype, and the third one with reduced tuning range of frequency designed in my group on the base of the previous prototype. One of them should be selected for the prototype under development.

Tab. 2.1 displays the characteristics of these DCOs. Using these known parameters (Δf_{DCO}, f_c and f_s), we can calculate the maximum residual phase error ($\Delta\tau_{DCO}$) by using Eq. (2.8).

Table 2.1: **Characteristics of DCOs and maximum residual phase errors due to DCO quantization**

	Mean frequency step Δf_{DCO}	Nominal frequency f_c	filter clock frequency $f_s = f_c/4$	$\Delta\tau_{DCO}$
DCO 1[a]	3.91 MHz	742 MHz	186 MHz	14.18 ps
DCO 2[b]	741 KHz	872 MHz	218 MHz	1.95 ps
DCO 3[c]	1.01 MHz	1.04 GHz	260 MHz	1.87 ps

[a] DCO designed by CEA Leti

[b] DCO designed by LIP6 for the first prototype of ADPLL network

[c] DCO designed by LIP6 for the second prototype of ADPLL network

A PFD with a resolution of 30 ps was implemented in the first prototype. From Eq. (2.2) we can get the maximum residual error due to PFD quantization ($\Delta\tau_{PFD}$) is 60 ps. Tab. 2.2 shows also the values of $\Delta\tau_{PFD}$ for resolutions of 20 ps and 10 ps.

Table 2.2: **Maximum residual phase errors due to PFD quantization**

	PFD 1 (30 ps/LSB)[1]	PFD 2 (20 ps/LSB)	PFD 3 (10 ps/LSB)
$\Delta\tau_{PFD}$	60 ps	40 ps	20 ps

[1] Used in the first prototype of LIP6

Now we can compare $\Delta\tau_{DCO}$ and $\Delta\tau_{PFD}$ to estimate whether a PFD is suitable with a given DCO in a PLL. It should be mentioned that $\Delta\tau_{DCO}$ is calculated based on an ideal PLL control: no delay in the loop, DCO switches between two adjacent frequency grids, etc. The maximum residual phase error limited by DCO quantization in practice should be larger than $\Delta\tau_{DCO}$ displayed in Tab. 2.1.

The DCO and PFD blocks used in the first prototype are DCO 2 in Tab. 2.1 and PFD 1 in Tab. 2.2. Apparently, $\Delta\tau_{DCO2}$ is a very small value compared with $\Delta\tau_{PFD1}$. In other words, in the first prototype, DCO is over-precise with respect to PFD. Hence, in this PhD project, we can release the design effort on DCO properly and try to improve the PFD resolution, for example, to 20 ps (PFD 2 in Tab. 2.2). Moreover, we can observe from Tab. 2.1 that $\Delta\tau_{DCO}$ of DCO 1 has a large value (14.18 ps), which is almost the same as the resolution of PFD 2 (20 ps). Thus the tuning step of DCO 1 may not be precise enough. By consequent, the combination of DCO 3 and PFD 2 may be a good choice. This is to be verified by further study in the next subsections.

2.3.2 Step 2: impact of rounding in digital filter on the correction of residual phase error

This section presents the problems related to the rounding in the digital filter of the ADPLL, and proposes some solutions. As in previous subsection, the analysis will be demonstrated

on a single ADPLL using the analytical model presented in Subsection 1.2.7, but the results are also valid for a network of ADPLLs.

The filter receives integer words, and the input of the DCO is integer as well (arbitrarily, it is possible to define fractional and integer part of the DCO word, but the simplest choice is to consider DCO input as an integer). However, in the filter, the proportional coefficient α and integral coefficient β (as shown in Eq. (1.2)) are generally not integers, but as said in Subsection 1.2.5, the output of the proportional and integral branches are rounded.

The rounding of the integral branch is fundamentally harmful, since it limits the precision of the output frequency definition. But the origin of this rounding is not in the integral branch itself, but in the limited width of the DCO input word, (equivalently, in the limited DCO resolution). The discretization the DCO frequency value is a fundamental property of an all-digital PLL, which is supposed to be corrected thanks to the modulation of the input DCO word between two neighboring values.

However, the most important danger for the ADPLL operation is the rounding in the proportional part. Imagine that α is smaller than 1. In this case, small phase error x_i such that $x_i \cdot \alpha < 1$, yields a zero output of the proportional path, as if α was zero. Hence, small error cannot be corrected by proportional path, and it is well known that a PLL with a zero proportional path oscillates [14]. In the context of digital PLL, a small α leads actually to a *variable* α: for large errors it has its nominal value, but for small errors it is zero. Such an ADPLL exhibits an output phase error oscillating with amplitude corresponding to the limit between the modes "rounding of small error" and "correct detection of big error".

In contrast, an α larger than 1 is equivalent to a normalized α with an enlarged DCO tuning step (a reduced resolution). For instance, if α equals 2, in steady state, instead of switching between two adjacent frequency values, the DCO changes at least two steps each time. In this case, the high resolution of the designed DCO is wasted.

For this reason, the best choice for α from the point of view of steady-state operation is around 1.

However, α impacts not only on the steady-state mode, but also on the dynamics of the PLL in transient mode, stability, etc. These characteristics are usually studied on a LTI discrete time model presented in Subsection 1.2.7. There are only two parameters defining the dynamic properties of such a system: the ratio between α and β and the PLL gain.

$$K = K_{\mathrm{PFD}} \cdot \alpha \cdot K_{\mathrm{DCO}} \qquad rad/sec, \tag{2.9}$$

where K_{PFD} is the gain of PFD, which is reversely proportional to ΔT_{TDC}. K_{DCO} is the gain of DCO, which is equal to its frequency tuning step.

For a single ADPLL, these parameters are found by standard methods of the control theory. For a network of ADPLL, extensive studies were made in the frame of the HODISS

project at Supelec and CEA-LETI, cf. Subsection 1.2.8.

The parameters (K and β/α) which define the transient dynamics of the ADPLL can be achieved by different combinations of ΔT_{TDC}, K_p, Δf_{DCO}. However, due to the rounding in the digital loop filter, they may have different characteristics in steady state. This is discussed in the next subsection.

As far as these parameters are defined, the sizing procedure is the following: α is fixed to 1, and the gains of the PFD and DCO are chosen so to obtain the target value of gain. Note that the choice of gains of PFD and DCO (actually, their resolutions) must consider the steady-state mode behavior, (cf. Subsection 2.3.1). The integral coefficient is then chosen. Usually, the order of magnitude of the ratio β/α is a small fraction of unity.

We now illustrate the presented analysis on a example of a single ADPLL. First, we study it with standard tools of the control theory, with use of an LTI discrete time model (cf. Subsection 1.2.7). We select the appropriate gain K=1.2e7 rad/s and $\beta/\alpha = 0.012$. This yields the phase margin around 68 degree and the gain margin around 13 dB (Fig. 2.3), which corresponds to a satisfactory transient process.

Usually, DCO is the most complex block for the design. For this reason the sizing of the ADPLL parameters starts from fixing the characteristics of the DCO (the frequency step Δf_{DCO}). Since α is fixed at 1 and K is known from the LTI model analysis, the required TDC gain is calculated from Eq. (2.9). If the DCO 3 is chosen (cf. Tab. 2.1), the required TDC resolution is about 20 ps.

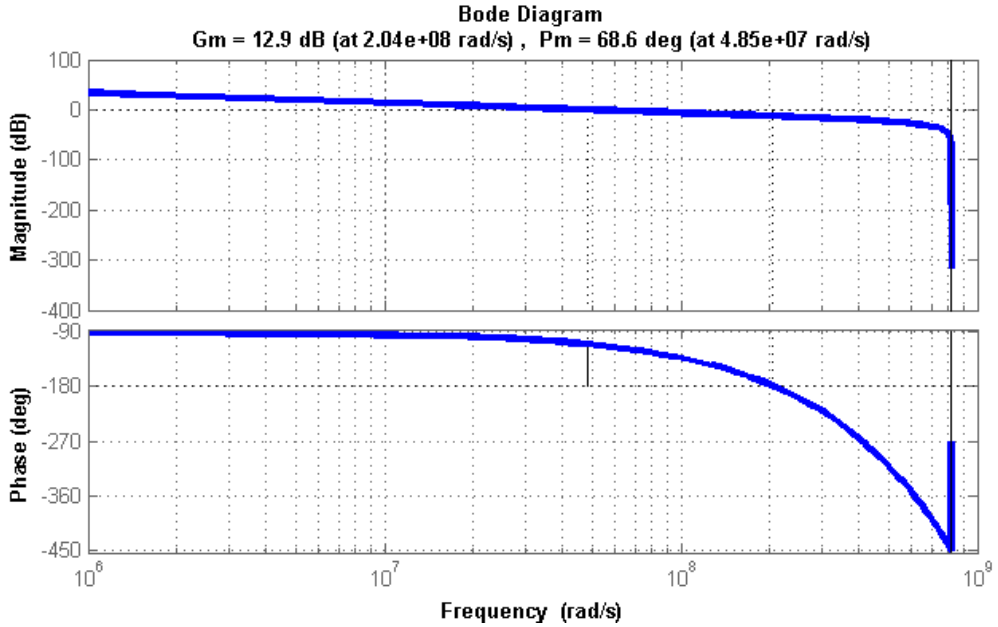


Figure 2.3: **Bode diagram of the LTI model of the system (Fig. 1.18)**: $K = 1.2e7$ rad/s and $\beta/\alpha = 0.012$

2.3.3 Step 3: validation of block parameters by transient simulations

The goal of this step is a validation of the study carried out at steps 1 and 2. This validation is done by behavioral modeling accounting for the quantization and for self-sampling. This model is implemented at VHDL language, as explained in Subsection 1.2.7. This validation is necessary, since the steps 1 and 2 provides only an estimation of the PLL parameters, on the basis of simplified LTI model. A more precise VHDL model is used to validate the obtained value of parameters, or eventually to correct them.

To validate the choice of the PFD gain after fixing the DCO parameters, we propose two modeling experiment.

1) We fix the proportional coefficient α as 1 (for reasons explained in Subsection 2.3.2), and we try different PFD resolutions seeking for the smallest error in the steady-state operation. The simulation results for three values of TDC resolution are given in Fig. 2.4.

Here we choose DCO 3 in Tab. 2.1 and filter parameters $\alpha = 1$, $\beta = 0.012$. When $\Delta T_{TDC} = 30ps$, the PFD is not precise enough compared with DCO. Although its PFD output is within ± 2, the residual error exceeds 40 ps. When $\Delta T_{TDC} = 20ps$, the maximum residual phase error has a value less than 40 ps as show in the zoomed figure from 7.3 μs to 7.9 μs in Fig. 2.4. However, $\Delta T_{TDC} = 10ps$ is an over high resolution, the DCO cannot generate a frequency precise enough to follow the phase correction. The steady-state error is far above 40 ps. Hence, we can estimate that 20 ps is an optimal value of PFD resolution, and the loop gain has an appropriate value with this group of parameters.

2) We fix the DCO parameters and the ADPLL loop gain K, and we perform three simulations with different PFD resolutions (10 ps, 20 ps, 40 ps), and with β/α fixed. In order to maintain the same gain for the three simulations, the β and α are adjusted. As shown in Fig. 2.5, all three ADPLLs have similar stability and transient process, but only the ADPLL ($\Delta T_{TDC} = 20$ ps, $\alpha = 1$, $\beta = 0.012$) has a residual phase error always smaller than 40 ps.

Indeed, for ADPLL ($\Delta T_{TDC} = 10$ ps, $\alpha = 0.5$), due to the "rounding of small error", small PFD output codes (± 1) are considered zero by the proportional path, the PLL is undamped and operates as a second order oscillator. The maximum PFD outputs in steady state are ± 6.

For ADPLL ($\Delta T_{TDC} = 40$ ps, $\alpha = 2$), phase error are amplified by the digital filter, so the output of filter for DCO varies in a larger range (99$\sim$112) compared with ADPLL with $\alpha = 1$ (104$\sim$107). The phase error in steady-state is much larger than for the two other TDCs.

It should be noted that the time of frequency acquisition is different for the three ADPLLs. This is because of a limited dynamic range of phase detection in the PFD. At the beginning of the frequency acquisition stage, since the phase error is large, the output of PFD saturates at ± 15 (5-bit signed). The filter with a larger α and β has a faster correction speed.

Through these twos sets of simulations (Fig. 2.4 and Fig. 2.5), we have validated our study in the two previous steps on the impacts of PFD, DCO and filter parameters on the residual error in steady state. We confirmed that the PFD with $\Delta\tau_{PFD} = 20ps$ is the optimal choice.

The results of this study can be represented graphically, so to put forward an existence of an optimal value of the PFD resolution for a given DCO.

Fig. 2.6 traces the maximum residual phase error with different PFD and filter parameters, for two DCOs (DCO 2 and 3). The curves are obtained in the following way. On the right of the point labelled as "saturation point", we fix α at 1 as simulations in Fig. 2.4, and only the TDC resolution are varied. On the left of the saturation point, we keep the same loop gain as simulations in Fig. 2.5. The saturation point indicates the optimal PFD resolution for a given DCO. For DCO 3, this value is about 23 ps, while for DCO 2, it is about 10 ps.

Similar simulations can also be performed on a network of ADPLL. After a consideration of steady state behavior and circuit complexity, power, etc., we decided to design a PFD with a resolution of 20 ps for DCO 3. The detailed specification is displayed in the next section.

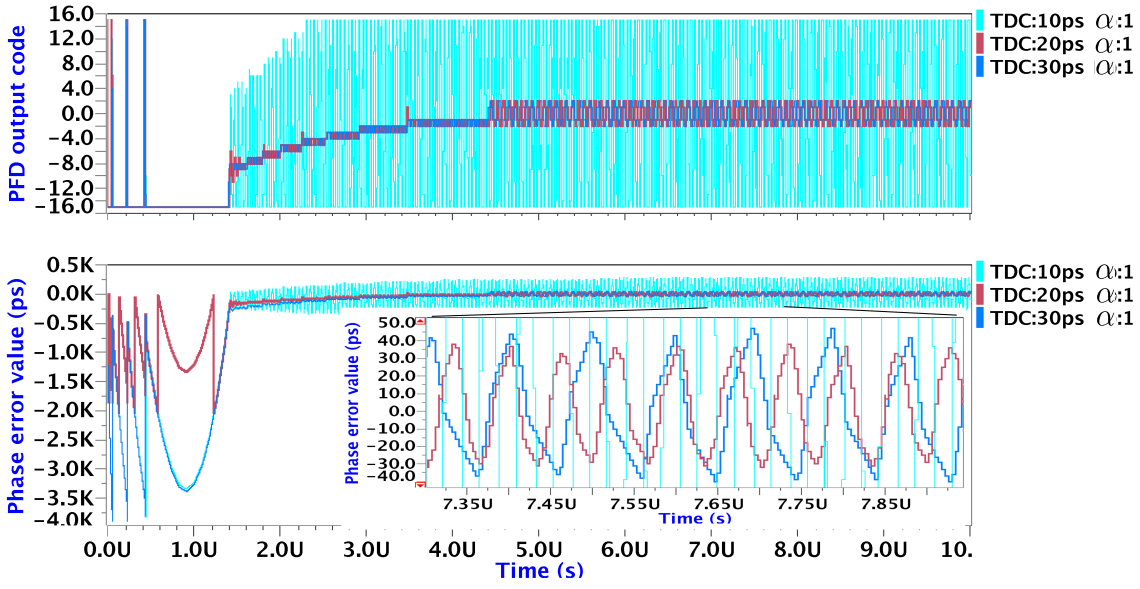


Figure 2.4: **Time simulations with different TDC resolution and same filter/DCO parameters (VHDL model)**: reference clock frequency: 249.5 MHz, DCO 3 in Tab. 2.1 used

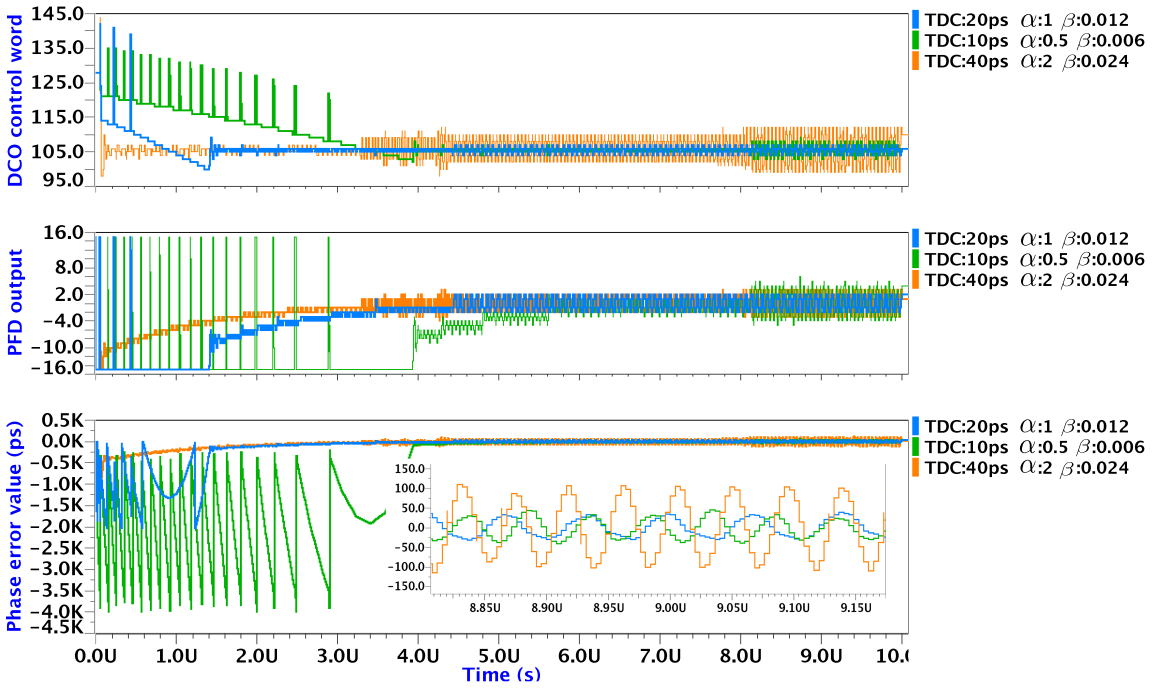


Figure 2.5: **Time simulations with the same loop gain and β/α ratio (VHDL model)**: reference clock frequency: 249.5 MHz, DCO 3 in Tab. 2.1 used

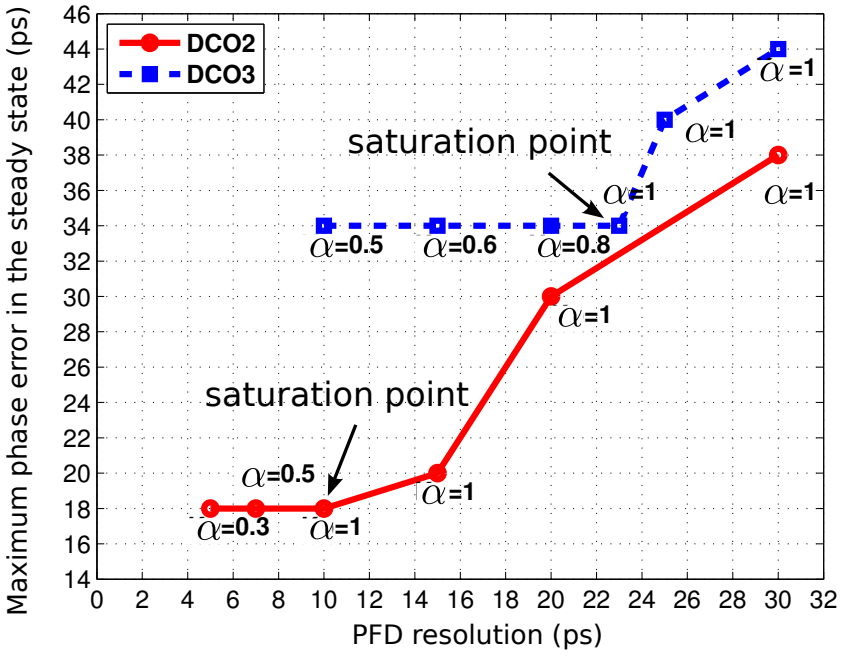


Figure 2.6: **The maximum values of residual phase errors of an ADPLL in the steady state with different PFD and filter coefficients (VHDL model)**

Table 2.3: 1^{st} **network test chip characteristics summary**

Parameter	***Value***
Central frequency of the SCA	870 MHz
Frequency range	550∼1190 MHz
DCO gain	1.5 MHz/LSB
DCO control word width	10 bits
PFD resolution	∼30 ps
PFD output bit number	5 bits
Filter integral path bit number	12 bits
Filter proportional path bit number	5 bits
Timing error [1]	$<$ 60 ps
Convergence rate [2]	$\approx$ 5 MHz/μs
Supply voltage	1.2 V
Power consumption [3]	186.2 mW @ F_{clk} = 800 MHz
Clocking core area	$\sim$ 0.72 mm^2
Chip area	$\sim$ 2.04 mm^2

[1] between neighbor nodes
[2] with coefficients $\alpha = 1.0 \pm 10\%$, $\beta = 0.0028 \pm 10\%$
[3] analog and digital

2.4 Specification of the network

The first ASIC prototype of a 4×4 ADPLL network has been designed prior to this study. This prototype has the characteristics summarized in Tab. 2.3. As analyzed in Section 2.3, the ADPLL used in this prototype has an over precise DCO and the residual phase error is limited by the resolution of PFD.

During this PhD work, a 10×10 prototype has been designed in order to study the clocking network in a large circuit, and to observe especially the phase error between two clock signals in distance. Before the design of this circuit, a specification has been defined based on the method presented in Section 2.3 and previous stability study in Section 1.2.8.

In this specification, we did the following modifications compared with the first prototype: First, we improve the resolution of PFD to 20 ps and reduce its output to 4 bits; Second, we reduce the DCO tuning range from ± 37% to ± 10-20%; Third, less number of bits are used for filter coefficient programming. Fourth, since the DCO tuning range is reduced, less number of bits can be used for DCO control word (8 bits is decided after analysis). Tab. 2.4 summarizes the specification of the second version test chip design.

Table 2.4: 2^{nd} **network test chip specification**

Parameter	***Value***
Central frequency of the SCA	1 GHz
Frequency range	central frequency $\pm$ 10 - 20%
DCO gain	1 - 1.5 MHz/LSB
DCO control word width	8 bits
PFD resolution	$\sim$20 ps
PFD output bit number	4 bits
Filter integral path bit number	8 bits
Filter proportional path bit number	2 bits
Supply voltage	1.2 V
Power consumption	$<$ 1 W @ central frequency

2.5 Conclusion

In this chapter we have introduced the architecture of the ADPLL network designed in this PhD project. It is a 10×10 network based on the principle presented in Chapter 1.2.1. Previous work on the first test chip has justified the feasibility of this approach and has also shown some problems as explained in Chapter 1.2.10. Section 2.3 in this chapter addressed the problem of block parameter limitation on the residual phase error and proposed a method on three steps to minimize the residual error at low cost.

Finally, we have derived specifications for the implementation of clocking network, which are used as input data for design of each block. The implementation of blocks is detailed in the following chapters.

Chapter 3

ADPLL blocks design

Contents

3.1 Phase frequency detector (PFD) 50
3.2 Digital filter in the ADPLL network 57
3.3 Digitally controlled oscillator (DCO) 60
3.4 Conclusion 75

This chapter present the transistor-level design of the blocks composing the network of ADPLL: the phase frequency detector (PFD) (Section 3.1), the digital filter (Section 3.2), the digitally controlled oscillator (DCO) (Section 3.3) and the serial programming interface (Subsection 3.3.4).

3.1 Phase frequency detector (PFD)

3.1.1 The digital PFD architecture

The structure of the PFD designed in the second prototype is given in Fig. 3.1. The difference between the first prototype is that the TDC receives the input signal, and not the MODE signal. The task of the bang-bang detector is only a detection of the sign of error. The output of the TDC is then combined with the sign bit to obtain a signed 2-complement digital word. As in the first prototype, the PFD has a linear characteristic (cf. Fig. 3.1(b)).

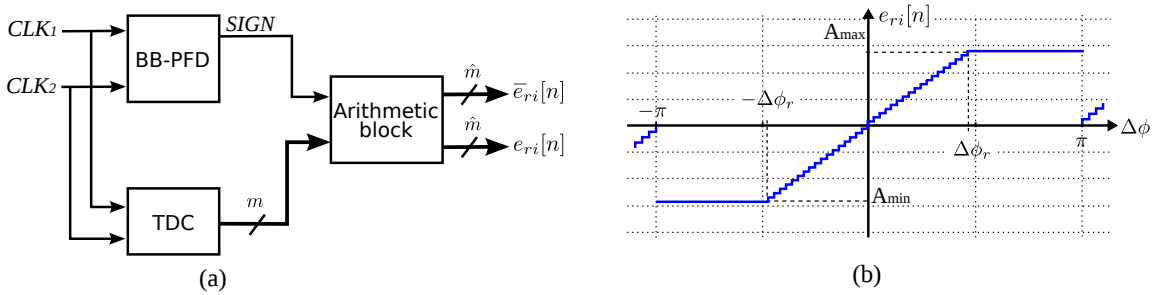


Figure 3.1: **Proposed phase/frequency detector for clock network**: (a) block diagram and (b) transfer function

3.1.2 Improvement of time-to-digital converter

In this subsection we discuss the architecture of TDC allowing a measurement of the absolute timing error ΔT between two clock signals.

TDC architecture

The TDC used in the first test chip has an flash ADC architecture, implemented with a single delay chain(cf. Fig. 1.14). Such a TDC has a quantification step of the same order as the elementary buffers delay available in the technology, which is about 30 ps for CMOS 65 nm. As analyzed in Chapter 2, 20 ps is required in order to optimize the system operation.

Another modification in the original TDC architecture concerns the dynamic range. In the first prototype, it was (0-450 ps), which corresponded to a phase error of $\Delta\phi_r = 0.4 \cdot 2\pi$ if the nominal output DCO frequency is 1000 MHz. However, if the network operates properly, the phase error will be few percents of the period, and the role of the PFD will be to measure small errors. hence, such a big linear range of PFD is not useful. For this reason, the parameter $\Delta\phi_r$ was chosen to be $2\pi/32$. For the nominal DCO output frequency 1 GHz, $\Delta\phi_r$ correspond to 125 ps. This range is obtained with 6.25 measurement steps: we round this value down to 7, so to allow the output TDC word to be coded by 3 bits. When output of the TDC is combined with the sign of the phase error, one bit is added, the output PFD word has 4 bits and the real dynamic range is ± 140 ps. Consequently, the PFD designed for this test chip is a 4-bit detector with a resolution of 20 ps.

To obtain the desirable resolution of TDC, we designed an architecture based on Vernier delay line (Fig. 3.2) inspired by the work of [11]. The Vernier delay line utilizes two delay chains (Fig. 3.2(a)). One delay chain (the left one in Fig. 3.2(a)) consists of buffers with delay τ_1 while the other delay chain has buffers with delay τ_2. The time interval to be measured ΔT is defined by the starting and ending events given by the rising fronts of the signals P_2 and P_1. Assume that P_2 comes before P_1 and τ_1 is larger than τ_2. As signals P_2 and P_1 propagate through the delay lines, the time difference between them is decreased by $T_R = \tau_1 - \tau_2$ after each stage. The position in the delay line, at which the delayed P_1 signal catches up the delayed P_2 signal defines the time difference between the original P_1 and P_2 signals with a resolution of T_R. The principle of this operation is depicted in Fig. 3.2(b). The resolution of this TDC does not depend on the delays of the unit elements used in the delay chains, but rather on their difference. Therefore, time intervals that are smaller than a single inverter delay in a given process technology can be measured. In order to have 7 measurement steps, 6 stages are necessary. The TDC outputs values are from 0 (the error is less than $\Delta\tau_{TDC}$) to 6 (the error is greater than $6\Delta\tau_{TDC}$).

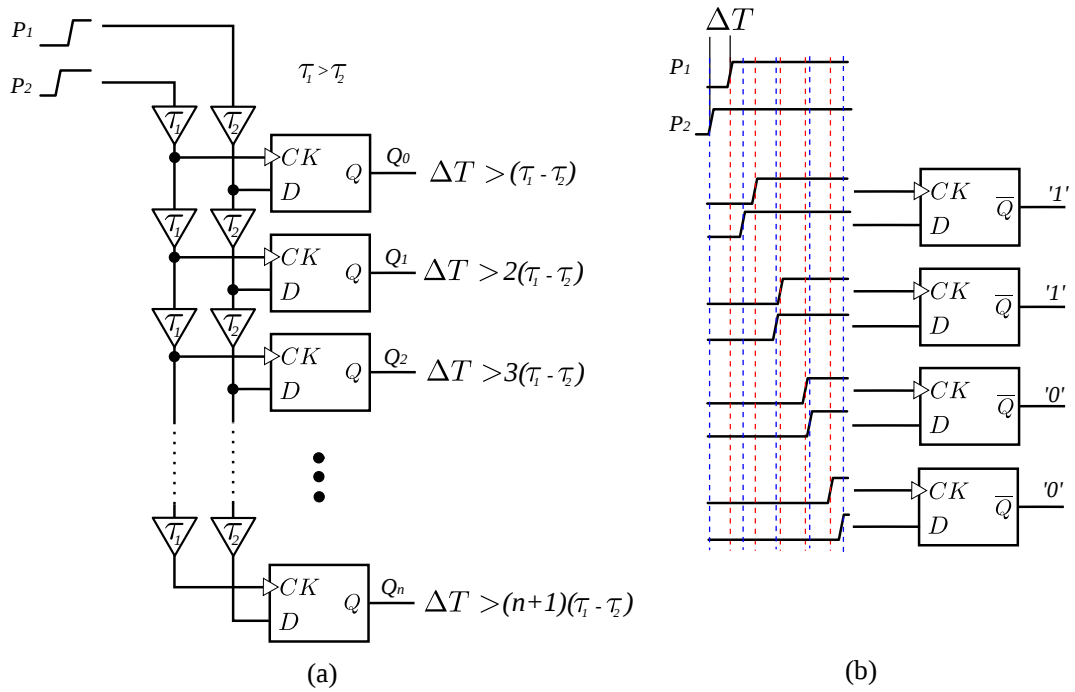


Figure 3.2: **Time-to-digital converter based on Vernier delay line**: (a) block diagram and (b) principle of operation

As for any quantifier, there is a risk of metastability if after a certain number of delay stages, the value of the interval is reduced to a value within the setup/hold timing window of the output register. Moreover, the rising edges of *CK* of sampling registers in Fig. 3.2 do not arrive at the same time. They are the same signal with different delays, thus with skews. This results in glitches at the output of PFD.

To solve these problems, we propose the circuit displayed in Fig. 3.3. The solution relies on two points: First, to observe which signal arrives first at each stage, instead of using

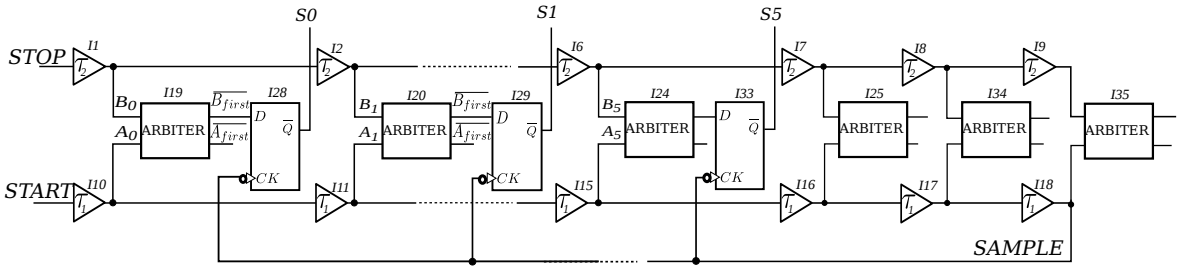


Figure 3.3: **Structure of proposed time-to-digital converter**

one of them to sample the other, we compare the two delayed signals by the arbiter circuit used for BB block (cf. Fig. 1.13). This helps filtering metastability during the comparison; Second, to avoid glitches at the output of TDC, the outputs of all arbiters are sampled at the same time by the signal named *SAMPLE*. The generation of *SAMPLE* is explained later in this section.

As shown in Fig. 3.3, the entry of the delay line (τ_1) is a signal called *START*, whose falling edge signifies the beginning of the interval whose duration is to be measured. The other delay line (τ_2, $\tau_2 < \tau_1$) has an entry *STOP*, whose falling edge marks the ending of the interval. The two signals for the delay line inputs are generated from the clocks whose phases are being compared:

$$\begin{aligned} START &= \overline{ref \ \text{OR} \ div} \\ STOP &= \overline{ref \ \text{AND} \ div} \end{aligned} \tag{3.1}$$

The operation of the TDC is presented on an example of the time interval Δt such that $\Delta\tau_1 - \Delta\tau_2 < \Delta t < 6(\Delta\tau_1 - \Delta\tau_2)$.

The two signals *START* and *STOP* representing the interval are sent to the two delay lines. In each stage, an arbiter decides which signal comes first, and sets "1" at the corresponding output (cf. Subsection 1.2.4). Since *STAR* arrives before *STOP*, and the first arbiters of the chain detects that *START* arrives before *STOP*. However, as τ_1 is larger than τ_2, after each stage delay between two signals is reduced by $\Delta\tau_{TDC} = \tau_1 - \tau_2$. After a certain number of stages M, the delayed *STOP* signal will arrive before the delayed *START* signal. Hence for stage i, $0 \leq i < M$, $\overline{B_{first}} = 0$, while for $i \geq M$, $\overline{B_{first}} = 1$. This data provides an idea about the value of the initial time interval under measurement.

The reading of the TDC measurement result is done synchronously with the falling edge of the signal *SAMPLE*. It is obtained by delaying the *START* pulse by 3 delays of τ_1, which allows enough time for establishment of the arbiter outputs and metastability resolving. We can observe that at the falling edge of *CK*, $\overline{Q}$ takes the complement value of the input D ($\overline{B_{first}}$ in TDC). By consequence, the registers in the TDC contain a digital word in thermometer code representing the time interval to be measured. For stage i, $0 \leq i < M$, $S_i = 1$, while for

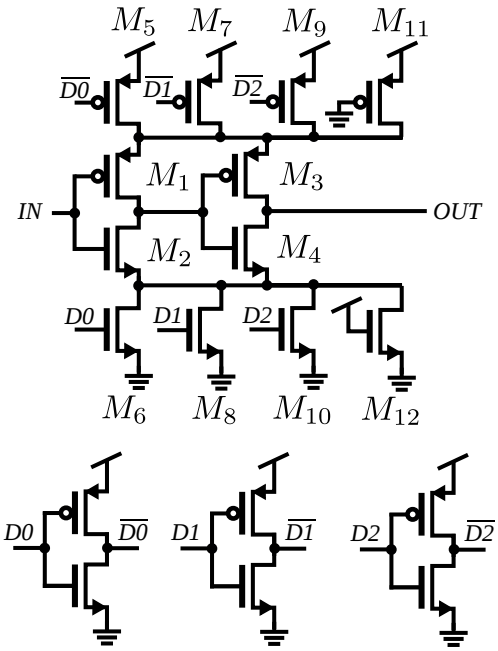
$i \geq M$, $S_i = 0$. The obtained thermometer code can be one of the 7 values from 000000 to 111111. It is then combined with the *SIGN* and converted to a 2's complement binary code as output of the PFD.

We note that in the three last stages (arbiters I25-I35) only delays τ_1 are used. Other blocks (delays τ_2 and arbiters) are dummy, and provide equivalent load for the last stage as for the previous stages. This is important for the linearity of TDC within its dynamic range.

Implementation of blocks of the TDC

In the implemented TDC, the key cells which are D-register, arbiter and delay are implemented as custom cells. The register and the arbiter are the same as implemented in the first prototype, and are described in [74]. Here we provide information about the implementation of the delays.

The delay elements τ_1 and τ_2 are the essential blocks of the architecture, since they decide the resolution of TDC ($\Delta\tau_{TDC}$). The two delay blocks have the same schematic topology (cf. Fig. 3.4) but employ transistor with different sizes.



Cell	τ_1		τ_2	
Parameter	W μm	L μm	W μm	L μm
$M_{1,3}$	1.4	0.15	1.4	0.12
$M_{2,4}$	1.0	0.15	1.0	0.12
$M_{5,7,9}$	0.28	0.12	0.28	0.12
$M_{6,8,10}$	0.2	0.12	0.2	0.12
M_{11}	1.12	0.12	1.12	0.12
M_{12}	0.8	0.12	0.8	0.12

Figure 3.4: **Schematic diagram of the delay cells in Vernier TDC**

As shown in Fig. 3.4, the delay cell is based on a 2-stage inverter line ($M_{1,2}$ and $M_{3,4}$) with additional pull-down NMOS ($M_{6,8,10}$) and pull-up PMOS ($M_{5,7,9}$) whose gate ports are controlled by binary signals $D\langle 2:0\rangle$ and $\overline{D\langle 2:0\rangle}$ respectively. For such a circuit with given sizes of transistors, the load capacitance is fixed. However, by programming $D\langle 2:0\rangle$, we can choose the number of transistors turned on, thus we can tune the charging and discharging current of the inverters. It should be noted that since $M_{6,8,10}$ have the identical size (the same case for $M_{5,7,9}$), the delay of the cell is decided by the number of tuning transistors turned

Table 3.1: **Values of τ_1 and τ_2 with different N_{on}**

N_{on}	τ_1 (ps)	τ_2 (ps)
0	81.5	68.7
1	77	64
2	73.6	60.8
3	71	58

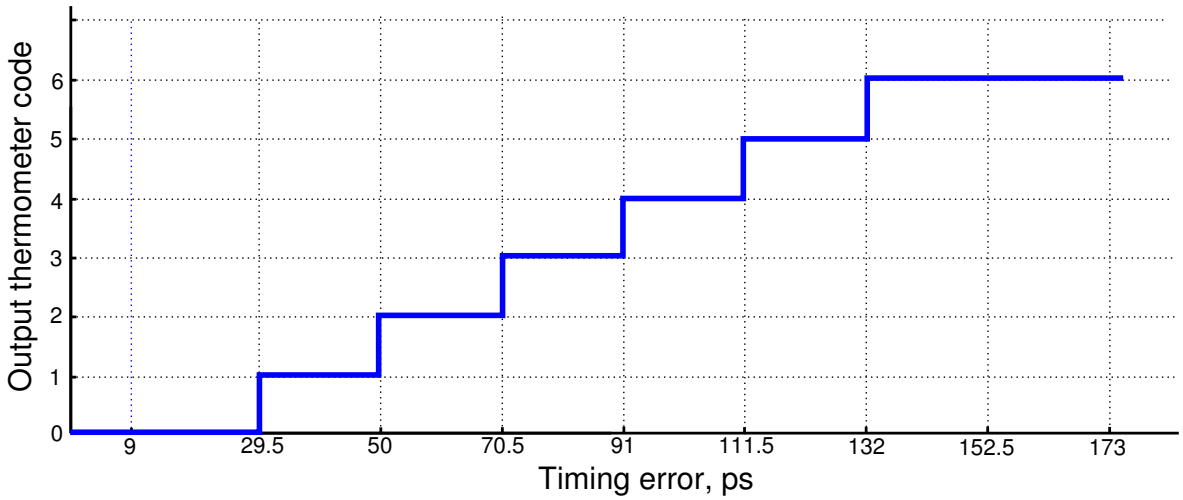


Figure 3.5: **Simulated transfer function of the designed flash time-to-digital converter**: measuring resolution 20.5 ps, range 0-152.5 ps

on (defined as N_{on}). By consequence, one delay cell can have four different delay values (N_{on} from 0 to 3). For instance, if $M_{6,8,10}$ and $M_{5,7,9}$ are all turned off ($N_{on} = 0$), it has the smallest current and largest delay. If all the transistors are turned on ($N_{on} = 3$), it has the smallest delay. Tab. 3.1 displays the values of τ_1 and τ_2 with different N_{on} obtained by post-layout simulation. From the table we can see that if we want $\Delta\tau_{TDC}$ to be about 20 ps, we can set the N_{on} of delay cell τ_1 to be 0 and that of τ_2 as 2 in schematic level design. In physical design, we just need to connect $\overline{D\langle 2:0\rangle}$ to supply or ground rail, according to the wanted delay value.

With all the cells presented above (arbiter, register and delay), we designed the TDC circuit in transistor level. Precise transistor-level simulations provided a 20.5 ps quantification step of the TDC, with $D = \{000\}$ for τ_1 and $D = \{101\}$ for τ_2. Fig. 3.5 demonstrates the code/error characteristics obtained from the post-layout simulations of the designed TDC. From Fig. 3.5, three regions of TDC operation can be noted:

1. The time interval is less than the first delay element plus the delay of one logic gate (Eq. (3.1)): the output register generates a word with all zeros;
2. If the time interval to be measured is greater than the delay of the first buffer but less than the total delay of the line: first bits of the register are at 1, and the rest are at zero (Fig. 3.2(b));

3. The input time interval is greater than the overall line delay: all bits of the register are at '1'. The TDC is unable to discriminate the time interval values, and the TDC output is in saturation providing the maximal code.

In that way, the output of the parallel register $S0-S5$ provides a thermometer code from 000000 to 111111 representing the measured and quantified value of the time interval in the range $(0, 7{\cdot}\Delta\tau_{TDC})$.

3.1.3 Implementation of PFD

The block diagram of the implemented PFD is presented in Fig. 3.1. The output of PFD is obtained by combining the TDC and BB detector outputs through an arithmetic block. The role of the arithmetic block is a generation of the binary coded 2's-complement signed word corresponding to the measured phase error. Its absolute value is composed of the thermometer coded output word of the TDC and the sign provided by the BB detector, accordingly to the following formula:

$$PFD_{out} = (TDC_{out} + 1)SIGN. \tag{3.2}$$

The arithmetic block is implemented by standard design flow starting from a behavioral description in VHDL, by using Design Compiler of Synopsis and Encounter of Cadence.

The TDC outputs values from 0 (the error is less than $\Delta\tau_{TDC}$) to 6 (the error is greater than $6\Delta\tau_{TDC}$). Hence, the PFD_{out} is in the range $[-7,-1]\cup[1,7]$. The minimum output of PFD is ± 1 to maximize the information carried by the output code (cf. Subsection 1.2.4). The PFD has two output of 4 bit binary signed words: the error $e_{ri}[n]$ and its complement value $\bar{e}_{ri}[n]$.

The sub-blocks for the PFD are assembled and Fig. 3.6 presents the layout of designed PFD. The I/O ports are located on the border of the cell and placed at Metal3 layer. The supply network is organized so to provide the compatibility with the global floorplan of the clock network chip.The designed PFD has a total area of $53.8\times31.4\ \mu m^2$, which is smaller compared with the first version($53.0\times40.04\ \mu m^2$) PFD designed with the same technology.

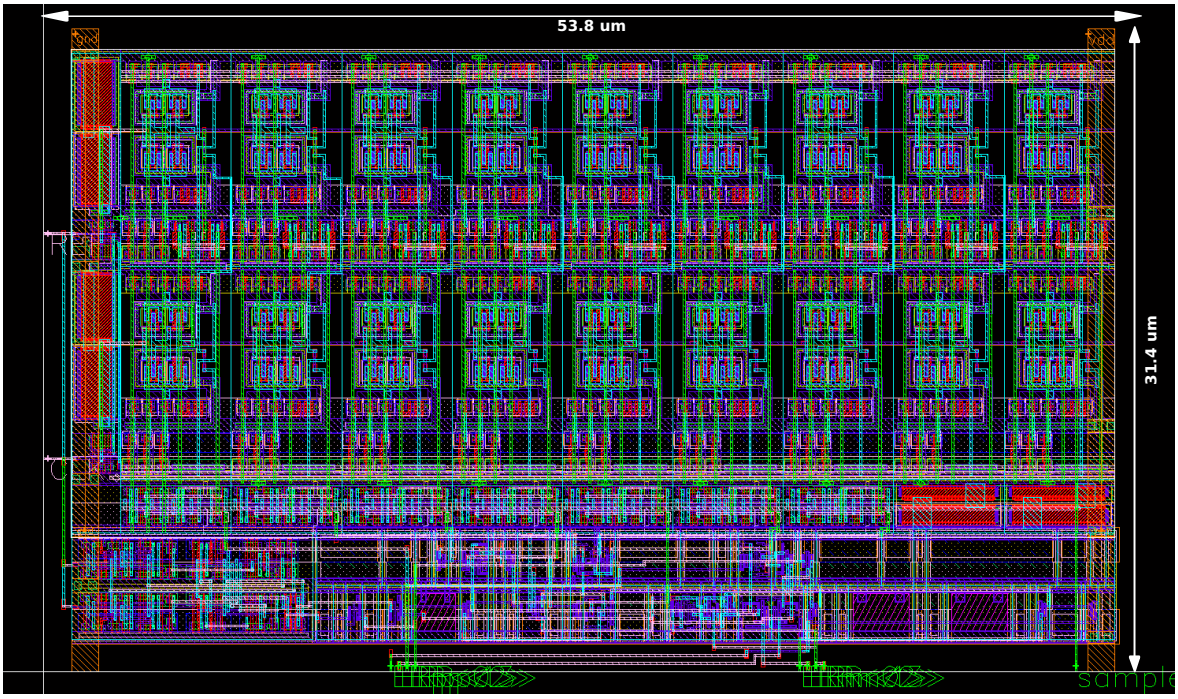


Figure 3.6: **Layout of the proposed PFD**

3.2 Digital filter in the ADPLL network

The architecture of the filter is similar with that used in the first prototype. The modifications aimed essentially a reduction of the power consumption (cf. Section 1.3), an acceleration of the speed of error correction. The implemented architecture benefits from the reduced word width of the PFD output, and from reduced range of the coefficients for the programmable Proportional-Integral filter. This allows also a reduction of the delay in the proportional branch of the filter, which is critical for the error correction in the steady-state mode.

As in the first prototype, the filter includes an error combining blocks at its input. It processes phase errors between the local clock and clocks issued from neighboring PFDs.

3.2.1 Architecture of digital filter

The schematic of designed digital filter is displayed in Fig. 3.7. It has four entries allowing receiving at most four 4-bit signed words as inputs and generates an 8-bit unsigned word for the DCO control (the output of the adder ADD5). This signal is encoded to A, B and C, which are applied to the inputs of the DCO (cf. Section 3.3).

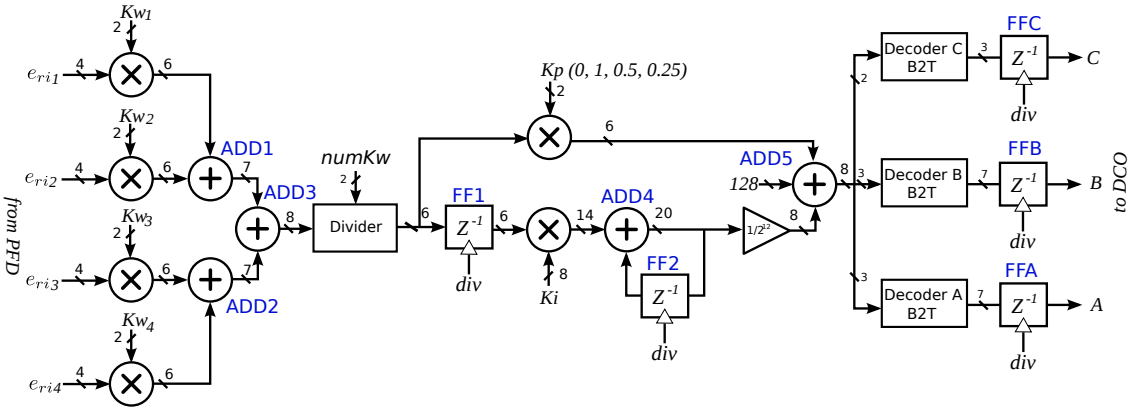


Figure 3.7: **Loop filter for error signal processing**: four input gain controllers followed by the four-input adder, PI filter and three B2T decoders.

The error combining block (from inputs to ADD3) has the same structure as the one presented in cf. Subsection 1.2.5. The weighting coefficients of the gain blocks $Kw_1 - Kw_4$ are programmable. Each gain can take independently a value in the set {0,1,2,4}. The adders (ADD1-3) operate with four 4-bit operands and produce a 8-bit sum. This adder is based on the carry look-ahead (CLA) architecture. Its delay is less than 510 ps in the worst case. The filter processes the average value of the sum of errors (e_{ri1}, e_{ri2}, e_{ri3} and e_{ri4})(cf. [29]), so the output of the adder ADD3 is followed by the block "Divider", which calculates the mean value of quantified phase errors. In fact, at functional level, this block is not necessary, because a division by N can be realized implicitly by programming both the proportional and integral coefficients N times smaller. However, the introduction of an explicit division has two advantages:

1. The calculation after division is simplified: less bits are needed for operators (adders, multipliers and integrators), thus the arithmetic operation is faster and the filter has smaller area;
2. The proportional and integral coefficients for all ADPLLs in the network are unified: without this block "Divider", K_p and K_i should be programmed diversely in function of the number of E_{ri} entries, which depends on the position of the ADPLL and the configuration of network. This needs a pre-calculation of coefficients for each filter and each configuration, which is unnecessary extra effort for users of the proposed methodology.

However, it should be noted that a 2-bit number (*numKw*) should be programmed together with other coefficients. It indicates the number of entries, in other words, the divisor. The integer value of *numKw* could be 1, 2 or 4. All are powers of 2 in order to simplify the arithmetic operation (in case of dividing by 3, the divisor 4 is used).

After the division, the value is sent to the integral and proportional paths.

The calculations inside the filter are achieved in *fixed point* arithmetic. For the proportional path, the 2-bit parameter K_p allows choosing the proportional coefficient α from a set of 4 values {0,1,0.5,0.25}.

$$\begin{aligned} \alpha &= 0, \quad \text{when } K_p = 3; \\ &= \frac{1}{2^{K_p}}, \quad \text{when others.} \end{aligned} \tag{3.3}$$

Each value of α in the subset 1,0.5,0.25 is a power of 2, thus the multiplication is simple enough allowing the whole proportional path processing to be finished within one cycle. Hence no delay is introduced in the proportional path so that it can react quickly to the measurement result of PFD in each clock cycle. The ADD5 receives the rounded-to-integer result of the proportional branch, on 8 bits.

The integral coefficient β is defined as a ratio:

$$\beta = \frac{K_i}{2^{12}}, \tag{3.4}$$

where K_i is in the range $(0, 2^8 - 1)$. Since the accumulator in this path (ADD4 and FF2 in Fig. 3.7) takes time for calculation, the divided value is buffered with a register FF1, providing time of two clock cycles for the arithmetic operations. This is necessary for sufficient timing margin in the integral path.

The architecture of the integrator path is given in Fig. 3.7. The main point of the integrator design is an appropriate choice of the size of the integrator register FF2. Here we summarize the integrator path design procedure.

A particularity of an integrator is an unlimited range of the output when this block is considered alone. Hence, a design of an integrator requires a specification on the range of the output value. This specification is defined by the system-level considerations and is practically ensured by a feedback. For the case of the PLL, the integrator defines the rough value of the frequency of the DCO; hence, the normal operation range is that of the DCO input (0, 255).

The integrator outputs unsigned integer values on 8 bits, receives input integer values on 7 bits. According to Eq. (3.4), the integral coefficient has a precision of up to 12 binary digits after the binary point. To achieve an integration without loss of information, the accumulator must be able to store a number on 12+8=20 bits. The 12 LSB are then ignored (this is represented by the block dividing by 2^{-12} in Fig. 3.7), and only the integer part on 8 bits is applied on the final adder ADD5.

After the global system reset, the integrator outputs 0, and the initial frequency of the ADPLL network node is the minimal one. If the target frequency is close to the maximal output frequency, the accumulator should increase its value from 0, and the frequency acquisition time can be long. To reduce the maximal frequency acquisition time, the integrator is preset with an initial value of 128 (the middle of the scale). This is done by adding a constant offset of 128 to the output value of the filter.

The parameters (Kw, Kp, Ki, $numKw$) are programmed to each loop control block in the network by using the SPI technique presented in Subsection 3.3.4. 20 bits are necessary for each node, and they are organized as shown in Fig. 3.8.

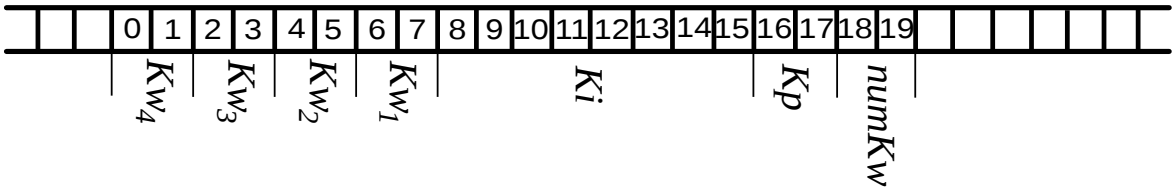


Figure 3.8: **Programming sequence of parameters**

3.2.2 Implementation of the filter

The digital filter has been implemented using standard RTL-to-GDS design flow with help of EDA tools. It was at first described in VHDL language (cf. Appendix A for details). The gate-level netlist and the layout was synthesized using standard cell library of STMicroelectronics.

At the layout level, the standard cells composing the filter were placed and routed together with the DCO and PFD macroblocks (blackbox view at Encounter), so to optimize the chip area. The details of implementation of this physical block called NODE is presented in Fig. 6.6, and is discussed in Section 6.3. The digital filter occupies a area of about 7.6×10^3 μm^2 in the NODE.

3.3 Digitally controlled oscillator (DCO)

Basically, the DCO presented in this section has the same architecture with the first DCO designed for the first test chip (cf. Subsection 1.2.6) but with reduced tuning steps thus less tri-state transistors in parallel for each stage.

3.3.1 DCO Architecture

Detailed architecture of the DCO is given in Fig. 3.9. It is a ring oscillator controlled by three thermometer coded signals A, B and C converted from a 8-bit binary code generated by the loop filter. The ring oscillator is constituted with 7 stages. Each stage contains one main inverter ($MI_0 - MI_6$), which is always active. The tuning inverters are connected in parallel with each stage and distributed over all 7 stages of oscillator. Each tuning cell is a controllable inverter associated with a local control logic decoding the row and column actuation code.

These tuning elements are also organized in three arrays implementing a coarse and a fine frequency tuning. If we reserve 4 fine tuning steps for each coarse tuning step as in Subsection 1.2.6, and we want totally 256 tuning steps, 64 coarse tuning steps are needed. Since the ring oscillator has 7 stages plus 1 virtual stage, for the first array we need only 8 rows instead of 32 rows in Fig. 1.16. Consequently, only one row of 7 virtual extension cells is enough for the 8 rows in the first array (the last row does not need an extension because 256 steps means 255 changes from 0 to 255).

In summary, the three arrays are organized as follows. The first array consists of 56 (7 stages $\times$ 8 rows) coarse-tuning tri-state inverters (CTI). The second array consists of a bank of 7 additional coarse-tuning inverters (CTIA) (7 stages $\times$ 1 row). They are implemented as the virtual extension of ring stages (cf. Subsection 3.3.2 for details). The CTI and CTIA cells (totally 63 identical elements) provide 2^6 equal frequency steps. The third array is implemented with three identical fine-tuning tri-state inverters $FTI0 - FTI2$, which are connected with stage 1, 3, 5. These cells provide 4 fine frequency tuning steps by each coarse tuning step. Together with coarse tuning inverters, they provide 256 frequency steps.

The size of transistors in FTI cells is defined by CTI/CTIA transistor dimensions divided by the number of the fine tuning steps within each coarse tuning step (size ratio 1:4). The number of fine tuning steps must be a power of two for control simplicity, and set to 4 as a compromise between the precision (monotonicity may degrade when the number of fine tuning steps increases) and the obtained gain in the DCO overall number of frequency steps. In the 8 bit input control binary code, the coarse and fine frequency tuning correspond to the 6 MSB and to the 2 LSB frequency variation respectively.

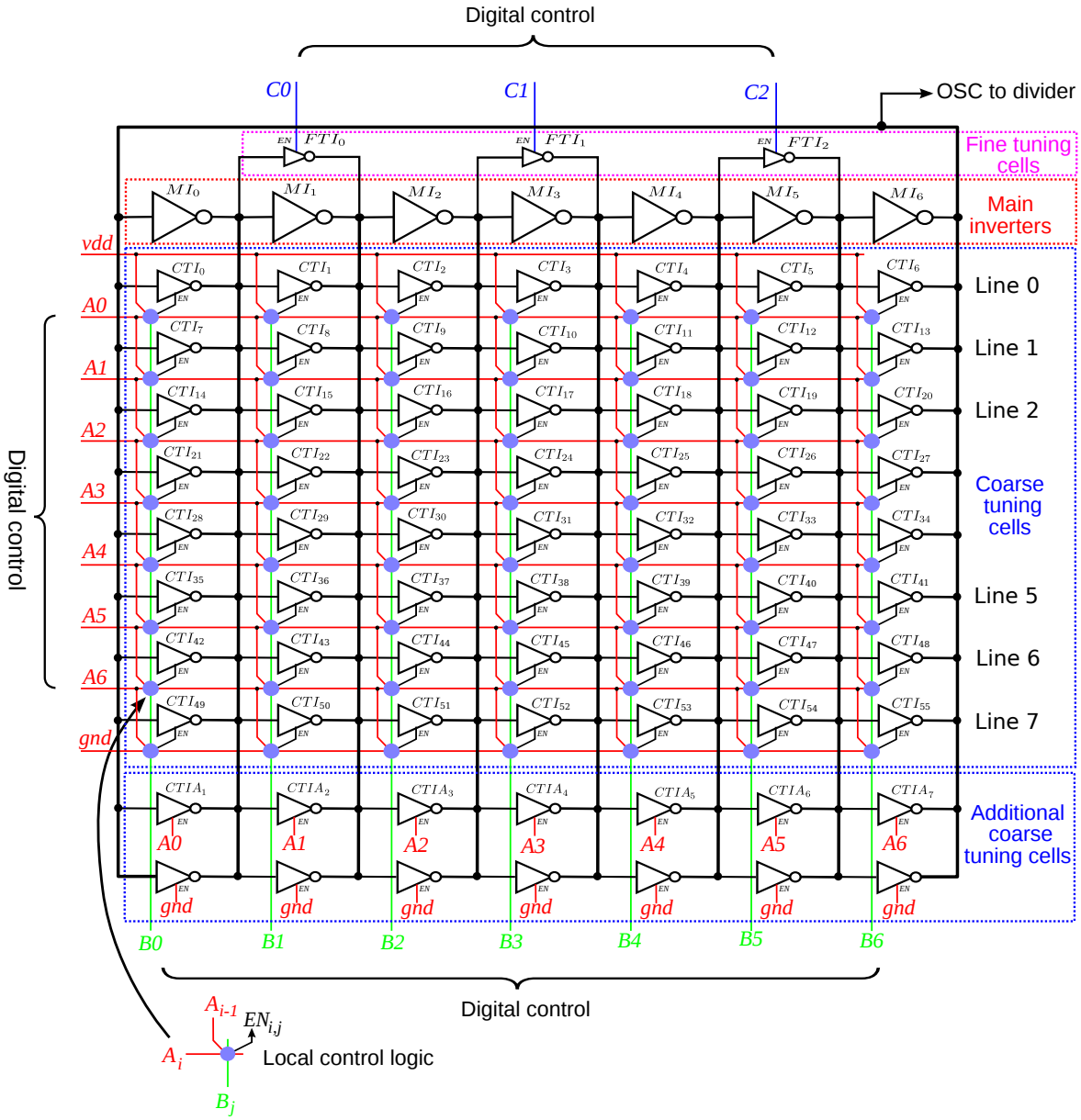


Figure 3.9: **Core of the proposed oscillator:** *MI* - main inverters, *CTI* - coarse-tuning inverters, *FTI* - fine tuning inverters

3.3.2 Control algorithm

A binary-to-thermometer (B2T) decoder is needed to generate three thermometer coded signals A, B and C from the 8-bit binary input code W. A, B and C are in charge of row, column and fine tuning cell controls respectively. From the explanation in Subsection 3.3.1 we can get that the code A, which controls the row selection (8 rows), should be decoded from the 3 MSBs of W ($W\langle 5:7\rangle$). The code B, which decides the selected stages (7+1 stages), should be decoded from the middle 3 bits of W ($W\langle 2:4\rangle$). The code C, which controls the 4 fine tuning steps, are from the 2 LSBs of W ($W\langle 0:1\rangle$). Therefore, the control signals A, B and C

are defined by the following equations:

$$\begin{cases} A = W\%32 \\ B = (W \text{ modulo } 32)\%4 \\ C = W \text{ modulo } 4 \end{cases} \tag{3.5}$$

where % means the integer division. The operation (K modulo 2^p) on a binary coded signal K is implemented by selection of p LSB from K, the operation $K\%(2^p)$ is a right shift of K by p position. We note that Eq. (3.5) is almost the same as Eq. (1.6) although the two DCOs do not have the same number of tuning steps and control bits. The reason is that they have the same number of fine tuning steps in each coarse tuning steps and the number of stages does not change neither, thus B and C keep the same. As for A, its width is smaller along with the reduction of rows. Moreover, since A_0 in the old version DCO is always 1 (cf. Eq. (1.6) in Subsection 1.2.6), we connect the row control of cells in the first row with *vdd* (cf. Fig. 3.9), and we don't need to add 1 to the formula of A.

The logical equation for individual enable signal of the coarse tuning inverters is given by:

$$EN_{cti\,i,j} = \begin{cases} A_i \cup B_j, & i = 0; \\ A_i \cup (B_j \cap A_{i-1}), & 1 \leq i \leq 6; \\ B_j \cap A_{i-1}, & i = 7; \end{cases} \tag{3.6}$$

where $i \in \{0,1,..7\}$ and $j \in \{0,..6\}$ are the number of the CTI row and column respectively.

For the additional coarse tuning cells, their enables are given by:

$$EN_{ctia\,i} = A_{i-1},\ i = 1...7. \tag{3.7}$$

Here i is the index of additional cell equal to the index of the CTI row completed by this cell. Note that the last CTI row with index 7 doesn't have the extension. That is because the coarse tuning cells provide 2^3 frequency values, and for that $2^3 - 1$ coarse tuning cells is required.

The corresponding individual *enable* signals of the code C is given by

$$EN_{fti\,i} = C_i \tag{3.8}$$

where $i = 0..2$, i is the index of the FTI cell.

These operations are implemented directly in the tuning cells (Fig. 3.12, Fig. 3.13 and Fig. 3.14).

Fig. 3.10 demonstrates the order in which the inverters are activated when the input code changes from 0 to 20_{16}. When the input code is 00_{16}, only MIs are active and the output frequency is the lowest. When the input code equal to 01_{16}, FTI0 becomes active, so increasing the charge/discharge current of the stage 1, resulting in the decrease of this stage's delay and thus total delay of the loop. After the activation of all FTIs (the code 03_{16}), the next code

04_{16} activates the first CTI #000, deactivating three FTIs. The codes $05_{16} - 07_{16}$ activate again three FTI. The cycle repeats for the FTIs until the activation of the CTI #001 and so on. When all FTIs and the first row CTIs are activated, the code is equal to $1F_{16}$. The next code 20_{16} keeps the 1st row CTIs on and activates the 1st CTIA in the additional bank. Such a cycle repeats 8 times when the input code goes from 0 to FF_{16}. At FF_{16}, all tuning cells in arrays are active, and total delay of the oscillator loop is the lowest and the corresponding output frequency is highest.

Code / Cell	000_{16}	001_{16}	002_{16}	003_{16}	004_{16}	005_{16}	006_{16}	007_{16}	...	$01E_{16}$	$01F_{16}$	020_{16}
CTI#000	○	○	○	○	●	●	●	●	...	●	●	●
CTI#001	○	○	○	○	○	○	○	○	...	●	●	●
...#...	...	...	...	...	...	...	...	...	...	...	...	...
...#...	...	...	...	...	...	...	...	...	...	...	...	...
CTI#006	○	○	○	○	○	○	○	○	...	●	●	●
CTIA#A1	○	○	○	○	○	○	○	○	...	○	○	●
FTI0	○	●	●	●	○	●	●	●	...	●	●	○
FTI1	○	○	●	●	○	○	●	●	...	●	●	○
FTI2	○	○	○	●	○	○	○	●	...	○	●	○

Figure 3.10: **Ring oscillator cell control table**

3.3.3 Implementation

Sizing of tuning cells

Since the number of tuning cells is reduced, the load capacity is not the same as before. Moreover, the specification has been modified. Consequently, the transistors should be re-sized. The sizing procedure follows the same methodology used for the DCO for the first test chip. It is repeated as follows.

The ratio between the frequency step ΔF (ΔF_{coarse} and ΔF_{fine}) and the initial frequency F_0 is given by the ratio between the currents generated by the main cells (I_m) and the tuning cells (I_{cti} and I_{fti}) respectively. It can be roughly considered that the current generated by an inverter is proportional to the W/L ratio of the N transistor. Hence, we come to the following relation:

$$\left(\frac{W}{L}\right)_{main} : \left(\frac{W}{L}\right)_{cti} : \left(\frac{W}{L}\right)_{fti} = F_0 : \Delta F_{coarse} : \Delta F_{fine} \tag{3.9}$$

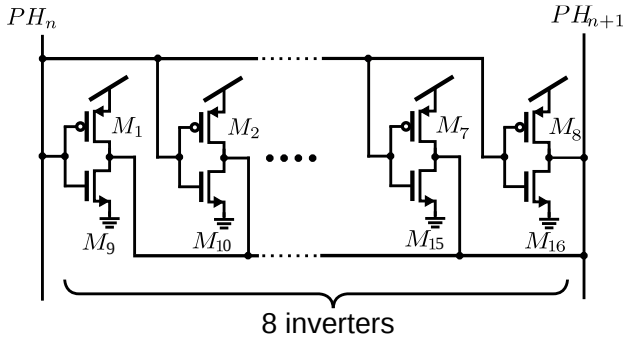
The parameters F_0 and ΔF_{fine} are given by the specifications (cf. Section 2.4). From the architecture of the designed DCO, $\Delta F_{coarse}/\Delta F_{fine} = 4$.

This double relation includes three unknowns (the W/L ratios of the three cell types). The third missing equation is given by the requirement about the absolute value of F_0 or F_{max}. The W/L should be such that when all tuning cells are activated, the ring oscillates at the

maximal desired frequency corresponding to the maximal input code value. A simulation-based adjusting/tuning has been performed to obtain the desired absolute value of F_{max}.

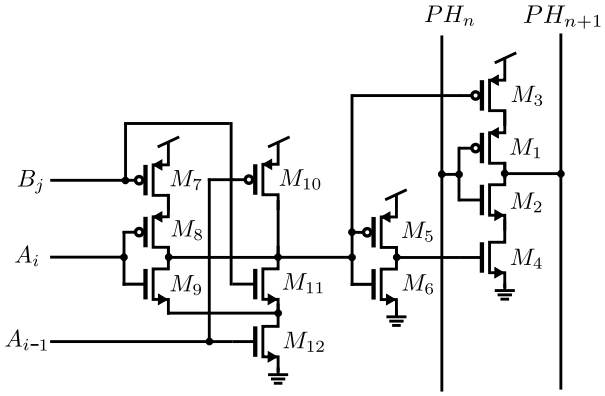
After fixing the W/L value, the values of L and W must be chosen. An immediate approach consists in minimizing the size of the smallest inverter of the architecture. However, in order to reduce the impact of the fabrication errors on the DCO code-frequency characteristic, the use of tuning cells with minimal sizes allowed by the technology should be avoided. We chose the L of the tuning cells equal to twice the minimal size allowed in the used 65 nm CMOS technology.

The dimensions of the PMOS and NMOS devices in inverters obtained from mentioned considerations are given in tables in Fig. 3.11, Fig. 3.12, Fig. 3.13 and Fig. 3.14.



Cell	Main	
Parameter	W μm	L μm
$M_1 - M_8$	1.38	0.07
$M_9 - M_{16}$	0.98	0.07

Figure 3.11: **Schematic of the main inverter of oscillator**: 8 inverters connected in parallel.

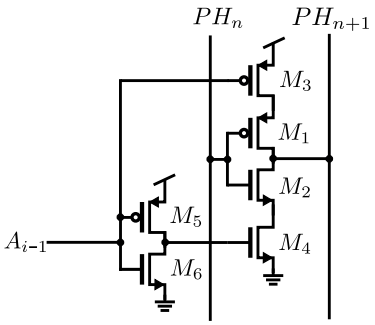


Cell	CTI	
Parameter	W μm	L μm
M_1	1.26	0.2
M_2	0.9	0.2
M_3	1.26	0.06
M_4	0.9	0.06
$M_{5,7,8,10}$	0.28	0.06
$M_{6,9,11,12}$	0.2	0.06

Figure 3.12: **Schematic diagram of the coarse tuning cells**

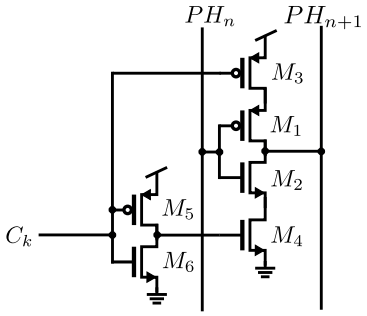
Dividers and buffers

The divider block generates three clock signals: *CLK* is a buffered OSC clock signal (the output of the ADPLL network clock generator), *DIV* is a divided-by-4 clock for the feedback of ADPLL and for the phase coupling with neighbors. The signal *DIV*8 is a divided-by-8 clock routed on the chip pads for the off-chip test purposes. Only this signal can be properly measured outside of the chip, because the communication frequency of the I/O pads available in the design kit is limited to 270 MHz.



Cell	CTIA	
Parameter	W μm	L μm
M_1	1.26	0.2
M_2	0.9	0.2
M_3	1.26	0.06
M_4	0.9	0.06
M_5	0.28	0.06
M_6	0.2	0.06

Figure 3.13: **Schematic diagram of the additional coarse tuning cells**



Cell	FTI	
Parameter	W μm	L μm
M_1	0.28	0.2
M_2	0.2	0.2
M_3	0.28	0.06
M_4	0.2	0.06
M_5	0.28	0.06
M_6	0.2	0.06

Figure 3.14: **Schematic diagram of the fine tuning cells**

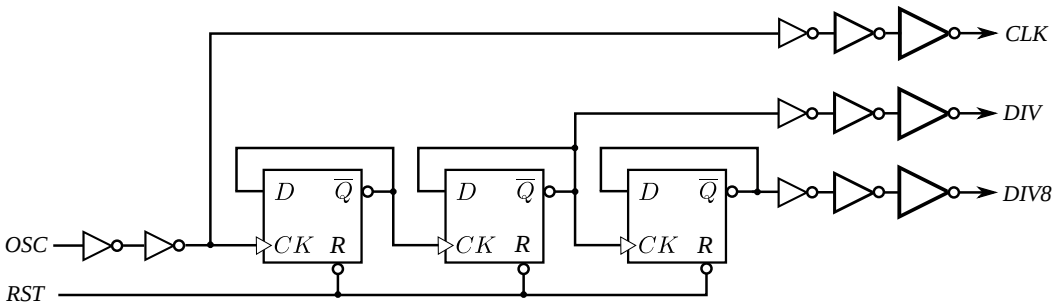


Figure 3.15: **Schematic diagram of the feedback frequency divider**

Fig. 3.15 shows the schematic of the divider. It consists of input buffering inverter and three D flip-flops with inverted output. Besides the original clock signal generated by the oscillator, it generates two additional divided clock signals: the divided-by-4 *DIV* signal and the divided-by-8 *DIV*8 signal.

DCO Floorplan

The general floorplan of the oscillator includes 8 groups of blocks. They are towered in stack in the order depicted in Fig. 3.16. These groups are:

- frequency dividers;
- main inverters;
- arrays of CTI cells;

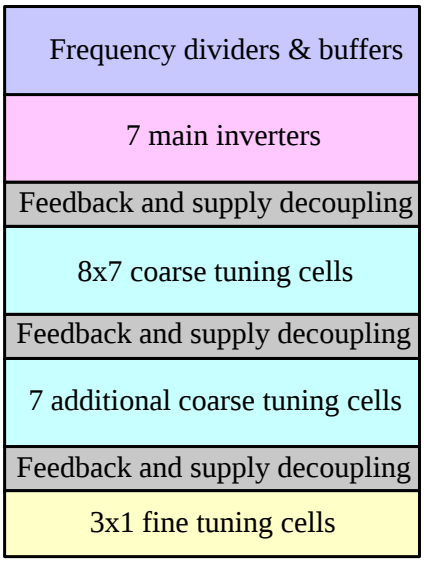


Figure 3.16: **Floorplan of the designed oscillator**

- arrays of CTIA cells;
- array of FTI cells;
- feedback line and supply decoupling cells.

The dividers are located far from the sensitive tuning cells. They are aligned with the main inverter row. A group of CTI cells (composed as 8×7), a row of CTIA cells, decoupling capacitors and global loop feedback lines are placed under the main inverter row. The array of FTI cells is placed at the bottom of this stack.

DCO layout

The proposed DCO has a regular structure. The main challenge of the DCO layout implementation is placement of the CTI array and routing of the control signals. The implementation strategy must optimize the performances of the DCO and the circuit area. The following techniques have been used for the DCO layout design:

- Cell based design. The DCO layout is an assembly of identical elementary blocks implementing CTI, CTIA, FTI, MI and interconnections;
- Connection by abutment. The cell layout is designed so that they are connected by geometrical abutment;
- A flow oriented location of the cells. The placement choice of cells depends on signal propagation direction;
- Cell oriented supply design. As in conventional geometries of digital circuit layouts, the supply is routed through multi-finger structures (Fig. 3.17). That allows each cell to share its ground and supply pads with the neighbors.

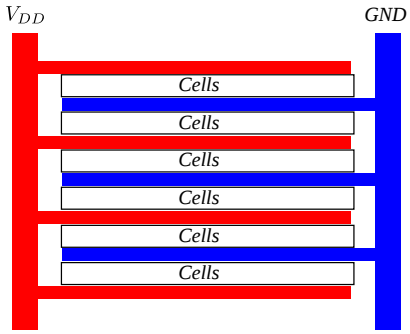


Figure 3.17: **Interdigital multi-finger power routing geometry**

Fig. 3.18 shows the layout of designed DCO. It has a dimension of 52 $\mu m \times 53.8$ μm. The core is about 45 $\mu m \times 45$ μm surrounded by guard rings, which decrease the effect of substrate coupling with surrounding digital circuits.

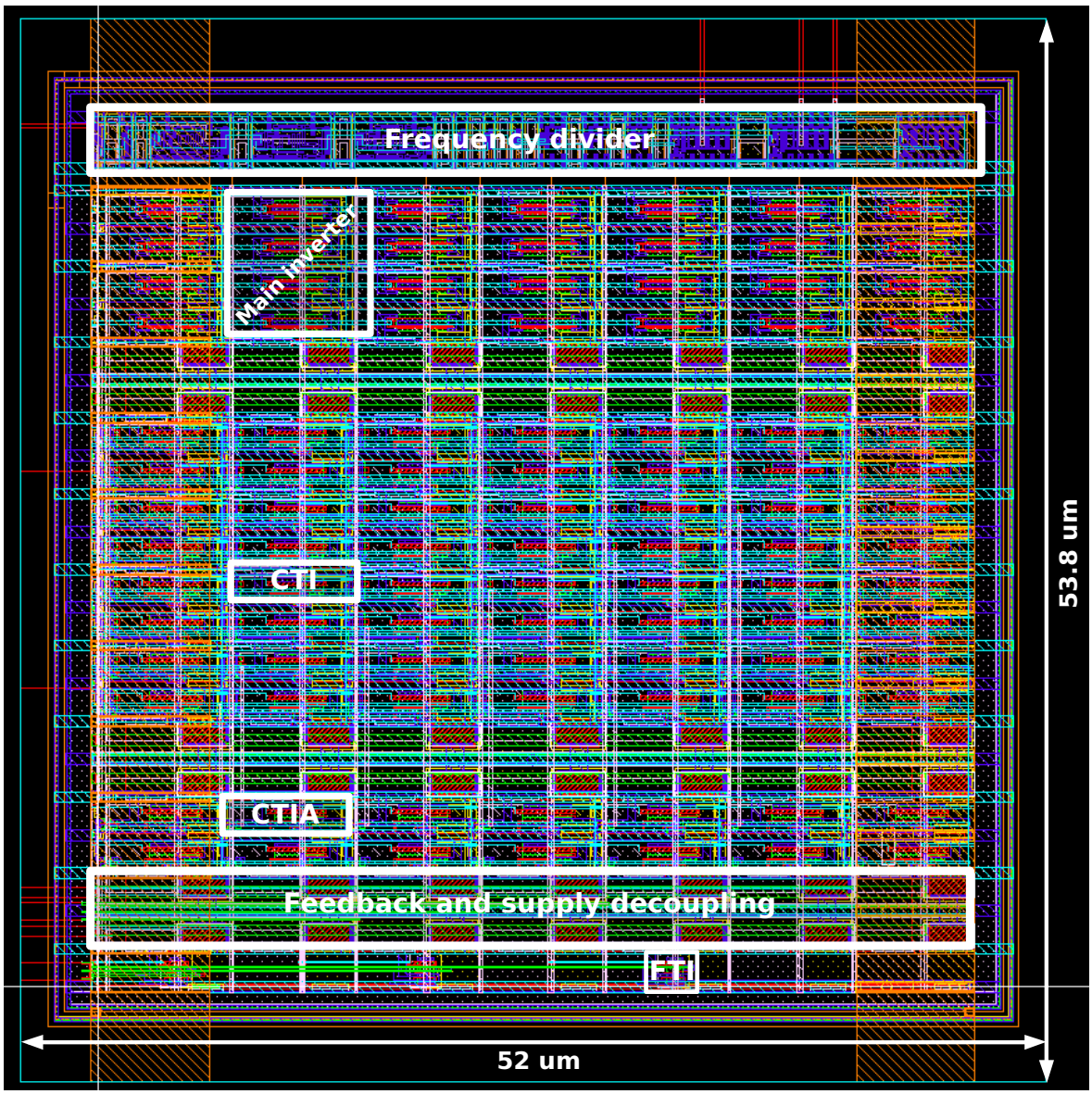


Figure 3.18: **Layout of the designed oscillator**

3.3.4 Serial programming interface (SPI)

A complex programming interface is needed for three reasons.

- For the testing purposes, parameters of the filter are programmable. The programming interface should allow a definition of the network parameters before each start-up of network. These parameters are K_p and K_i in the loop filter and $Kw_1 - Kw_4$ in the gain controller.
- The normal operation of network is start by a reset signal: the reset should be generated after the network parameters are programmed.
- The dynamic mode selection technique chosen for our system (cf. Subsection 1.2.9) requires an interface allowing a *dynamic* (on the fly) reconfiguration of the network. The reconfiguration process must not perturb the operation of the network.

The programming interface must be flexible and extensible, so to be easily adaptable to the topology of the network (number of nodes, etc.). Since the number of the bits to be programmed is large (20 bits per filter), serial interface should be used for the programming sequence transmission.

The designed serial programming interface (SPI) fulfils the above mentioned requirements. It is composed of two registers and one flip-flop (Fig. 3.19). The register XI0 is a serial-to-parallel converting register receiving the input series data on SDA_i and generating a parallel word at its outputs *parallel data out*. The actual values of the outputs of XI0 represent the values to be programmed. However, during the input sequence reading, the *parallel data out* outputs have transient meaningless values. For this reason, the outputs of this register are not applied directly to the node block, but to the storage parallel register XI2 playing the role of a buffer. This loading is ordered by a global UPD signal.

This programming interface complies with the above mentioned specifications:

- The start-up of the ADPLL network operation can be perfectly controlled: the global reset signal controlling the initial state of all registers of the network is not applied to the programming interface register. In this way, once the network is programmed, the global reset can be applied and the network starts from a well-established state.
- If the network parameters need to be modified during a regular network operation (for example, the weight coefficients Kw_i), the new values are firstly loaded in the register XI0. Then the UPD signal is sent, and after a delay, the network operates with updated parameters. This delay is short comparing with one clock cycle of the DIV signal. It is equal to the propagation time of the UPD signal and the settling delay of the parallel register (few tens of picoseconds).

- The programming interface is easily extensible by cascading (Fig. 3.20). The programming interfaces of two blocks are joined by connecting the last output bit of the register XI0 of one block to the SDA_i of the another block. The two blocks share the same UPD and SCK signals. In this way, for any length of the programming sequence, the interface requires only three external pads: SCK, SDA and UDP (clock, data and load signals).

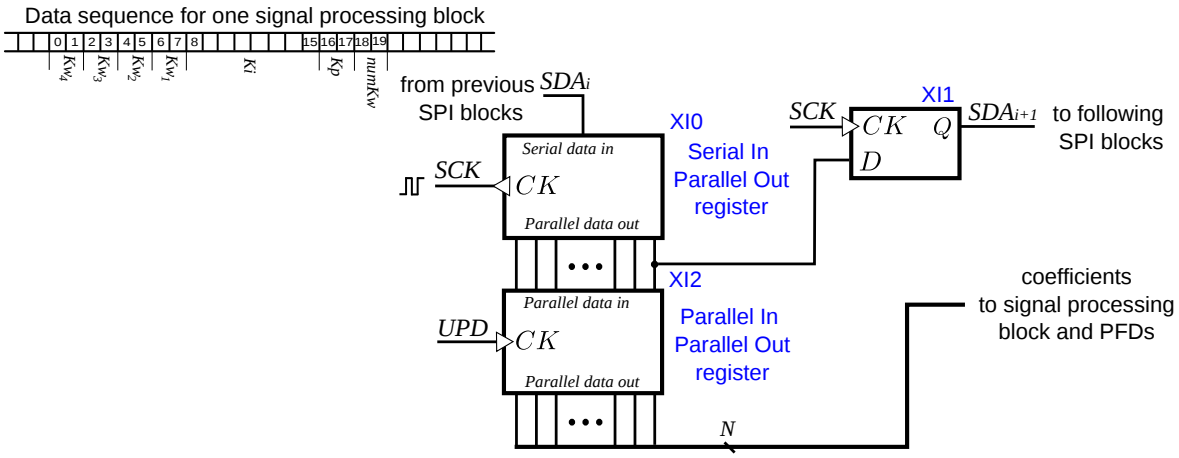


Figure 3.19: **Schematic of the programming interface**

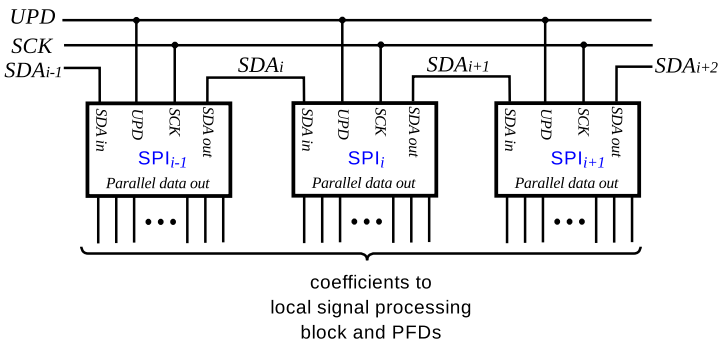


Figure 3.20: **Cascading the programming interfaces of several blocks**

3.3.5 Simulation results

This subsection presents simulation results of the DCO block. The Tab. 3.2 presents the performance and parameters summary.

Tuning curve in typical conditions

Fig. 3.21 shows the measured transfer function of DCO in typical condition (1.1 V power supply, 27°C). We can observe a good linearity of designed block. Fig. 3.22 demonstrates the monotonic and linear feature of the block in another way: the variation of frequency tuning step corresponding to each FCW. We can observe that there is no negative frequency step, which guarantees the monotonicity of DCO.

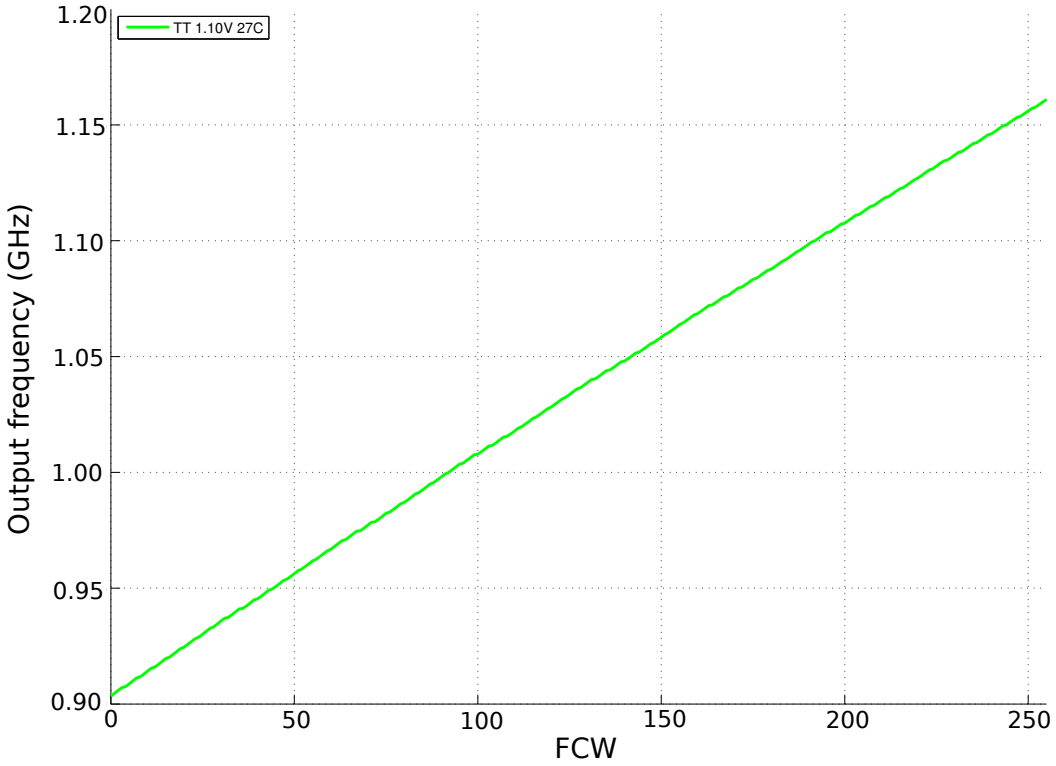


Figure 3.21: **Simulated output frequency versus frequency control word (FCW)**: typical condition

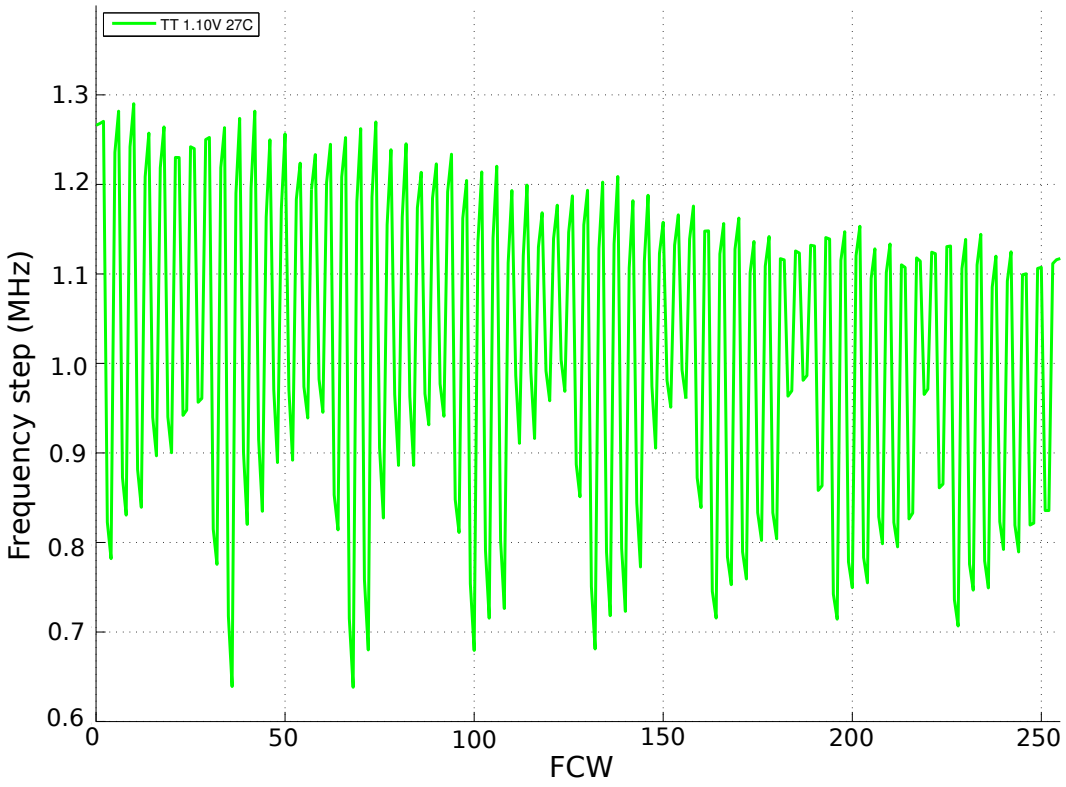


Figure 3.22: **Frequency step vs. FCW**: typical condition

PVT variations

Fig. 3.23, Fig. 3.24 and Fig. 3.25 shows the simulated code-frequency characteristics under different process, supply voltage and temperature conditions. We observe that the circuit has

a good immunity of temperature variation. The relative variation of the oscillation frequency compared with the typical condition is less than 4 % at 125°C, and is less than 2 % at 0°C. However, to achieve lower power consumption, smaller transistors are used compared with the first version DCO [74]. At a cost, the designed DCO is more sensitive to process and supply voltage variation. But it should be noted that the monotonicity and linearity of transfer function are always guaranteed in different working conditions, which is the most important thing for a network of coupled oscillators.

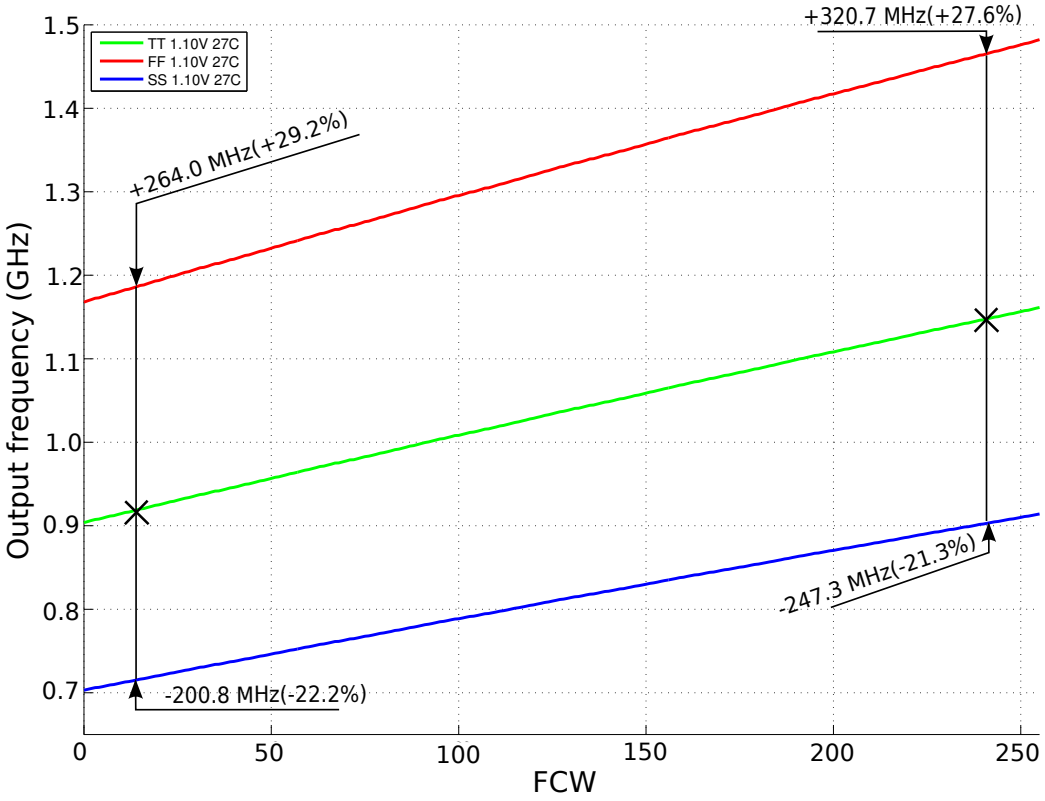


Figure 3.23: **Simulated output frequency versus frequency control word (FCW) in different process corners**: process variations: TT, FF and SS

A Monte-Carlo simulation is performed on the designed DCO working at its central frequency in typical condition. Fig. 3.26 displays the result with a 392 KHz interval.

Power consumption

To implement the proposed clock generator in a large dimension, the power consumption of each block, especially the DCO is essential. And it is one of the key improvements in the second version prototype. In typical condition, this block has a consumption of 1.1 mW working at central frequency and 1.2 mW working at its maximal frequency. Powers for different working conditions are traced in Fig. 3.27.

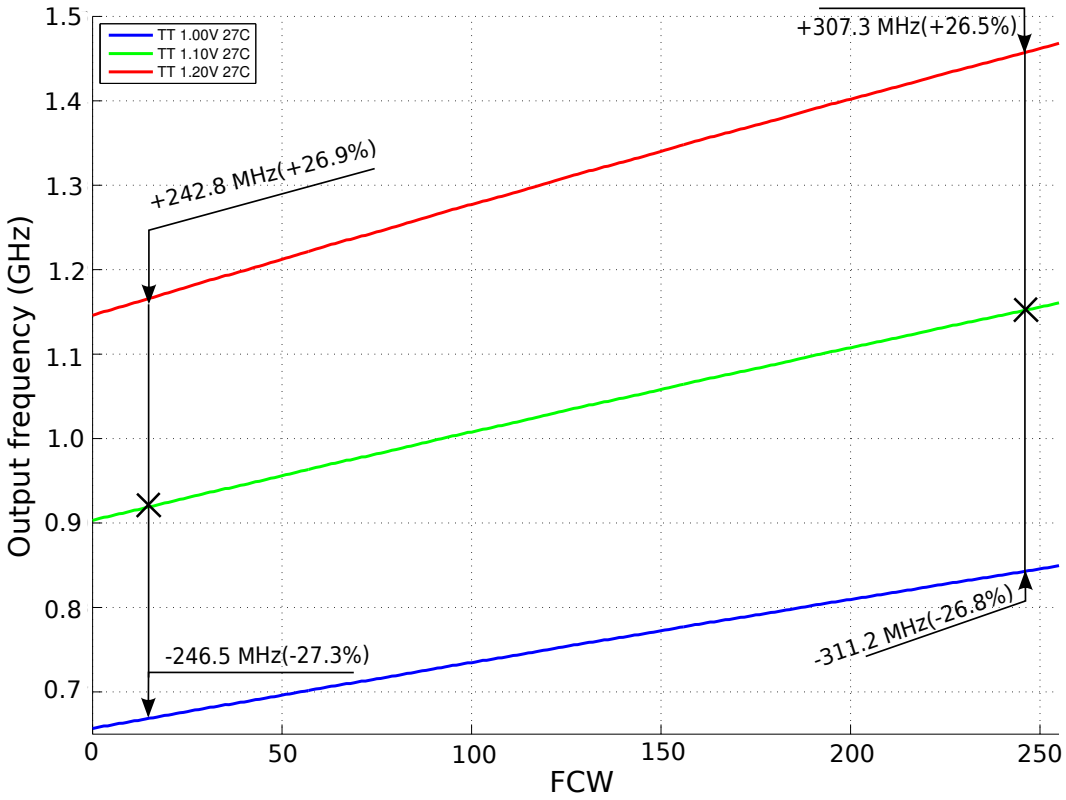


Figure 3.24: **Simulated output frequency versus frequency control word (FCW) with different supply voltages**: 1 V, 1.1 V and 1.2 V

Table 3.2: **DCO chip performance summary**

Parameter	***Value***
Central frequency	1.036 GHz
Output divided frequencies	259/129.5 MHz
Tuning range (typical)	903∼1161 MHz
Gain	1.01 MHz/LSB
Supply voltage	1.1 V
Power consumption	1.2 mW @ F_{max} or ∼1.17 mW/GHz
Area	∼2797.6 μm^2

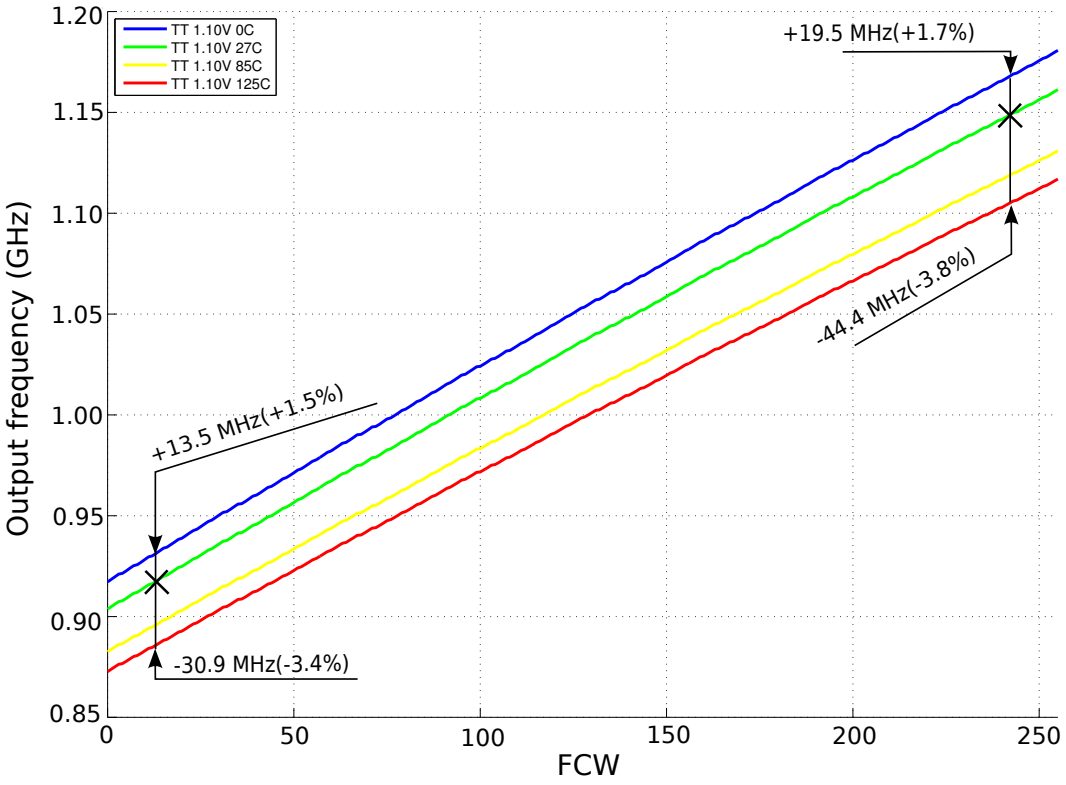


Figure 3.25: **Simulated output frequency versus frequency control word (FCW) at different temperatures**: 0°C, 27°C, 85°C and 125°C

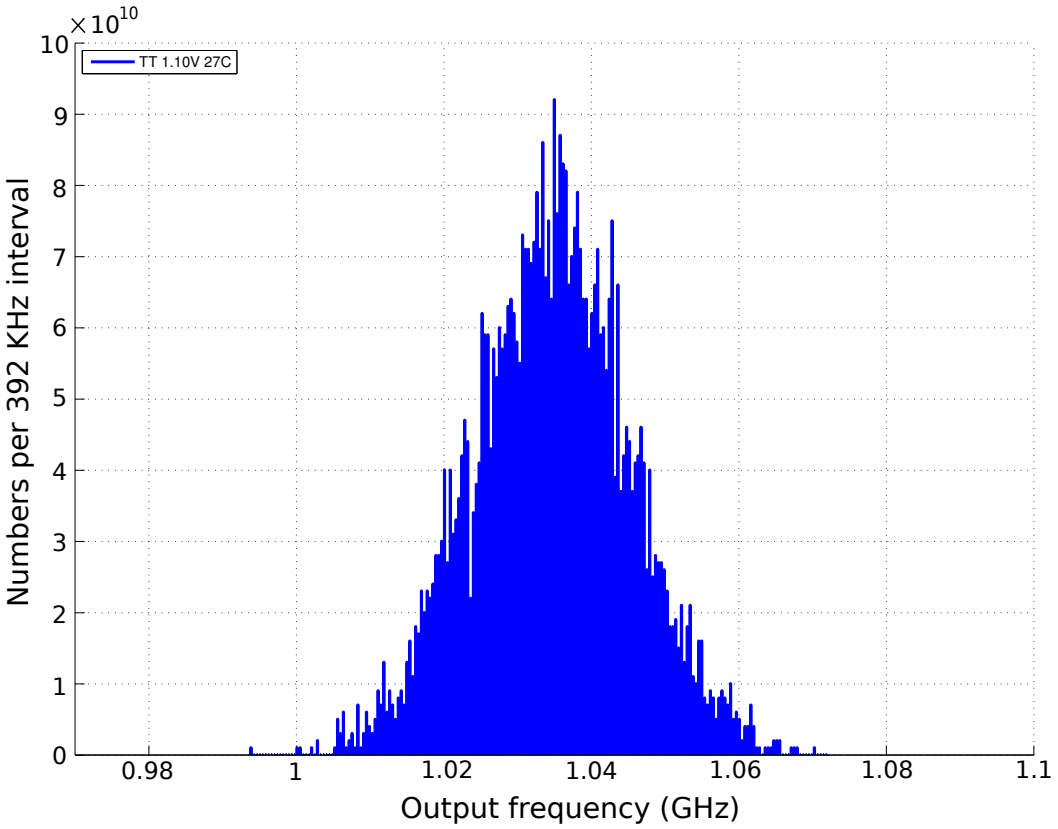


Figure 3.26: **Monte-Carlo simulation at center frequency**

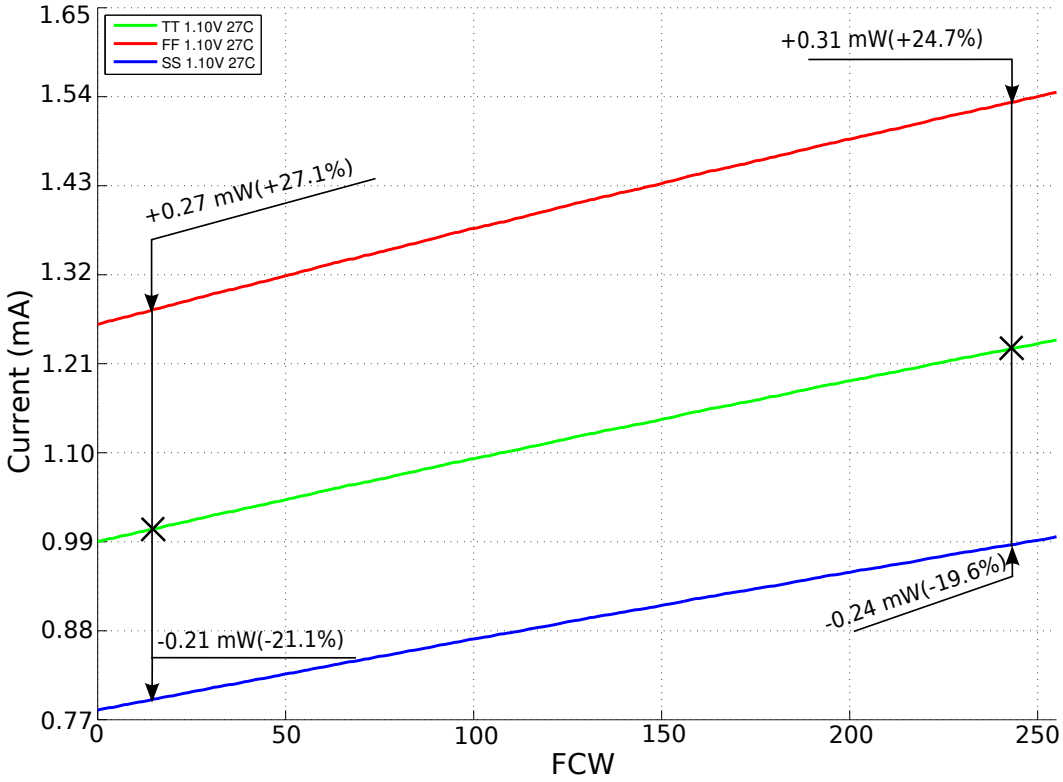


Figure 3.27: **Power consumption vs. FCW in different process corners**: process variations: TT, FF and SS

3.4 Conclusion

This chapter presents the digital blocks of the designed ADPLL. We have explained the principle of phase and frequency error measurement in PFD, which will be placed at the border between neighboring clock domains. We have also presented the architecture of loop filter and DCO, which are two local blocks generating the clock signal in each clock domain.

These blocks introduced in this chapter are essential components of one *NODE* in the coupled clocking network. In general, a block *NODE* is the whole circuit necessary for one locally synchronous clocking area. The implementation of this IP block is presented in detail in Chapter 6.

Chapter 4

Built-In Clock Error Characterization Circuit

Contents

4.1 Introduction 77
4.2 State of art 79
4.3 Test methodology 82
4.4 Low frequency discrete circuit prototype 87
4.5 High frequency on-chip prototype 90
4.6 Procedure of measurement 104
4.7 Conclusion 106

4.1 Introduction

In modern large-scale integrated circuit, a global clock distribution system is used to synchronize communications between various points in the system [57][52]. A synchronous communication requires the clocks at the transmitter and receiver sides to be synchronized, i.e., to have the same phase and frequency. In practice, a perfect synchronization is impossible: the must-be-simultaneous clock events are separated by time intervals called "clock error". The awareness of the maximal clock error is necessary for establishment of timing budget for communication and data processing.

In this chapter, we investigate a test strategy for characterization of clock error statistics between two clock domains in high-speed clocking systems (gigahertz and more). The method allows precise characterization of the clock error by observing the integrity of a periodic sequence transmitted between two clocking domains. The critical tasks of the measurement related to delay sensitivities and high speed signals are completely achieved on the chip; the readout of the results and the calibration of the instrument are achieved with an off-chip interface, with signals cadenced at low rate. The proposed technique aims at

picoseconds measurement resolution, without complex calibration procedure. The idea was first validated on a discrete prototype operated at downscaled frequencies, and then a high frequency on-chip prototype was designed using 65 nm CMOS technology. Simulation results predict a measurement precision of less than ± 2.5 ps. The chapter presents the theory, exposes the hardware implementation, and reports the experimental validation and simulation results of two prototypes.

This chapter is organized as follows. Section 4.2 presents state of the art of current clock error measurement methods, and does a brief comparison between them and the proposed circuit. Section 4.3 explains measurement principle and proposed circuit. In Section 4.4, the discrete device prototype and measurement results are presented. Post-layout simulation results of ASIC prototype are exhibited in Section 4.5.

4.2 State of art

In a synchronous circuit, clock signal is distributed in different places of the chip by a certain method, ex. clock tree. In an ideal case, all clock signals (the leaves of the tree) must be synchronized in phase and have the same period T. In practice, because of unmatched delays and noise, there are clock errors and fluctuation of the clock period. To formalize the method of clock error measurement, we make two hypotheses. 1) the average value of the period of all clocks is the same (T), 2) the clock error is small comparing to the average clock period. The second hypothesis means that the clocks are roughly synchronized, but some small perturbation exits. These two hypotheses are true in practically useful cases: the method we developed is intended to characterize small (residual) synchronization errors between clocks in an operational clock generator.

Let us consider two clock signals, $clk1$ and $clk2$. The two hypotheses presented above states that (i) for each period T there is in average one clock event for each signal, and (ii) it is possible to choose the numeration of the clock events so that to each clock event of $clk1$ number i corresponds to a clock event of $clk2$ with the same number, so that

$$\|\Delta t_i\| \ll T, \tag{4.1}$$

where

$$\Delta t_i = t_i^2 - t_i^1, \tag{4.2}$$

t_i^1 is the i^{th} edge arriving time of clk1, t_i^2 is the i^{th} edge arriving time of clk2.

We call Δt_i the clock error between two clocks clk1 and clk2, at the period number i (Fig. 4.1).

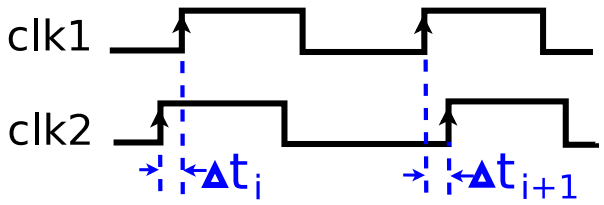


Figure 4.1: **Definition of the clock error** Δt_i

$\{\Delta t_i\}_{i\in\mathbb{N}}$ is a discrete time random process characterized by the time average $S_{in} = \overline{\Delta t}$ that we call *skew*[*] and the dynamic component of the phase error $\{\Delta t_i - S_{in}\}_{i\in\mathbb{N}}$. This process is usually considered as ergodic. The values of a realization of the process $\{\Delta t_i\}_{i\in\mathbb{N}}$ are characterized by a distribution function, which is also a probability density function (PDF) since the process is ergodic. In practice, a probability density function of a clock error is a function defined on a limited domain with minimal and maximal non-zero values B and C (for example, a truncated Gaussian distribution as in Fig. 4.2).

[*]Sometimes in the literature, the term "skew" is used as a synonym of "clock error". In this manuscript, we only employ this term according to the definition given in this paragraph.

The measurement of clock error can be required in many cases. This work is motivated by the need for characterizing clocks generated by the distributed clock generator presented before. The typical clock errors for gigahertz clocks are of tens of picoseconds or less. An off-chip measurement requires the transmission of clock signals off the chip, thus introducing additional delays whose values are difficult to control [68][65]. The sampling rate and the vertical gain of available measurement tools are also major issues.

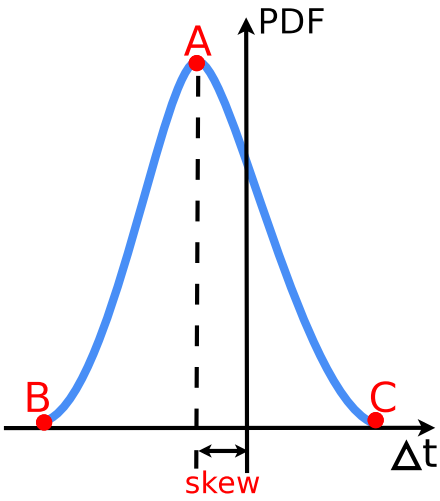


Figure 4.2: **Gaussian distribution of clock phase error**

On-chip solutions are usually based on time measurement techniques using a cascaded delay chain [13], cf. Fig. 4.3. When a local clock is sent down to the inverter chain, the sampling latches record the number of inverters that the clock edges travel through before being latched. Every sampling latch output is XNORed with its neighbor to obtain a 1 at the location of the detected edge. The result of the XNOR is written into a second register called the accumulation, or âĂIJstickyâĂİ register. However, the minimum delay of a CMOS buffer is of the same order as the clock error to be measured, and a precise measurement of small error is not possible. An improvement may be achieved by using a Vernier delay line [11] [10][16], explained in Subsection 3.1.2.

Precise measurement of the clock error may require a large number of stages (e.g., 129 in [13]). The outputs of the TDC stages are then processed by digital circuits at high rate. Although the methods using TDC provides precise value of timing error at each period, it is area and energy expensive, since in most cases only the statistics about the clock error are required (the mean, the minimum and the maximum values).

The technique proposed in this chapter is based on an integrity check of a periodic sequence transmitted from one clock domain to the other. If there is a clock error, the received sequence contains errors. The error rate is related to both static and dynamic errors between two clock domains. To characterize the process $\{\Delta t_i\}_{i\in\mathbb{N}}$, an externally controlled on-chip delay element is inserted in the test sequence data path. By varying the delay value, the error rate is modulated, and then the intrinsic clock error characteristics of the process $\{\Delta t_i\}_{i\in\mathbb{N}}$

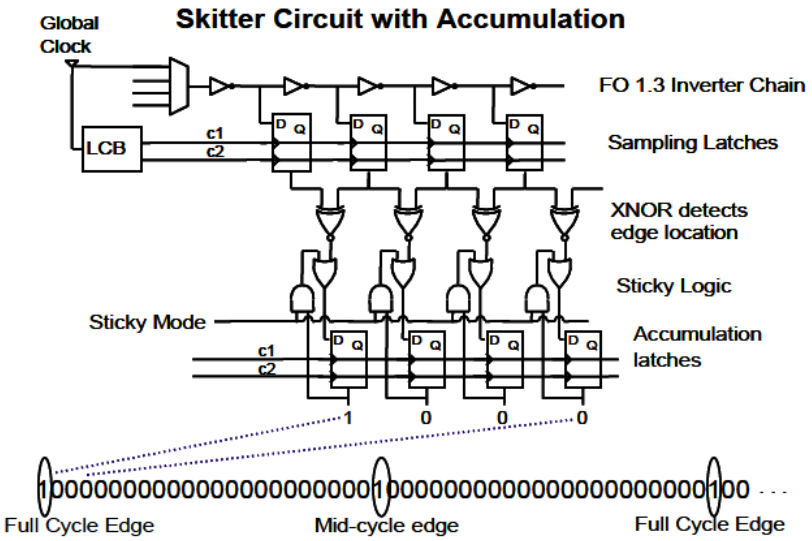


Figure 4.3: **Delay chain based "Skitter" circuit proposed in [13]**

can be abstracted by a simple processing unit. Only one bit is processed at each clock cycle, different from existing methods based on direct time interval quantification (e.g., 129 bits are processed in architecture described in [13]). The measurement resolution is defined by precision of the delay control. An on-chip delay with a high precision (few ps) can be achieved by techniques described in Section 4.5.

4.3 Test methodology

The proposed test method is based on the architecture presented in Fig. 4.4, which detects the variation of the sign of clock error Δt. It is done in the following way. The test pattern generator issues a periodic sequence "...010101..." which is synchronized with clk1. This pattern is transmitted to the clk2 domain, and a processing unit block checks its integrity. The received pattern is considered correct if and only if it is the same as the sent pattern. The circuit cannot detect the delay between two patterns since they are not in the same clock domain. Hence, if Δt_i sequence is constant (only a static error is present), the transmission is always correct (Fig. 4.5). This circuit cannot detect a pure static error (skew) between clocks.

However, if the clock error has a dynamic component – which is usually the case in practice – the clock error may change its sign. In this case, the sequence received by the processing unit contains two successive 0 or 1 (Fig. 4.6): the integrity of the sequence is violated and can be detected by the processing unit. After that, digital circuits can easily estimate the error rate ER. The error rate obtained from such a measurement is related to both the static and dynamic component of the clock error. For example, if the skew is greater than the amplitude of the dynamic component, the sign never changes, and the detected error rate is zero. If there is no skew, the sign change happens on average once over two clock events, and the measured error rate is 1/2.

Now we insert a controlled delay Δ in the transmission path(Fig. 4.4). This delay is equivalent to an additional skew between the clocks. By varying the delay Δ, we change the effective skew, hence, the measured error rate (ER), which is now a function of Δ. The next section shows the analysis providing the relation between the measured function $\mathrm{ER}(\Delta)$ and the unknown PDF of the clock error.

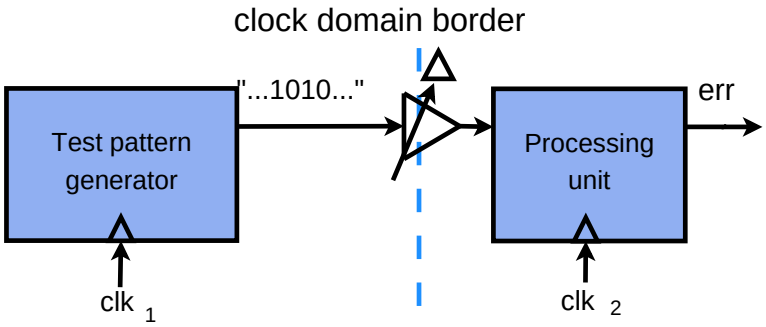
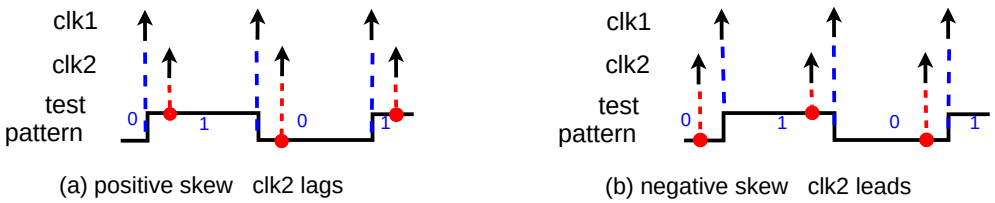


Figure 4.4: **Basic architecture of measurement circuit**

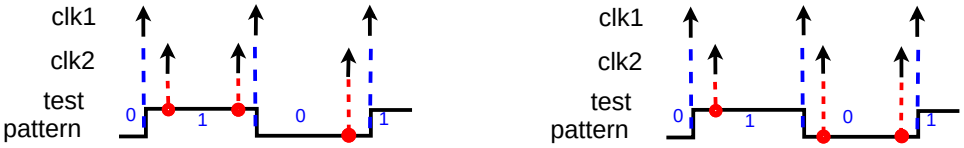
4.3.1 Measurement theory

The introduction of the delay Δ (Fig. 4.4) modifies the skew between the clocks, and hence the effective PDF of the clock error distribution, by mapping $\Delta t \rightarrow \Delta t + \Delta$ (Fig. 4.7(a,b)).



(a) positive skew clk2 lags (b) negative skew clk2 leads

Figure 4.5: **Received data integrity**: fixed static error



(a) dynamic error two '1's received (b) dynamic error two '0's received

Figure 4.6: **Received data integrity**: dynamic error

We can find a mathematical relation between the error rate ER and the original PDF of clock error by observing two neighboring clock cycles.

The error rate represents the probability of the sign change of the effective (modified by the delay) clock error. Considering that there is no dependency between Δt_i and Δt_{i+1} (cf. above in Section 4.2), we have :

$$\begin{aligned} \text{ER} &= P(\Delta t_i > 0 \; and \; \Delta t_{i-1} < 0) \\ &\quad + P(\Delta t_i < 0 \; and \; \Delta t_{i-1} > 0) \\ &= P(\Delta t_i > 0) P(\Delta t_{i-1} < 0) \\ &\quad + P(\Delta t_i < 0) P(\Delta t_{i-1} > 0) \\ &= 2P(\Delta t_i < 0) P(\Delta t_{i-1} > 0) \\ &= 2P(\Delta t_i < 0)(1 - P(\Delta t_{i-1} < 0)) \end{aligned} \tag{4.3}$$

where $P(\cdot)$ means "probability". Since the clocks errors are independent, we may replace $P(\Delta t_{i-1} < 0)$ by $P(\Delta t_i < 0)$, which we name a. We have:

$$\begin{aligned} \text{ER} &= 2P(\Delta t_i < 0)(1 - P(\Delta t_i < 0)) = \\ &= 2a(1-a), \end{aligned} \tag{4.4}$$

where $a = P(\Delta t_i < 0)$ is the probability that the i^{th} cycle phase error is negative (Fig. 4.7(c)). Note that a depends on Δ, and it is given by:

$$\begin{aligned} a(\Delta) &= P(\Delta t_i < 0) \\ &= \int_{-\infty}^{0} \text{PDF}(z + \Delta) dz = \int_{-\infty}^{\Delta} \text{PDF}(y) dy. \end{aligned} \tag{4.5}$$

When Δ is equal to the intrinsic skew S_{in}, on average, clock errors are positive in 50% of clock cycles and negative in the other 50% cycles. In this case the measured error rate is equal to $1/2$ and is maximal (cf. Fig. 4.7(c)). This allows a measurement of the intrinsic

skew. To find the original PDF, the expression (Eq. (4.5)) is derived:

$$\mathrm{PDF}(\Delta) = \frac{\partial a}{\partial \Delta}. \tag{4.6}$$

According to the definition of $a(\Delta)$, a is greater than 1/2 when the effective skew $S_{in} - \Delta$ is positive, otherwise, a is less than 1/2. Hence, from (Eq. (4.4)), a can be expressed as:

$$a = \begin{cases} \dfrac{1+\sqrt{1-2\mathrm{ER}}}{2} & \Delta \geq S_{in} \\ \dfrac{1-\sqrt{1-2\mathrm{ER}}}{2} & \Delta < S_{in} \end{cases} \tag{4.7}$$

By differentiating, (Eq. (4.7)) turns into the following equation:

$$\frac{\partial a}{\partial \Delta} = \frac{1}{\sqrt{1-2\mathrm{ER}}} \frac{\partial \mathrm{ER}}{\partial \Delta} \mathrm{sign}(S_{in} - \Delta). \tag{4.8}$$

where $\mathrm{sign(x)} = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$

From (Eq. (4.6)) and (Eq. (4.8)), we obtain:

$$\mathrm{PDF}(\Delta) = \frac{1}{\sqrt{1-2\mathrm{ER}}} \frac{\partial \mathrm{ER}}{\partial \Delta} \mathrm{sign}(S_{in} - \Delta). \tag{4.9}$$

The PDF(Δ) calculated by the formula (Eq. (4.9)) is non-zero only when the variation of function ER(Δ) doesn't equal zero. Hence, the maximal and minimal dynamic error values (points B and C in Fig. 4.7(a)) can be easily calculated. The largest and the smallest Δ values at which PDF(Δ) is non-zero are given by the limit of the Δ range at which ER(Δ) is non-zero (points B and C in Fig. 4.7(d)). Indeed, the delay $\Delta = \Delta_B$ or $\Delta = \Delta_C$ shifts the original PDF rightward/leftward until $a = 0$ or $a = 1$. Hence, Δ_B and Δ_C correspond respectively to Δt values at points B and C of original PDF (Fig. 4.7(a)), which are the min/max clock error values.

However, to abstract the full original PDF of clock error, Δ must be able to take either positive or negative values. However, a real delay is always positive. A negative delay can be imitated by two methods. The first way is to use a single variable delay whose value is close to the clock period. The relation between Δ and the implemented physical delay Δt_d can be expressed by the following equation:

$$\Delta = (T/2 + \Delta t_d) \bmod T - T/2. \tag{4.10}$$

In this case, the relation between the ER distribution and the real delay value is illustrated in Fig. 4.8, where delay values at points A, B, and C correspond to the skew and min/max clock errors respectively.

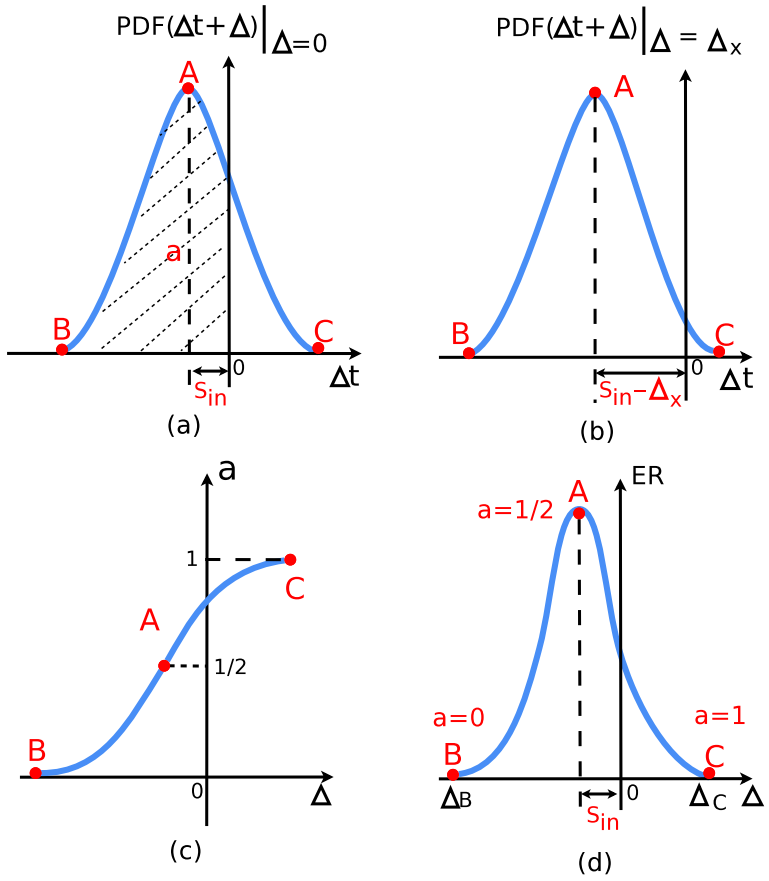


Figure 4.7: **Theory analysis**: (a) Original PDF of clock uncertainty; (b) PDF with a shift $\Delta = \Delta_x$; (c) a vs. Δ; (d) ER vs. Δ

The second way to have a negative delay is to use a differential delay pair – one delay for each clock domain. Therefore, the real delay introduced between two clocks is the difference value between two delays. If the range of a single delay is [a, b], that of a differential delay using two identical delays is [-(b-a), b-a].

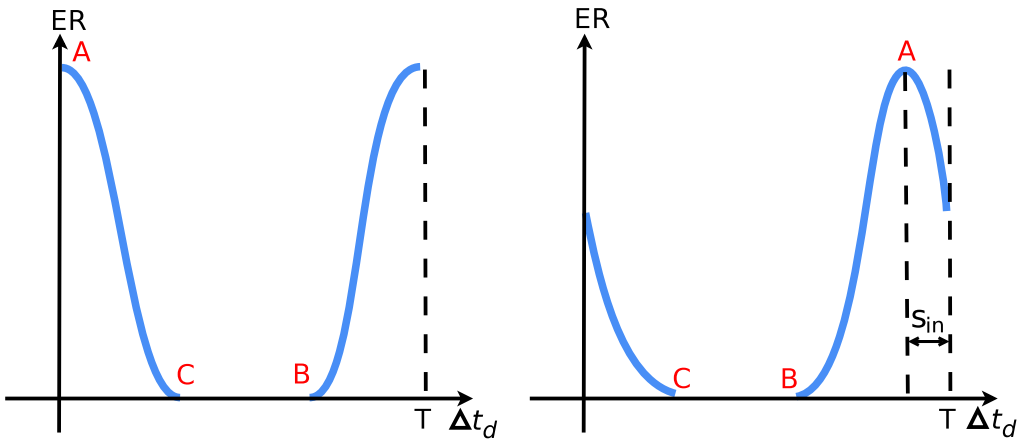
We have studied the two techniques in our work: the former technique has been implemented in a low frequency discrete circuit prototype (cf. Section 4.4); the latter has been implemented in a high frequency ASIC prototype (cf. Section 4.5).

4.3.2 Architecture of measurement circuit

The measurement circuit is implemented as shown in Fig. 4.9.

A binary sequence "...1010..." is generated on-chip in clk1 domain by a D flip-flop synchronized with the clock clk1 (Fig. 4.9). The binary sequence is sent to the clk2 clock domain after a controllable delay (Δ), which allows varying the effective static error between two clocks manually.

The data sequence is sampled in the clk2 domain. To reduce the probability of metastability in flip-flops [19], a 4-stage shift register samples the input signal (D2) of the clk2



(a) Intrinsic skew is null (b) Intrinsic skew is negative

Figure 4.8: **ER distribution with respect to single positive delay**

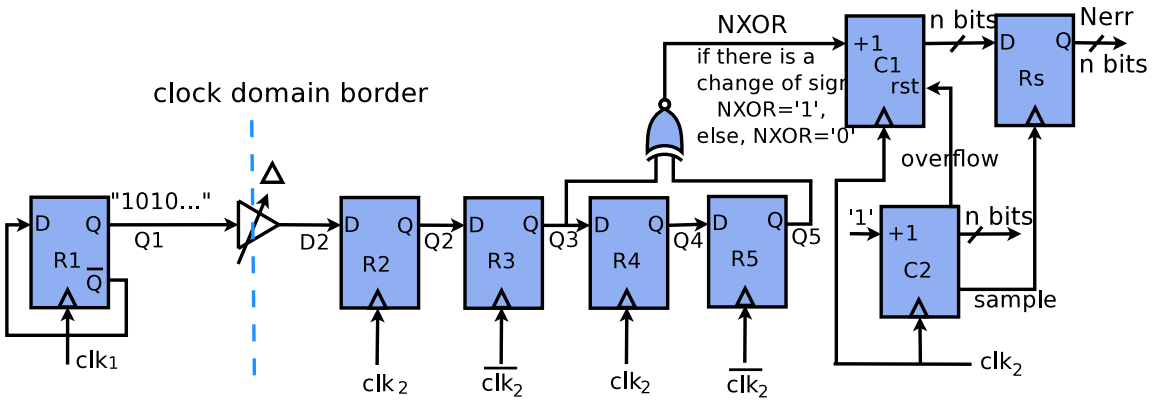


Figure 4.9: **Architecture of test circuit**

domain. A 2-input NXOR gate detects transmission errors by comparing R3 and R5 outputs; each detection event is then counted by the n-bit counter C1 cadenced by clk2. Counter C2 has two roles. The first one is to generate an *overflow* event every 2^n clk2 cycles: this provides a time interval during which C1 counts the errors. When C2 overflows, the counter C1 is reset to 0 to restart counting in the next 2^n clk2 cycles. The second role of C2 is to generate another signal *sample*, which is used as a sampling clock signal. When the rising event of *sample* arrives, the value of C1 is written into the output register Rs. The event of *sample* is generated one period before the event of *overflow* to avoid the concurrency between sampling and reset of the value of C1. Therefore, the value stored in the register Rs represents the number of errors during $2^n - 1$ clk2 cycles and it can easily be transmitted off-chip, because the readout is at a frequency 2^n lower than the clock frequency.

4.4 Low frequency discrete circuit prototype

A low frequency prototype using discrete devices is implemented to evaluate the proposed test method. The test environment is shown in Fig. 4.10.

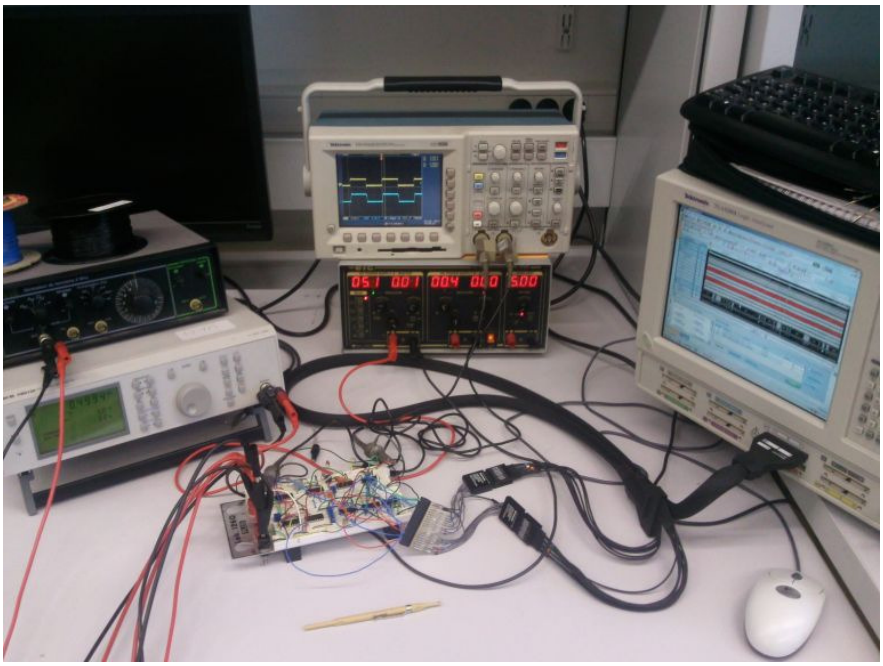


Figure 4.10: **Measurement environment of prototype**

Two 8-bit counters (C1 and C2 in Fig. 4.9) are implemented for sign changes rate calculation. A 8-bit register (Rs) samples the counter result every 256 clock cycles. Negative delays are implemented with the first technique described in Subsection 4.3.1.

clk1, a 500 KHz signal with a 50% duty cycle, is generated by a function generator. The clock clk2 with dynamic error and skew is generated from clk1 by the circuit in Fig. 4.11. RC delay provides a positive delay (a negative skew is obtained by a large delay close to the clock period, cf.Subsection 4.3.1). A clock edge uncertainty with max/min value of ± 100 ns is achieved by adding a noise signal over the power supply of inverter. To eliminate glitches, a Schmitt trigger is used at the output.

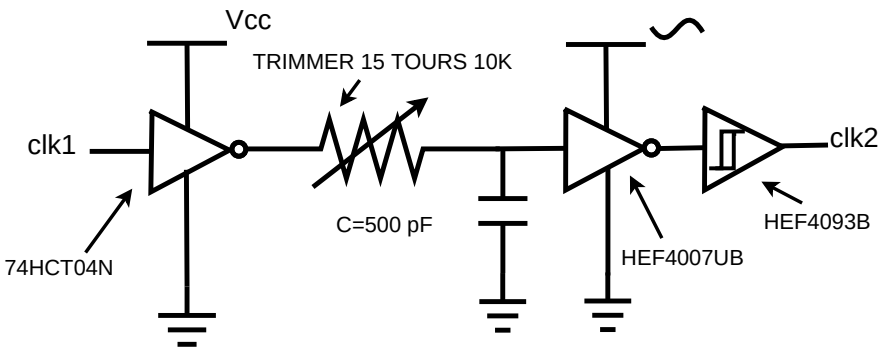


Figure 4.11: **Generation of clk2 with static and dynamic errors**

The controllable variable delay Δt_d is implemented as a 4-stage delay chain shown in Fig. 4.12, in which the resistor is a trimmer. Four delay stages could generate a large delay with a value comparable to the period of the sequence Q1. To acquire the function $ER(\Delta t_d)$, the delay Δt_d varies from 0 to 2 μs and the output ER value is observed.

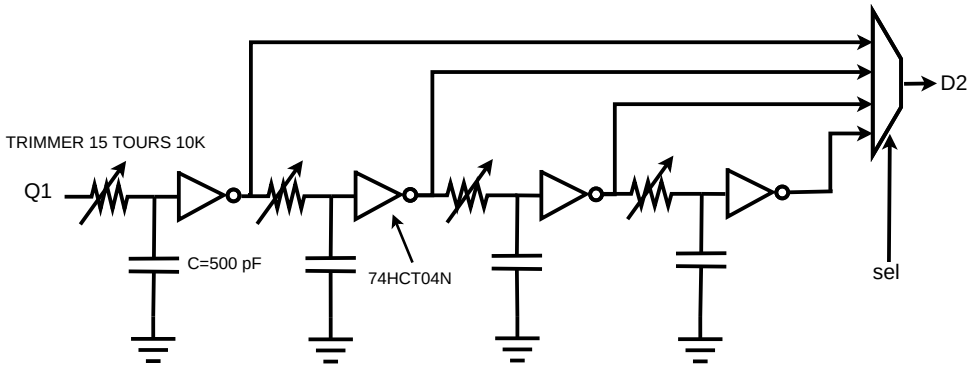


Figure 4.12: **Variable delay circuit in discrete circuit prototype**

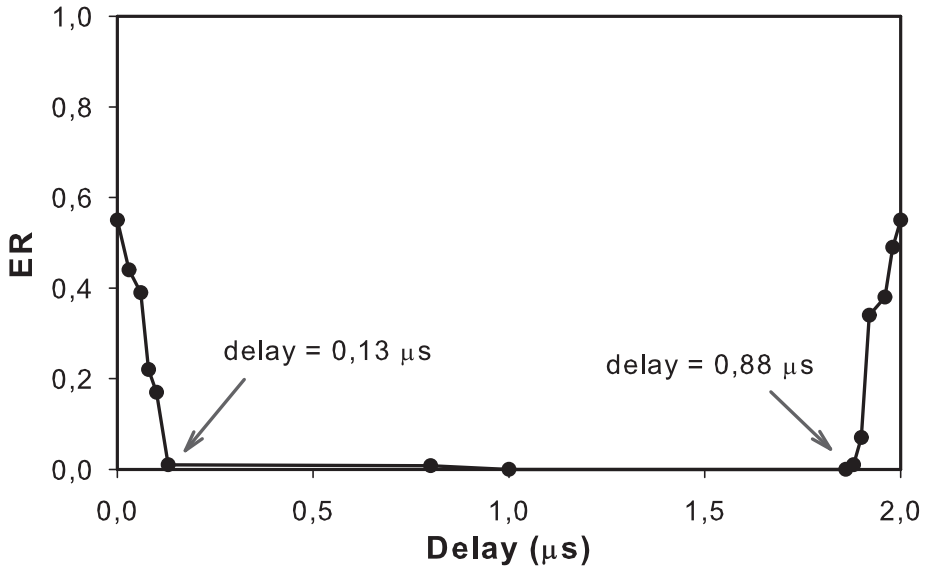


Figure 4.13: **Test of prototype without skew**

We have performed three tests to evaluate the prototype. In the first test, only pure dynamic errors without skew ([-100 ns, 100 ns]) are added. The measured error range is [-120 ns, 130 ns] as illustrated in Fig. 4.13. In the second test, a negative skew of 40 ns exists along with $\pm$100 ns dynamic errors. Test result shown in Fig. 4.14 (mean value -50 ns, range [-150 ns, 80 ns]) is in good agreement with the values we measured directly (mean value -40 ns, range [-140 ns, 60 ns]). In the third test, a positive skew of 110 ns with $\pm$50 ns dynamic errors are added on clock 2. The measured result has a skew of 100 ps with a range of [43 ns, 168 ns], which has a difference of less than 10 ns with respect to ideal values. The obtained plots (Fig. 4.13, Fig. 4.14) and Fig. 4.15) are qualitatively similar to those predicted by theory (Fig. 4.8).

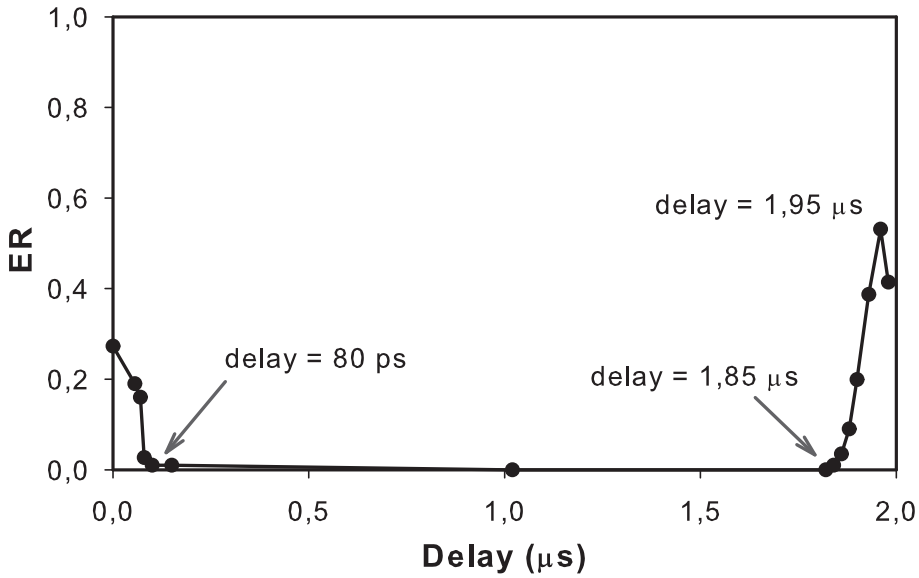


Figure 4.14: **Test of prototype with a skew of -40 ns**

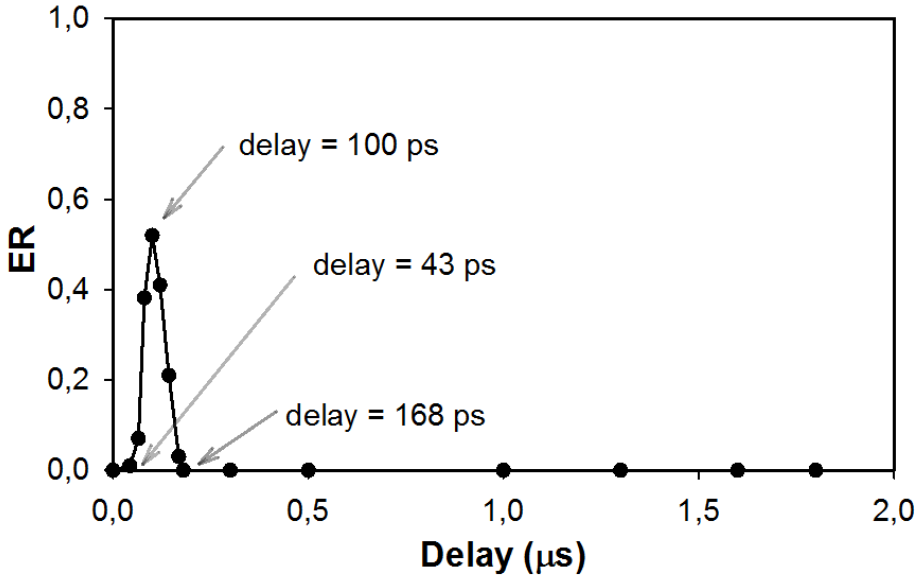


Figure 4.15: **Test of prototype with a skew of -40 ns**

4.5 High frequency on-chip prototype

The prototyped architecture has a serious drawback for the on-chip implementation, in what concerns the implementation of the delay Δ: First, an approximation of a pure delay by RC networks is only efficient for delays largely inferior to characteristic time of the signals. However, the implementation of negative Δ requires pure delays of the same order as the signal period. Second, small values of Δ are difficult to implement, because of the threshold imposed by intrinsic technology delays. To overcome these difficulties, Δ is defined as differential delay. The new topology of the on-chip measurement system is shown in Fig. 4.16. One variable delay Δt_{d1} is used in the test pattern path, the other delay Δt_{d2} is applied to the clock *clk*2. As explained in Section 4.3.1, the effective delay value is $\Delta = \Delta t_{d1} - \Delta t_{d2}$, which can be a very precise value (either positive or negative) close to zero. Moreover, the delay range is doubled. Implementation details and characteristics of proposed variable delay are presented in Subsection 4.5.1.

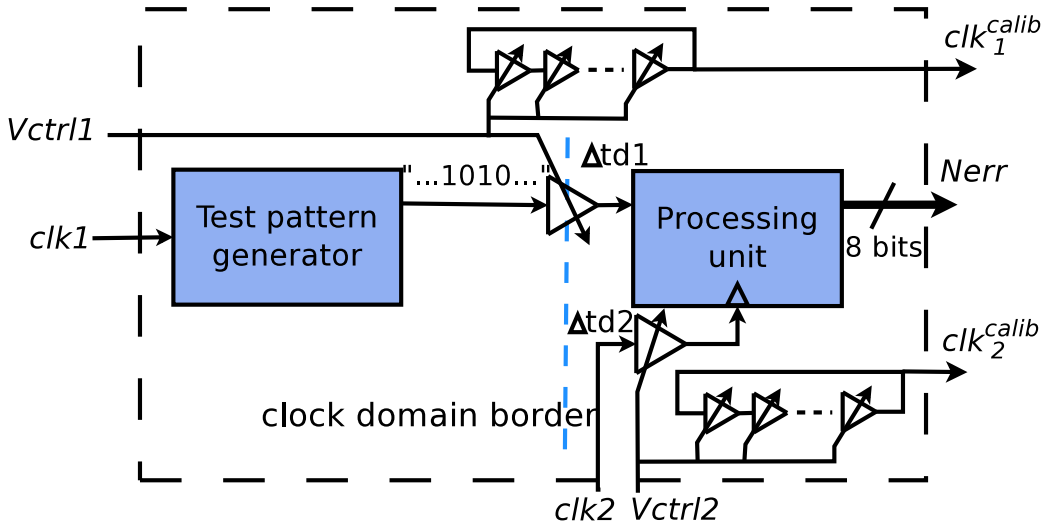


Figure 4.16: **ASIC prototype architecture**

4.5.1 Voltage-controlled delay (VCD)

VCD topology

The voltage controlled delay element (VCDE) is based on the topology in Fig. 4.17. It is composed of two CMOS inverters. The charging and discharging currents of the output capacitance of the first inverter (M4-M5) are controlled by a PMOS (M2) and a NMOS (M7), respectively. M1 and M6 constitute a current mirror for controlling the gate voltage of M2 and M7. Hence in this delay, both the rising and falling edges of the input signal can be controlled. The second inverter (M8-M9) improves the rise and fall times of the circuit.

In this prototype, to have a large delay range, two VCDE as shown in Fig. 4.17 are cascaded (Fig. 4.18). To be aware of the delay during measurement, a replica of variable delay is repeated and set as a ring oscillator. Since the exact delay value is known during

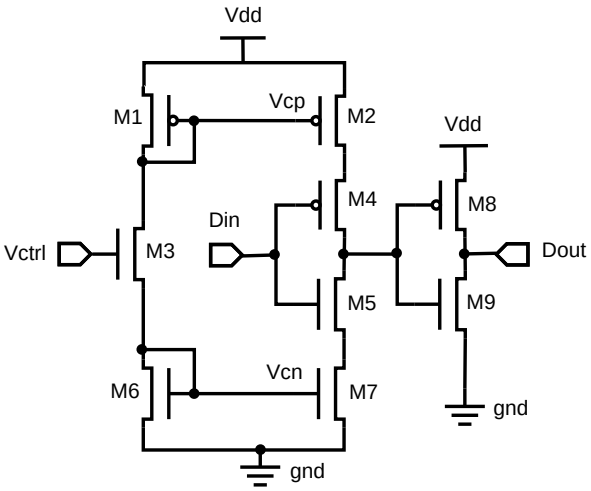


Figure 4.17: **Voltage-controlled delay element[24]**

the test, a complex calibration is no more necessary. It should be noted that in order to measure exactly the value of variable delay, each VCDE and each replica of VCD should have the same load capacity. That is the reason why inverters with the same size are added at the output of the replica. Moreover, since a critical requirement for ring oscillator is that it should contain an odd number of inverters, three inverters are used in VCD. Another inverter is added between the output of VCD and the input of processing unit not only to have a non-inverted delay but also to make sure the VCD has same load as its replicas in the ring oscillator.

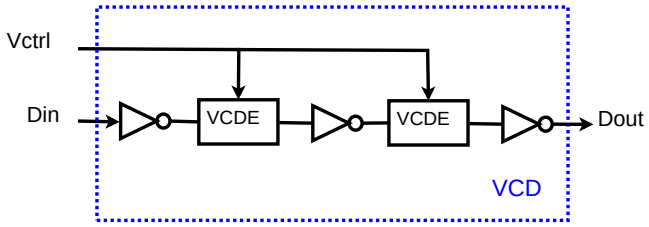


Figure 4.18: **Voltage-controlled delay with 2 stages**

VCD layout design

To guarantee the compatibility with digital circuit and facilitate place and route, we use the cell oriented design methodology for layout design of the delay element. In the cell oriented design, the functional blocks are implemented as cells having common geometrical parameters. For compatibility with the standard library cells, the custom VCD cell must have the same geometrical constraints as the standard design kit cells:

1. The height of the cells;
2. The topology of the supply/ground wires;
3. The location and maximal dimensions of substrate and N-well polygons

Fig. 4.19 demonstrates a template of the cell with prerouted supply and reserved space for active transistors and routing wires. The prerouted supply and ground are done in metal layer Metal1. The width of the power strips is 0.56 μm. They are shared with the neighboring cells from bottom and top sides.

The routing of signals is free within the cell area at the levels of metal Metal2-Metal7. The routing space is only limited by the supply strips at metal layer Metal1.

The active zones of transistors should be placed inside the area specified by the red dotted rectangle. The N-well zone is specified by the blue rectangle area: it is reserved for P transistors. The green rectangle specifies the P-well zone for the N transistors.

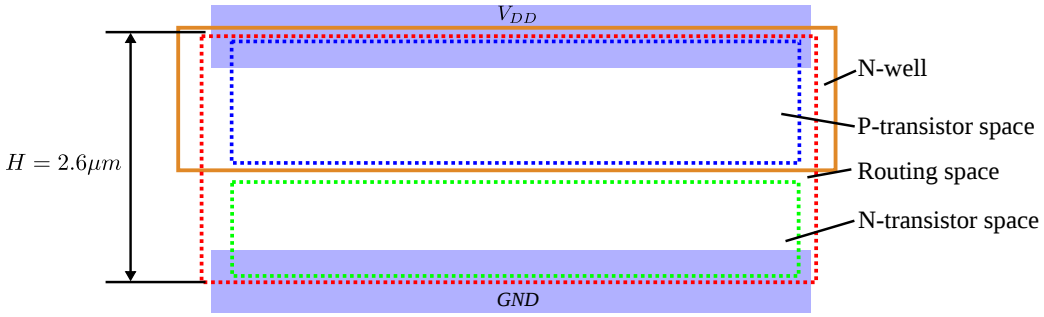


Figure 4.19: **Cell layout template:** routing space includes the whole cell area, the N-well defines the location of the P-transistors. The height of the cell (2.6 μm) and the supply/ground polygons size are the same as those of the standard library cells.

However, in order to get better matching in current mirror, M1-M3, M6 and M7 should have large size, and the 2.6 μm space for active transistors is not enough. To have enough space for sizing without violating geometrical constraints of standard cells, we created a new template (illustrated in Fig. 4.20) whose height is three times the one shown in Fig. 4.19. The borders of the template are standard filler cells of design kit. At each side, three cells are connected by abutment. Since the V_{DD} and GND stripes alternate, each pair of neighboring cells must have an vertical symmetry, hence the middle cell is mirrored with respect to the other two cells. By doing this, the placement space is three times the standard cell, and no DRC rule is violated thanks to the standard cell border. The layout of one variable delay element is displayed in Fig. 4.21.

VCD characteristics

Fig. 4.22 shows the delay-voltage relations at different input signal frequency conditions and the delay measured by ring oscillator. From Fig. 4.22, we can observe that if the control voltage is greater than 0.85 V, the delay value used for manipulation matches the value measured by oscillator with a difference of less than 2 ps. The delay range is [333.89 ps, 435.97 ps] for one delay, hence for the differential delay the range is [-102 ps, 102 ps]. The latter range is large enough comparing with typical phase errors between synchronized high frequency on-chip clocks. The resolution of the differential delay control is about 2 ps/5 mV at high

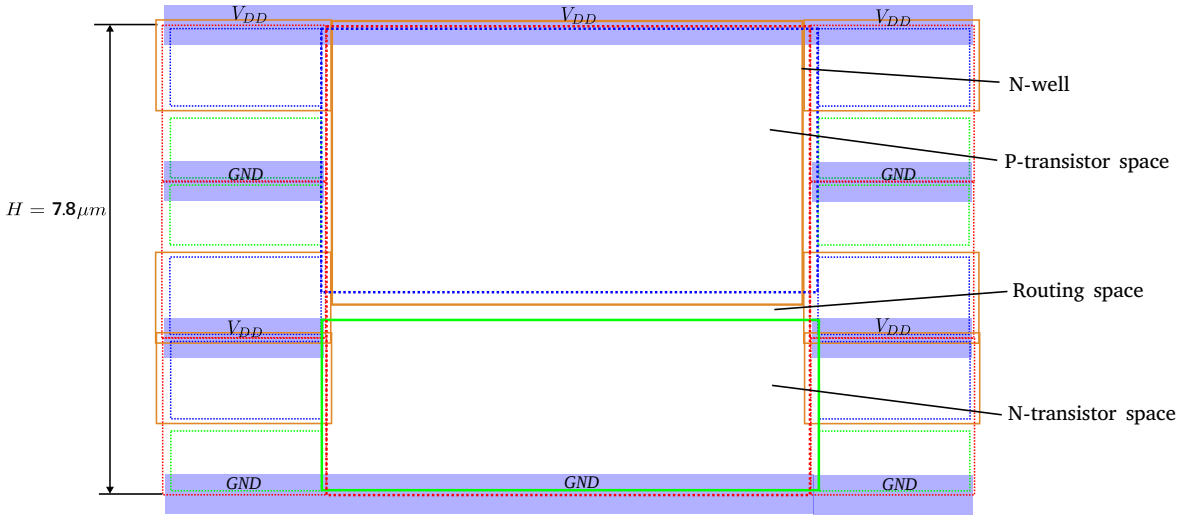


Figure 4.20: **Delay element layout template:** routing space includes the whole cell area, the N-well defines the location of the P-transistors. The height of the cell (7.8 μm) and the supply/ground polygons size are the same as those of the standard library cells.

control voltage and 5 ps/5 mV at low control voltage. A step of 5 mV with a precision of less than 0.05% can be achieved by a modern voltage supply device (ex. Agilent 6625A Power Supply); therefore we can have a precision of ± 1 ps for small clock error measurements and ± 2.5 ps for large error.

4.5.2 Physical design of test circuit on silicon

As illustrated in Fig. 4.23, five replicas of voltage controlled delay block (VCD_0) are flipped horizontally (VCD_{1-2}) or vertically (VCD_{3-5}) and connected as a ring oscillator. The six delay blocks are placed in two rows. The connection $VCD_2 \rightarrow VCD_3$ and $VCD_1 \rightarrow VCD_5$ are routed manually, and the rest connections between blocks in the ring oscillator are realized by abutment. A control voltage signal $Vctrl$ is routed in the space between the two rows so that all variable delays share the same signal. This compact layout is displayed in Fig. 4.25.

In practice, the position of *Pattern Generator* and variable delay Δt_{d1} in Fig. 4.16 can be swapped without affecting the functionality of circuit. In this way, the two digital blocks *Pattern Generator* and *Processing Unit* can be realized in one digital circuit. This has two advantages: First, it makes the whole circuit symmetric (as shown in Fig. 4.24), which decreases the mismatch; Second, it helps optimizing the digital circuit during synthesis and place & route.

The layout of the proposed test circuit is shown in Fig. 4.25.

4.5.3 Modeling of clock generator for system verification

In order to verify the performance of proposed circuit, we need a model of DUT (Device Under Test), which generates two clock signals with deterministic static and dynamic phase

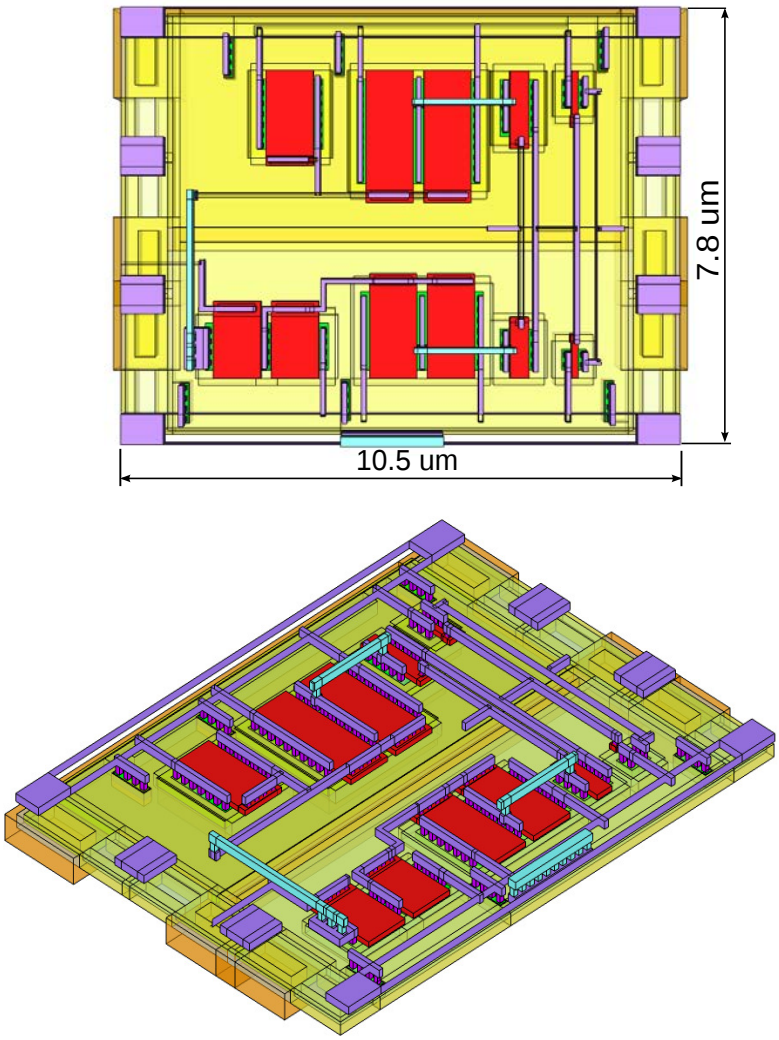


Figure 4.21: **layout of VCD and calibration oscillator**

errors, with close-to-realistic statistical properties of the error sequence. The test sequence should be repeatable, so that we could compare measurement results with original values, and expect the proposed circuit to give the same accurate value even with process, voltage supply or temperature (PVT) variations.

Because of its high controllability, observability, simulation speed and precision, an ideal candidate of DUT is a VHDL model. In the clock generator model, a clock signal with a frequency f can be generated by the instruction $clock <= not\ clock\ after\ T/2$, where T is the period ($T = 1/f$). The jitters are modeled by adding a random Gaussian variable Yn to the value T at each period of the output signal. This variable represents the additive random fluctuation at each ideal time-stamp of clock signal. VHDL library does not provide a dedicated command for the generation of random variables with normal distribution. Instead, it provides a function generating pseudo-random numbers with uniform distribution, which can be used by Box-Muller transform to generate a Gaussian variable. The Box-Muller transform generates a normally distributed random variable from the two uniformly distributed random

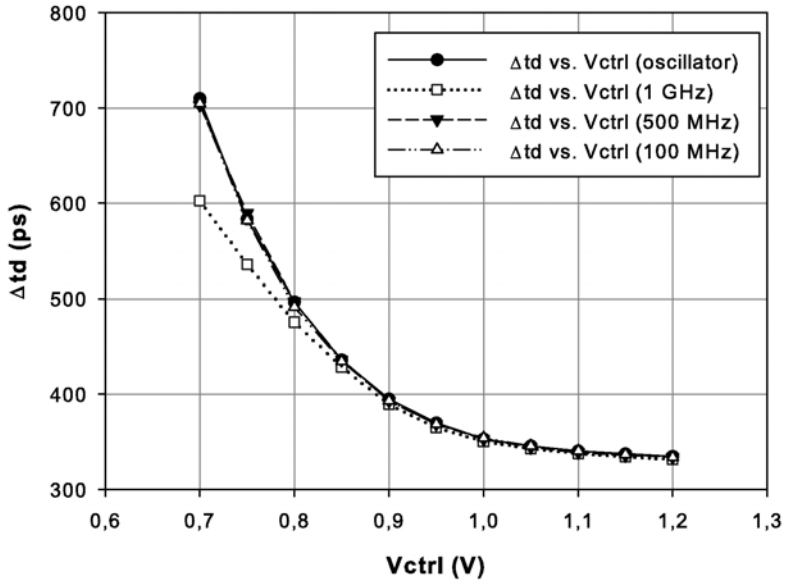


Figure 4.22: **Variable delay in function of control voltage**

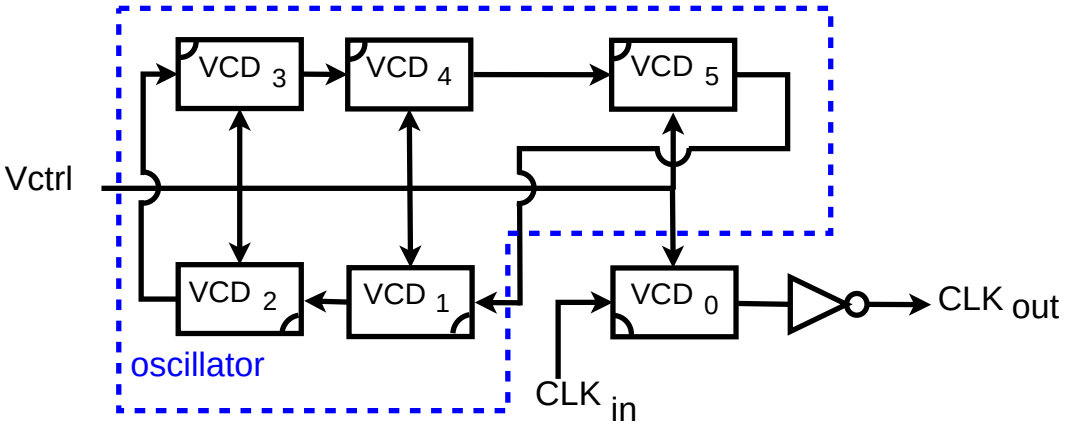


Figure 4.23: **Place of VCD and calibration oscillator**

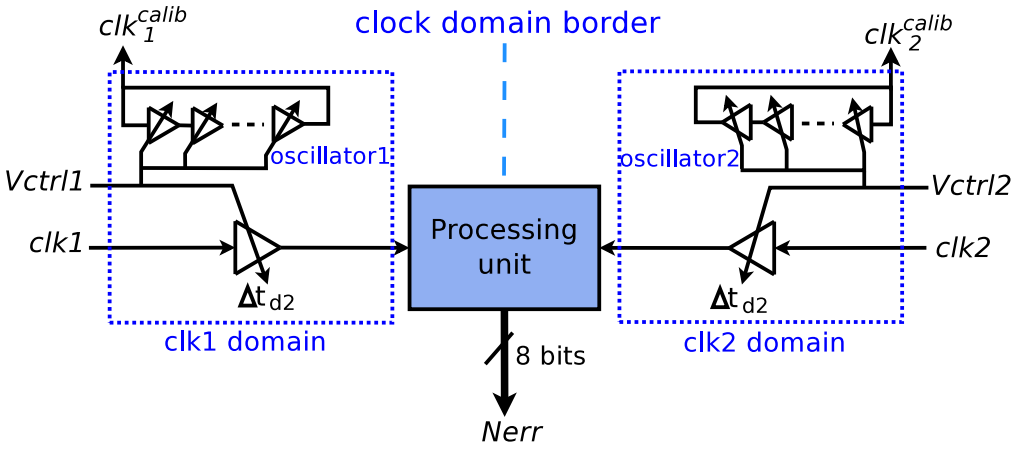


Figure 4.24: **Architecture of implemented built-in test circuit**

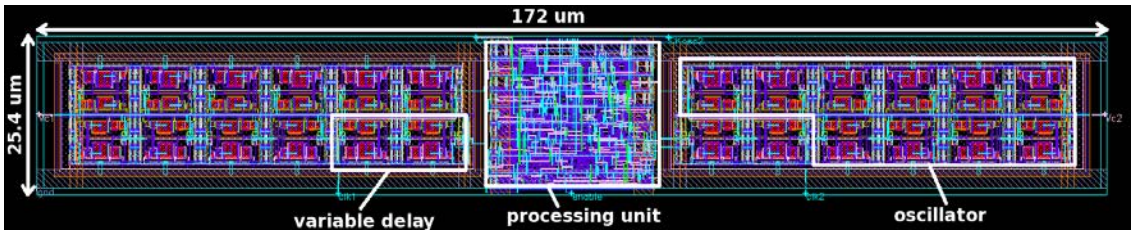


Figure 4.25: **Layout of built-in test circuit**

variables. This transform is given by

$$Y_n = \sqrt{-2\ln x_1}\cos(2\pi x_2) \tag{4.11}$$

where Y_n is a normally distributed variable with expected value 0 and variance 1; x_1 and x_2 are independent random variables uniformly distributed in the interval [0, 1].

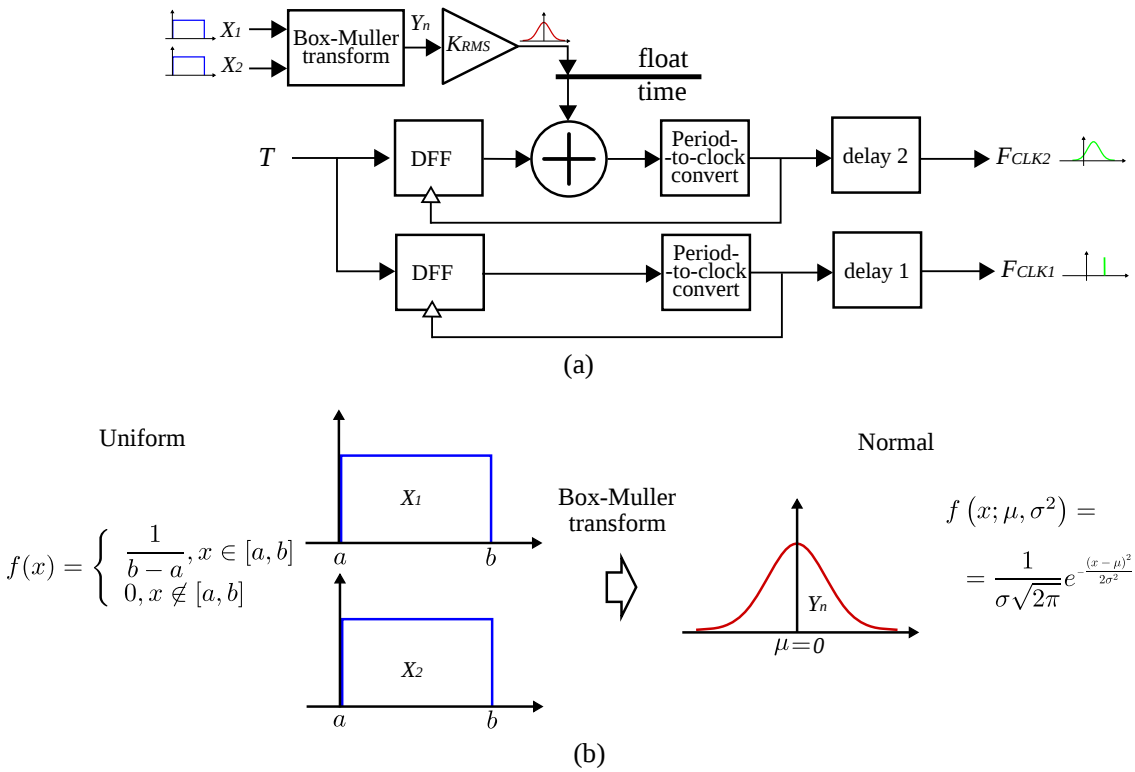


Figure 4.26: **VHDL model for clock generation**: (a) block diagram and (b) generation of a random variable with normal distribution using Box-Muller transform.

Once we get the Gaussian variable, we can inject dynamic errors by multiplying the variable by a RMS (root mean square) value and adding the result to ideal period value T. By adjusting the delay value in each clock generation path (Fig. 4.26(a)), we can introduce a static error, which could be both positive and negative. The total relative phase error of $clk2$ with respect to $clk1$ can be expressed as:

$$\text{phase error} = \text{jitter} + \text{delay2} - \text{delay1} \tag{4.12}$$

The complete listing of the precise VHDL macro-model is given in Appendix D.

4.5.4 Simulation results

To evaluate the proposed circuit, we performed various post-layout simulations. In the test-bench, normally distributed phase errors are added between two 1 GHz clock signals. The mean value (skew) is -20 ps and max/min error values are 19.6 ps/-53.4 ps. The histogram of the generated clock error distribution during 1020 cycles is displayed in Fig. 4.27.

The error rate (ER) is calculated as the total number of errors during 4 cycles of test (255 periods each cycle for the reason explained in Subsection 4.3.2), which yields 1020 clock periods. The measured ER(Δ) curve is drawn in Fig. 4.28. Tab. 4.1 records detailed results of each step. The measurement allows a localization of the minimum error (Δ_B in fig. Fig. 4.7a) in the interval of [-55.62 ps, -53.21 ps), whereas the real minimum error value is -53.4 ps, a precision of [-2.22 ps, 0.19 ps) is achieved. In the same way, the maximum error (Δ_C in fig. Fig. 4.7a) is between 16.62 ps and 20.3 ps, which has a difference lying in the range (-2.98 ps, 0.7 ps] with respect to the real value 19.6 ps. If we define the median value of the range as an estimation of real value, we find that the estimation error is within $\pm$2 ps. The measured skew is -20.73 ps, which is 0.73 ps less than the real value. The measurement results demonstrates a good precision in accordance with the theory.

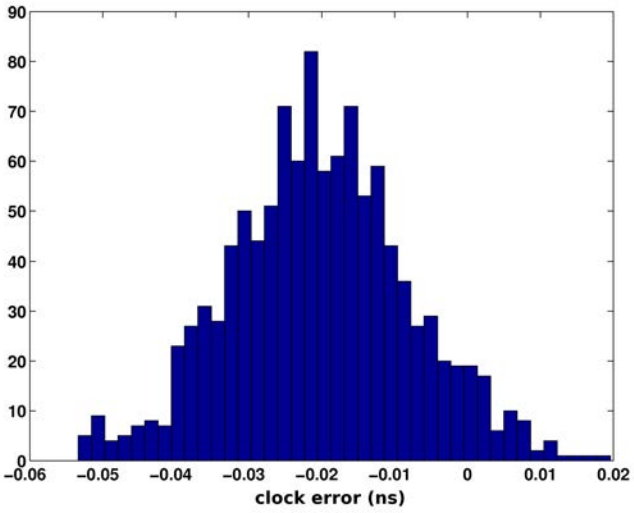


Figure 4.27: **Histogram of clock errors between clk1 and clk2**

Temperature variation immunity

Fig. 4.29 shows the measured voltage-delay characteristics under different temperature conditions. We observe that the circuit has a good immunity of temperature variation. At high

Table 4.1: **Post-layout simulation results (T = 27 °C)**

Vctrl1 (V)	Vctrl2 (V)	Δt_{d1} (ps)	Δt_{d2} (ps)	Δ (ps) $(\Delta t_{d1} - \Delta t_{d2})$	err (255 periods each cycle)					Error rate
					1^{st} cycle	2^{nd} cycle	3^{rd} cycle	4^{th} cycle	total	
0.95	0.87	418.57	474.19	-55.62	0	0	0	0	0	0
0.945	0.87	420,98	474,19	-53,21	0	2	0	0	2	0.001968504
0.945	0.875	420.98	469.76	-48.78	10	2	2	8	22	0.021653543
0.945	0.88	420.98	465.5	-44.52	14	4	4	12	34	0.033464567
0.945	0.885	420.98	460.97	-39.99	30	14	10	22	76	0.07480315
0.945	0.89	420.98	456.78	-35.8	29	33	44	46	152	0.149606299
0.945	0.895	420.98	452.76	-31.78	70	58	47	58	233	0.229330709
0.945	0.9	420.98	448.88	-27.9	84	75	70	75	304	0.299212598
0.945	0.905	420.98	445.2	-24.22	104	93	117	109	423	0.416338583
0.945	0.91	420.98	441.71	-20.73	128	127	122	134	511	0.502952756
0.945	0.915	420.98	438.36	-17.38	123	122	124	122	491	0.483267717
0.945	0.92	420.98	435.15	-14.17	114	113	112	112	451	0.443897638
0.945	0.925	420.98	432.04	-11.06	71	82	88	86	327	0.321850394
0.88	0.875	465.5	469.76	-4.26	42	48	30	46	166	0.163385827
0.88	0.88	465.5	465.5	0	30	32	18	32	112	0.11023622
0.88	0.885	465.5	460.97	4.53	12	16	10	10	48	0.047244094
0.88	0.89	465.5	456.78	8.72	0	4	6	0	10	0.00984252
0.88	0.895	465.5	452.76	12.74	0	2	4	0	6	0.005905512
0.88	0.9	465.5	448.88	16.62	0	0	0	2	2	0.001968504
0.88	0.905	465.5	445.2	20.3	0	0	0	0	0	0

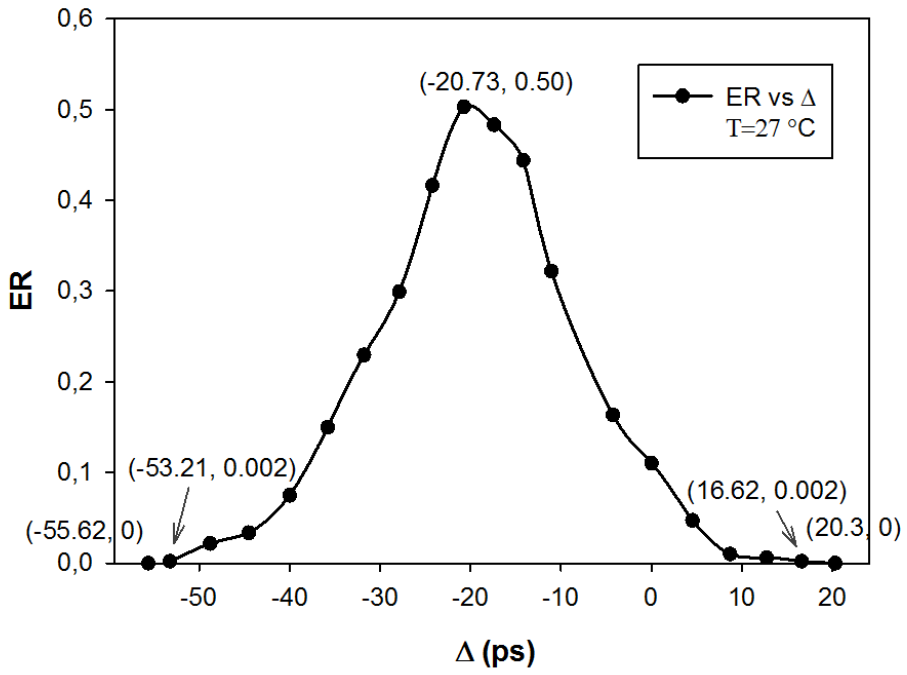


Figure 4.28: **ASIC prototype error rate (T = 27 °C)**

control voltage ($\sim$ 1 V), the relative variation of the oscillation frequency compared with the typical condition is about 12 % at 100°C, and about 5 % at 0°C. At low control voltage ($\sim$ 0.8 V), the variation is only 1 %.

To prove the temperature variation immunity property of proposed circuit, a simulation under 100 °C is performed. The same clocks in previous simulation are applied. Measurement results shown in Fig. 4.30 localize the minimum error in the interval of [-56.97 ps, -50.36 ps). Compared to the real minimum error value -53.4 ps, a precision of [-3.57 ps,

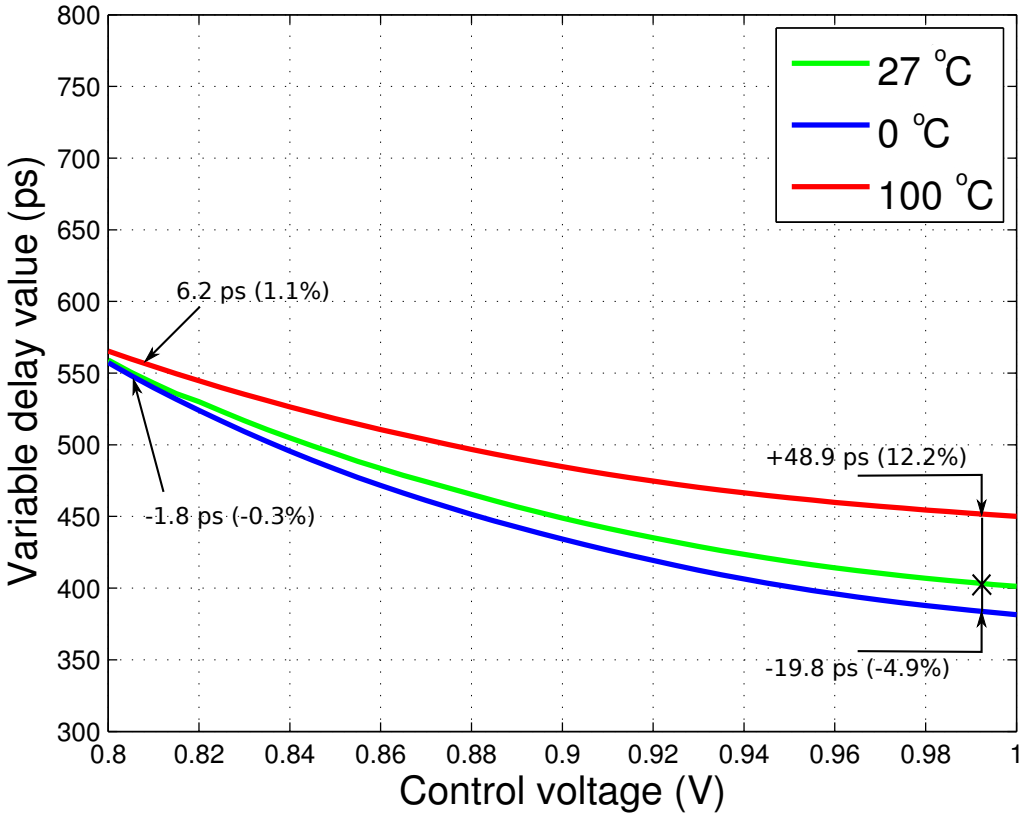


Figure 4.29: **Simulated variable delay value versus control voltage at different temperatures:** 27 °C, 0 °C, 100 °C

3.04 ps) is achieved. In the same way, the maximum error is in the interval (15.42 ps, 20.81 ps], which has a difference lying in the range (-4.18 ps, 1.21 ps] with respect to the real value 19.6 ps. The measured skew is -20.81 ps, which is 0.81 ps smaller than the real value 20 ps.

Process variation immunity

Fig. 4.31 shows the measured voltage-delay characteristics at different process corners. In the SS case, the variation is from 23.2 % to 32.3 % with respect to TT condition. While in FF case, the variation is between 18.3 % and 23.3 %. We can see that different from temperature variation, the circuit is more sensitive to process variation at low control voltage.

To evaluate the immunity level of process variation, two tests are performed in "SS" corner and in "FF" corner. Fig. 4.32 demonstrates measurement results in "SS" corner. A precision of [-2.98 ps, 4 ps) is achieved for minimum error detection, and the precision of maximum error measurement is (-2.95 ps, 2.95 ps]. The experimental value of skew is -19.23 ps. For the "FF" case, as shown in Fig. 4.33 the experimental minimum error has a difference of [-1.33 ps, 3.35 ps) compared to real value. The precision of maximum error measurement is (-3.92 ps, 1.94 ps]. The measured skew -21.09 ps is 1.09 ps less than 20 ps.

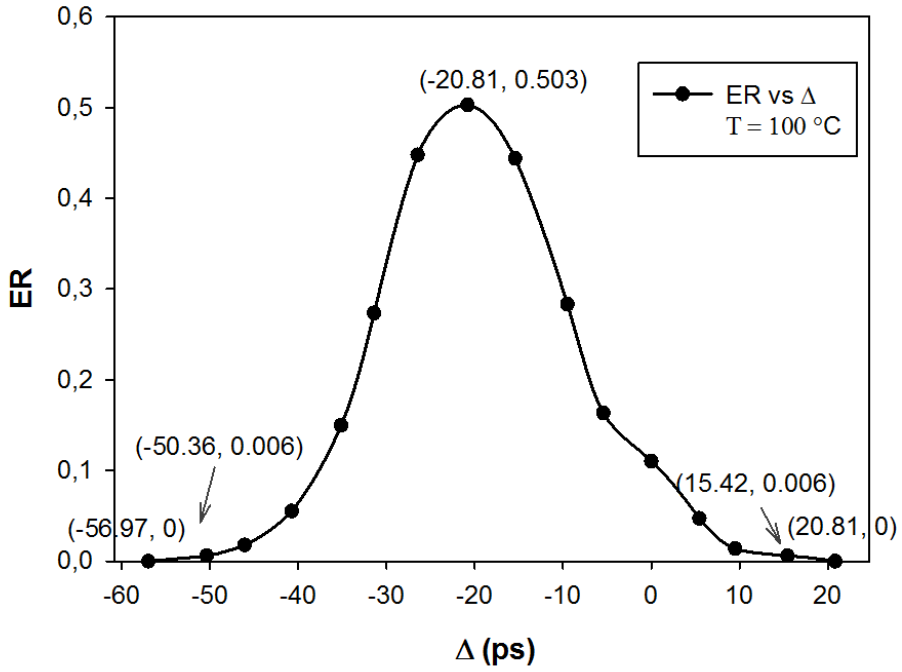


Figure 4.30: **ASIC prototype error rate measurement (T = 100** °C)

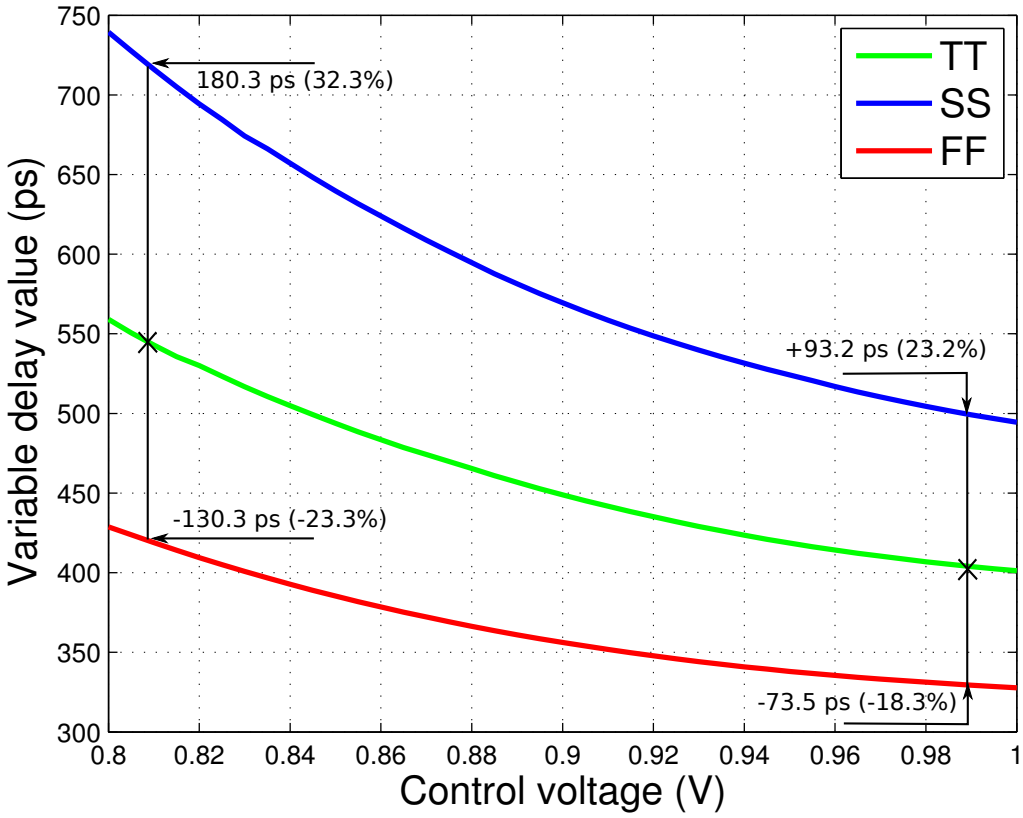


Figure 4.31: **Simulated variable delay value versus control voltage in different process corners:** process variations: TT, FF and SS

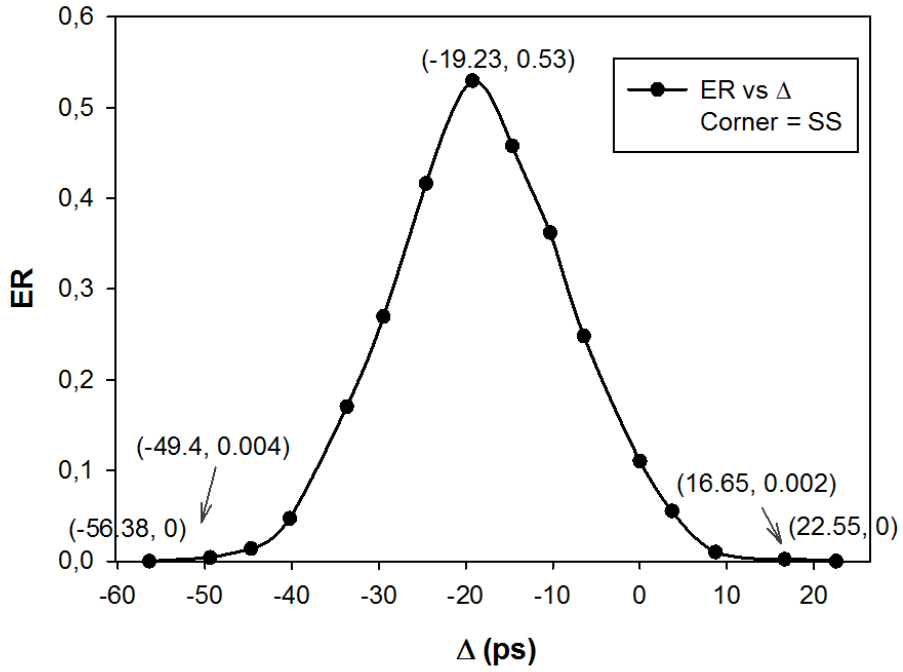


Figure 4.32: **ASIC prototype error rate measurement (Corner SS)**

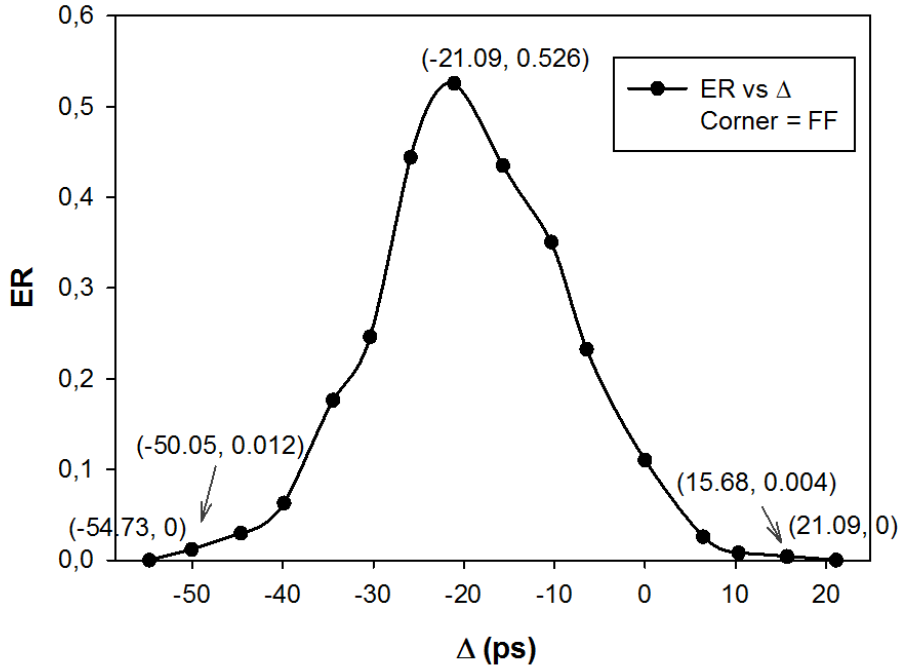


Figure 4.33: **ASIC prototype error rate measurement (Corner FF)**

Supply voltage variation immunity

Fig. 4.34 shows the measured voltage-delay characteristics with three different supply voltages (the standard 1.2 V and 1.2 V $\pm$ 5 %). At high control voltage ($\sim$ 1 V), the relative variation of the oscillation frequency compared with the typical condition is less than 7 %.

At low control voltage ($\sim$ 0.8 V), the variation is about 1 %. We observe that the circuit has a good immunity of power supply variation.

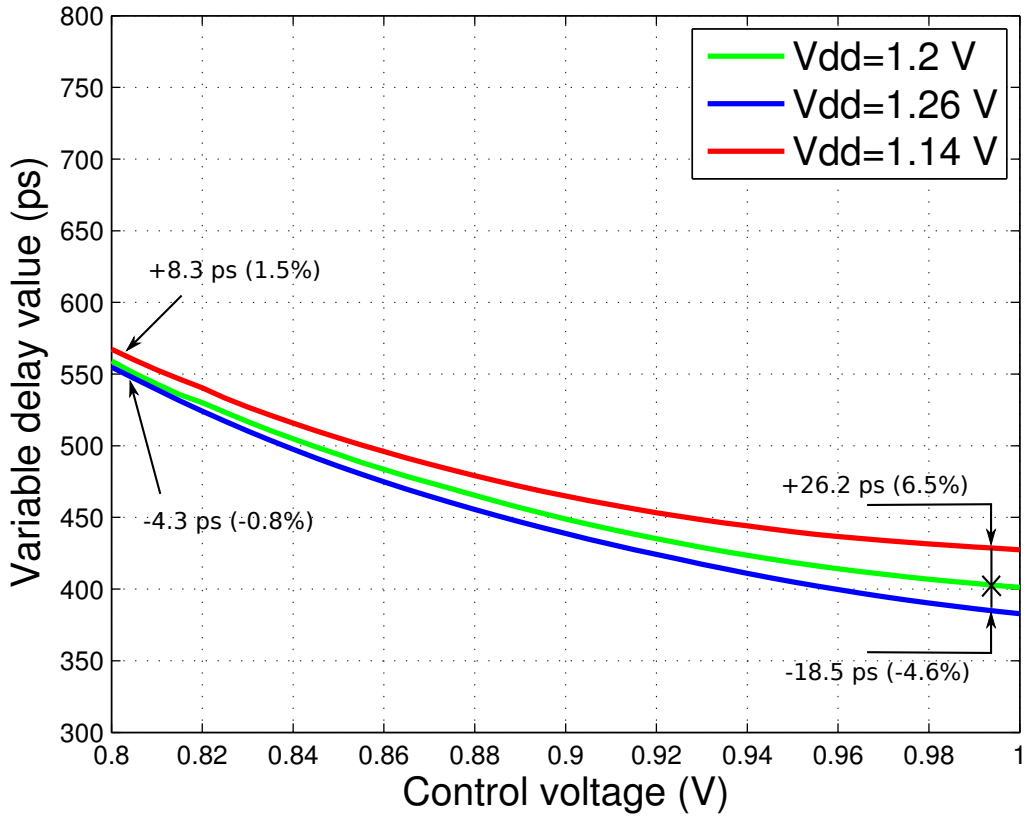


Figure 4.34: **Simulated variable delay value versus control voltage with power supply variation:** nominal and changed by $\pm$5%

To evaluate the immunity level of supply voltage variation, two tests are performed with a supply at 1.26 V (Vdd=1.2 V + 5%) and at 1.14 V (Vdd=1.2 V - 5%). Fig. 4.35 demonstrates measurement results with Vdd at 1.26 V. The test circuit achieves a precision of [-1.58 ps, 2.9 ps) for minimum error detection, and a precision of (-5.12 ps, 0.8 ps] for maximum error measurement. The experimental value of skew is -20.40 ps.

When the supply voltage is 1.14 V (Vdd=1.2 V - 5%), as shown in Fig. 4.36, the experimental minimum error has a difference of [-2.12 ps, 2.04 ps) compared to real value. The precision of maximum error measurement is (-3.93 ps, 0.02 ps]. The measured skew -19.62 ps is 0.38 ps less than 20 ps.

In conclusion, although the voltage controlled delay value varies at different PVT conditions, from Fig. 4.34, Fig. 4.29, and Fig. 4.31 we can see that the delay-voltage relation is always linear and monotonic. And a delay range of more than 100 ps is guaranteed. Moreover, since we use the differential value of two variable delays close to each other, the two delay cells have similar PVT conditions. The curves and data presented in this chapter are limited to certain working conditions. By measuring the delay value using integrated ring

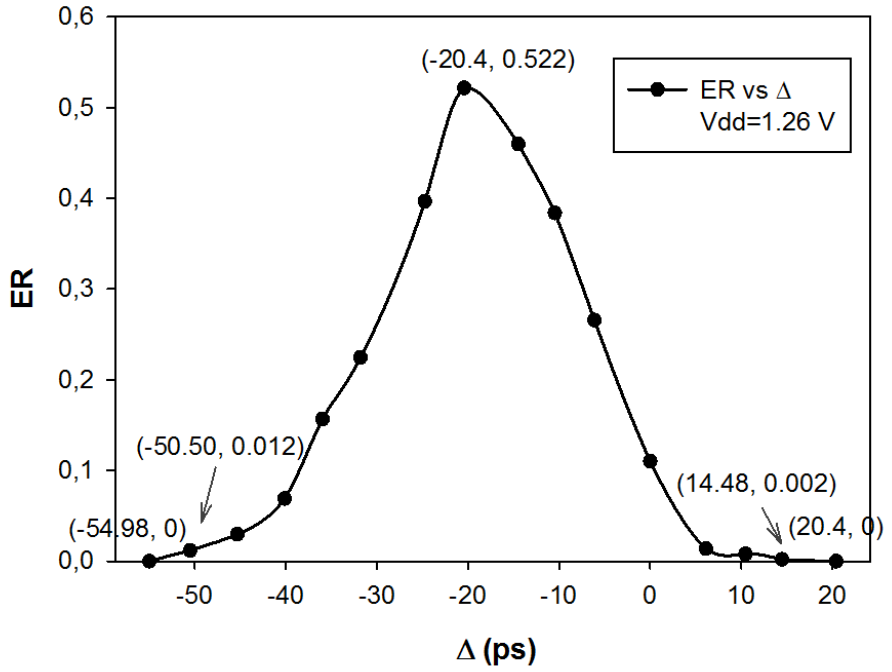


Figure 4.35: **ASIC prototype error rate measurement (Vdd=1.2 V + 5%)**

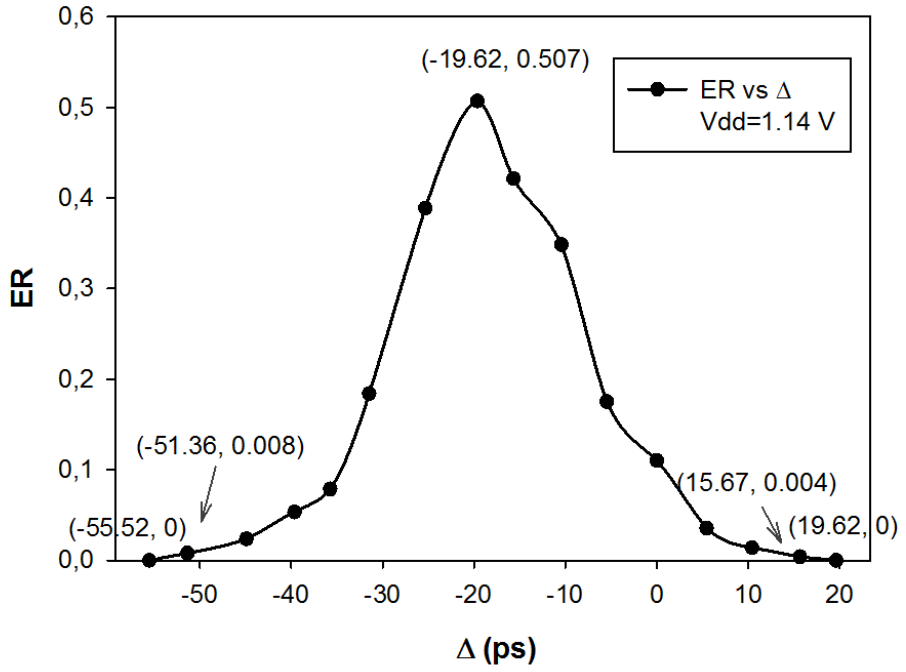


Figure 4.36: **ASIC prototype error rate measurement (Vdd=1.2 V - 5%)**

oscillator, we can know the delay value in current test environment.

4.6 Procedure of measurement

This designed test circuit can be injected in the chip of clocking network (cf. Fig. 4.37) and is used to measure the phase error between two clock signals in the chip. In this section, we present the user interface of the test method and the procedure of measurement.

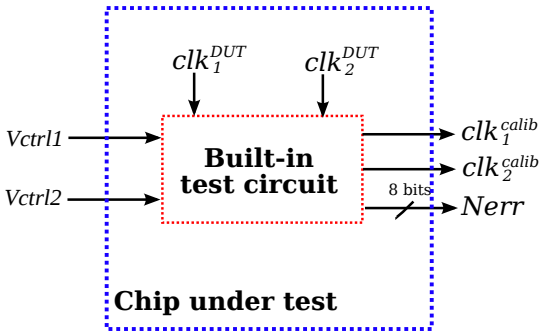


Figure 4.37: **The integration of proposed test circuit on the chip**

The proposed test circuit has four inputs and three outputs. Two of the inputs — the clock signals under test (clk_1^{DUT} and clk_2^{DUT} in Fig. 4.37) are internal signals of the chip. They can be the divided clock signals generated by DCOs or the input reference clock. One block of test circuit results in five additional pins of the chip: two input pins ($Vctrl1$ and $Vctrl2$) for the control and three additional output pins (clk_1^{calib}, clk_2^{calib} and $Nerr$) for the observation (cf. Fig. 4.37). A specification of the I/O pins are as follows:

- $Vctrl1$ and $Vctrl2$: Analog control signals. Their voltage values decide the values of variable delay elements inside the test circuit. These two inputs should have a range from 0.7 V to 1.1 V with a tuning step of 5 mV and a precision of 1 mV. They can be generated by an external power supply device such as Agilent 6625A, which allows a precision of less than 0.05%.

- clk_1^{calib} and clk_2^{calib}: Clock signals generated by calibration oscillators (oscillator1 and oscillator2 in Fig. 4.24). By observing the period value of these two periodical signals, we can get the values of the variable delay elements inside the block, and we can tune the corresponding control signals $Vctrl1$ and $Vctrl2$ until we get the desired delay values. The frequency of these two signals are in the range from 143 MHz to 333 MHz. They can be observed on oscilloscope such as LeCroy WaveRunner 625Zi.

- $Nerr$: 8-bit digital signal. Its value represents the number of errors during the last 255 clock cycles. The update rate of this output signal is around 1 MHz. The multi-bit value can be captured by Agilent 16902B Modular Logic Analysis System.

As presented in Section 4.5, the basic manipulation of measurement is as follows:

1. Modify voltages of the two control signals $Vctrl1$ and $Vctrl2$, and observe the frequencies of clk_1^{calib} and clk_2^{calib}. From the values of frequencies we get the values of variable delays (Δt_{d1} and Δt_{d2} in Fig. 4.16). Tune the control voltages until Δt_{d1} and Δt_{d2} give a desirable effective delay value Δ ($\Delta = \Delta t_{d1} - \Delta t_{d2}$).

2. Fix the voltages of $Vctrl1$ and $Vctrl2$ and run the test for a certain number of clock cycles (at least 256 cycles). Capture the value of $Nerr$ by logic analyzer.

3. Repeat the two previous steps for other values of Δ and note the corresponding $Nerr$ values.

4. Calculate the error rate ER for each Δ and trace the curve of ER in function of Δ. Find the points A, B and C in the curve as shown in Fig. 4.7(d). The Δ values at the three points correspond to the skew, min/max clock errors between the two clock signals under test.

4.7 Conclusion

A simple test circuit is proposed to evaluate the clock error statistics between two clock signals in chip. It provides an easy and straightforward way to measure static error (skew) and minimum/maximum dynamic error values. Not based on a direct time interval measurement, this on-chip test method reduces cost and difficulty of high frequency clock distribution quality test. The method was validated experimentally on a low-frequency discrete prototype, and a high-frequency integrated prototype was designed in 65nm CMOS technology and validated by simulation.

Chapter 5

Clock network FPGA prototyping

Contents

5.1 Introduction 107
5.2 Implementation of FPGA based blocks 110
5.3 Experimental results 116
5.4 Conclusion 128

5.1 Introduction

FPGA prototyping is a conventional and effective verification step in the digital ASIC design flow. In particular, it allows a validation of the functionality of the designed systems or some of its blocks and detects potential problems and errors before the ASIC fabrication against very small additional design efforts. In our case, the FPGA prototyping allows :

- validation of the programming interface;
- validation of the design of the error processing block ;
- validation of the technique of elimination of undesirable synchronization mode (modelocks);
- functional validation of the global operation of ADPLL network, particularly of network having large size, e.g. 10×10 or 12×12;
- estimation of the quality of synchronization between non-neighboring nodes.

The key limitations of the FPGA prototyping concern the impossibility to implement properly (a) the mixed signal blocks : the TDC and the DCO and (b) blocks operating at high frequency. The former problem is solved by implementing digital equivalent of the mixed signal blocks (behavioural models implemented in the hardware), as it will be shown in this chapter. To address the latter problem, it is enough to downscale all frequency parameters with the same scaling factor. Details of these techniques are presented in Section 5.2.

The FPGA prototyping of the ADPLL network was one of the tasks I animated in the research project, in the frame of HODISS and HERODOTOS grants of ANR. Two FPGA prototypes have been realized during this PhD project. The FPGA prototype with a dimension of 4×4 realized in 2011 is the first physical hardware implementation of ADPLL network, published in [52]. This study proved the feasibility of a fully synchronized clock distribution for 16 clock domains *before the fabrication of a chip with the same architecture in 65 nm CMOS*. Moreover, it allowed studying complex phenomena related with coupled ADPLL operation. Before the tapeout of the 10×10 version of silicon chip, an equivalent FPGA prototype with the same size is realized to validate this design and to explore the synchronization performance in a large network. Since the two FPGA prototypes share the same principle, we present them together in this chapter. In Section 5.2, we present the implementation of the system and its block. Downscaling of parameters are also explained in this section. While in Section 5.3, we first present the experimental results of the first prototype, which demonstrate the stability of system and the effectiveness of dynamic reconfigurability in undesired steady state prevention.

The test of the FPGA prototypes of a large ADPLL network responded to a fundamental question about the scalability of this clocking solution. In particular, by testing a 10×10 FPGA prototype, we have proven that the phase error between non-neighboring nodes increases according a "slow" law and exhibits a saturation. It means that the proposed approach is suitable for synchronisation of networks containing a very large number of local clock sources.

Since the two prototypes share the same structure, we only present the architecture of the second prototype in Fig. 5.1. The implemented clock network has 100 nodes. It is configured as 10×10 Cartesian 2 dimensional mesh. Each node is composed of the ADPLL blocks whose design is addressed in preceding Chapter 3. The reference clock is injected at a corner node. The implemented network topology is chosen so to compare the designed clocking system with the unique implementation of PLL network based clock generator reported in literature [15].

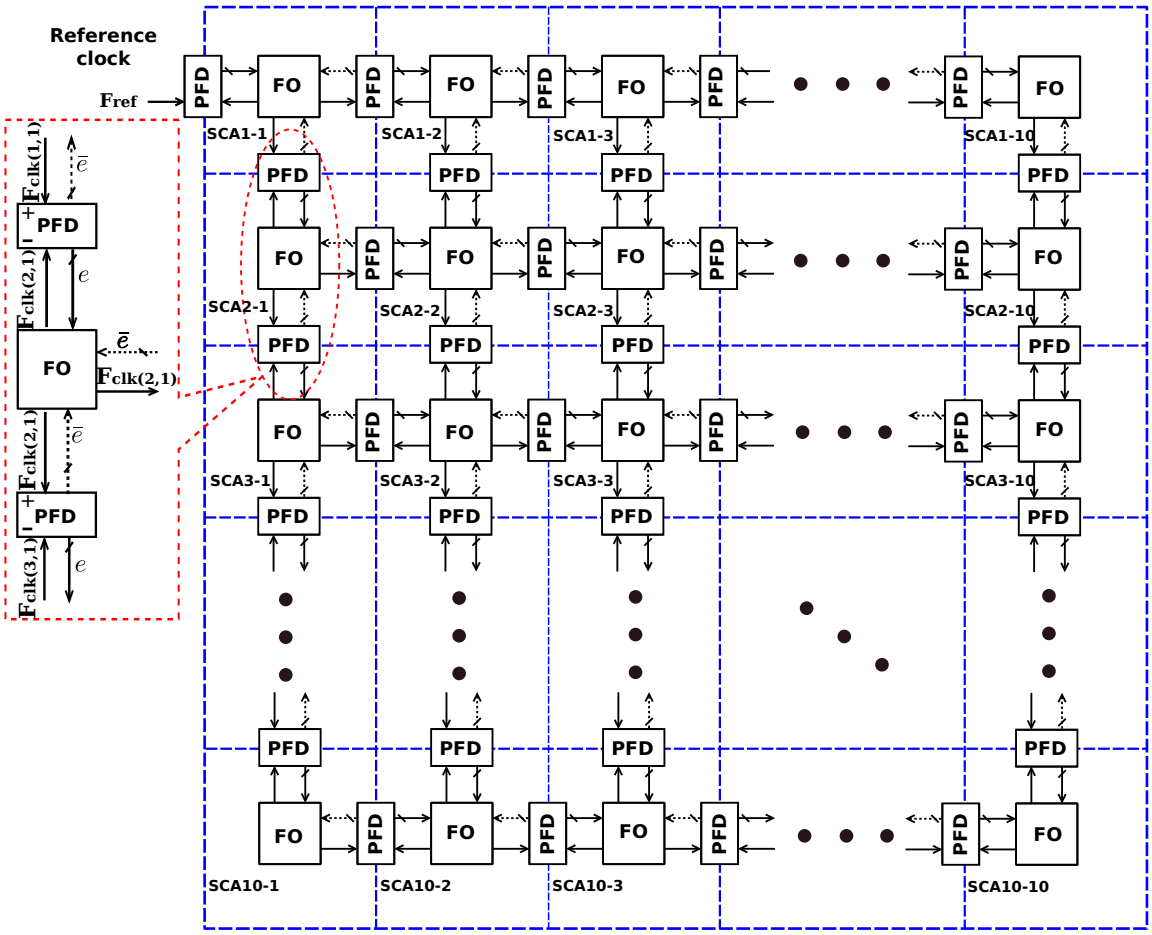


Figure 5.1: **Structure of the implemented clock network**

5.2 Implementation of FPGA based blocks

Here we summarize the principal issues limiting the efficiency of the FPGA prototyping for validation of an ASIC:

- **Difficulty of implementation of analog/mixed features**. The key limitation of the FPGA prototyping flow is an impossibility of implementing a pure continuous-time delay. In particular, the VHDL code with time definition

  ```
  A <= B after delay
  ```

  which are used for behavioral modeling of TDC is not synthesizable. Hence, blocks using continuous-time delays of the gates in ASIC cannot be implemented through a standard FPGA design flow, for instance, the behavioral VHDL model of DCO using expression like

  ```
  clk <= not clk after period
  ```

  where *period* is a time-type variable. Moreover, the designed DCO needs non-standard CMOS tri-state inverters (cf. Subsection 3.3.3) which are not available in FPGA. As a result, the FPGA implementation of the ADPLL requires a different architecture for the TDC and DCO, and the silicon design of these two blocks cannot be verified through FPGA prototyping.
- **Frequency**. The typical clock frequency of the commercially available FPGA chips is in a range of hundreds of MHz: that is obviously insufficient for the prototyping of an oscillator generating gigahertz frequency signal. By consequence, a downscaling of the frequencies is needed for the FPGA prototype.

The frequency downscaling of the FPGA prototype of the ADPLL network follows the following principle: all timing parameters of the system are scaled linearly with the same scaling factor α:

$$\begin{aligned} f^{fpga}/f^{asic} &= \alpha, \\ t^{fpga}/t^{asic} &= 1/\alpha. \end{aligned} \tag{5.1}$$

Here f and t denote the frequencies and the time parameters of the FPGA and ASIC systems.

The next two subsections present the design of the DCO, of the TDC and the procedure of choice of optimal frequency scaling.

5.2.1 Synthesizable DCO

The aforementioned structure of DCO including an array of tri-state inverters or a matrix of variable capacitors is not implementable in the FPGA platforms.

However, the well-known direct digital frequency synthesis (DDFS) technique can be used to synthesize a fully digital DCO, which, in the context of ADPLL, behaves similarly with a mixed-signal DCO whose design is presented in Section 3.3.

The DDFS consists of two steps. The first step is the synthesis of the digital phase for the oscillator. A digital phase is a digital sequence $\{s_i\}_{i\in\mathbb{N}}$ defined in the discrete time given by the external clock with period T_{clk}^{fpga}. The sequence $\{s_i\}$ provides a saw-tooth digital signal with period equal to the period of the signal to be synthesized (Fig. 5.2).

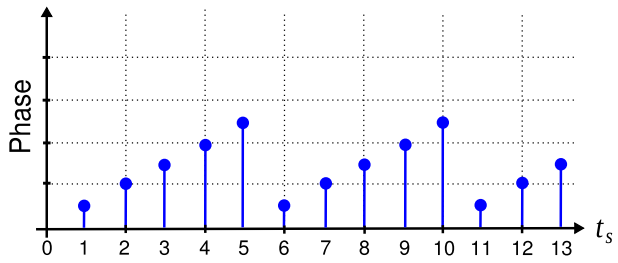


Figure 5.2: **Repeating discrete ramp function in the DDFS**

The second step is the use of the sequence values to generate a function $f(s_i)$ defining the waveform of the periodic signal. The output signal is digital, and can be converted to an analog representation with an DAC if required. In a contrast with the mixed-signal DCO, the output signal of the DDFS oscillator is synchronous with the clock defining the digital phase sequence, and the period of the obtained DCO signal T_{DCO}^{fpga} is necessarily a multiple of the period of this clock $T_{clk\,DCO}^{fpga}$. If $T_{DCO}^{asic} << T_{DCO}^{fpga}$, the DDFS DCO is a fair model of a proper mixed-signal DCO.

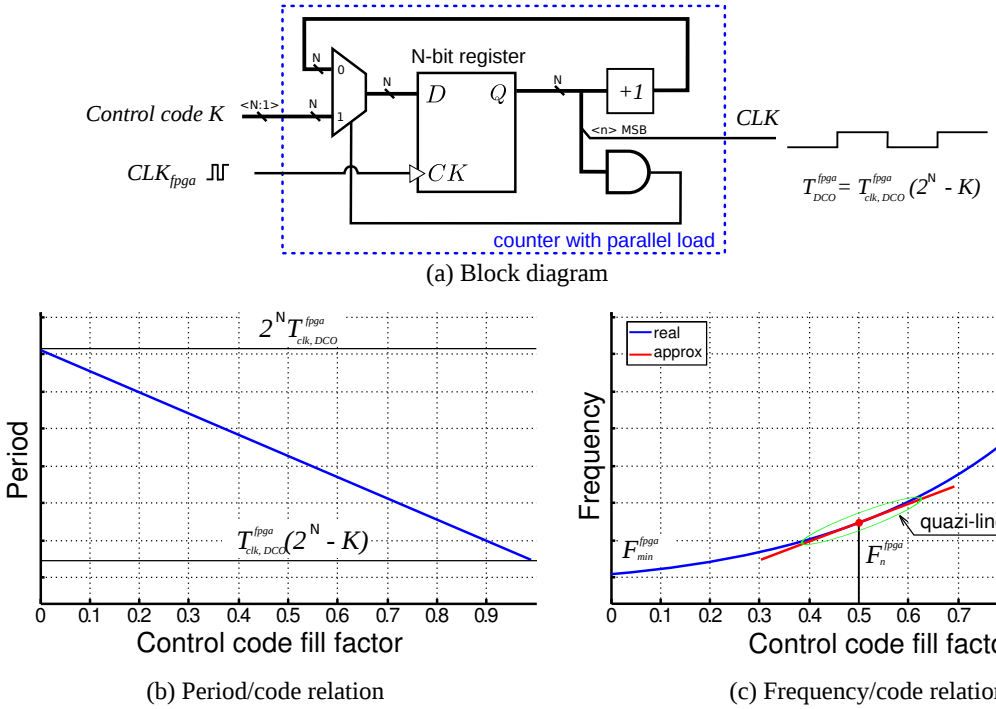
The phase synthesis is achieved with a programmable counter/divider receiving the FPGA clock signal and generating an increasing digital phase sequence at its output. When the counter output reaches the maximal value, the count starts from zero, so generating the phase waveform as in Fig. 5.2. The ADPLL DCO may use the overload output of the counter which marks the end of the current period and the beginning of a new period. In such a context, the counter is used as a programmable frequency divider whose division coefficient is equal to the desired period of the output sequence (measured in number of $T_{clk\,DCO}^{fpga}$). This period (the duration of one teeth of the saw) can be modulated by loading the initial output value K of the counter at the beginning of the new cycle (Fig. 5.3(a)). In this way, the period of the output sequence is given by:

$$T_{DCO}^{fpga} = T_{clk\,DCO}^{fpga}(2^N - K) \tag{5.2}$$

where N is the number of bits of the counter.

The corresponding period-code and frequency-code DCO characteristics are given in Fig. 5.3(b,c). In contrast to ASIC DCO, the code-frequency characteristic of FPGA DCO is nonlinear. However, it is not critical for the ADPLL prototyping, since the target ADPLL operation mode is when the output frequency is settled, and the input DCO code varies in a small range, so to correct the residual phase error. Consequently, the fluctuations of the frequencies in a network will be small and the DCO characteristic can be considered as locally linear (cf. Fig. 5.3(c)).

However, because of nonlinearity of the FPGA DCO frequency-code characteristic, it is not possible to ensure a linear scaling (Eq. (5.1)) at all frequencies. Hence, the downscaling is defined for a particular frequency considered as nominal. Arbitrarily, we define that the the nominal divided frequency of the ASIC DCO F_n^{asic} is in the middle of the tuning range, (218 MHz which corresponds to the code 512 for prototype-1; 265 MHz which corresponds to the code 128 for prototype-2).



(a) Block diagram

(b) Period/code relation

(c) Frequency/code relation

Figure 5.3: **Schematic diagram of the proposed FPGA implementation of the oscillator**: (a) schematic, (b) period/code and (c) frequency/code characteristics

The FPGA-prototyped ADPLL must emulate the ASIC system, however, it is clear that the FPGA based clock generator cannot output signals at the same frequencies as the original system. The maximal internal FPGA clock frequency is a hundred of megahertz, and, as stated, the synthesized frequency must be much lower than the DDFS DCO clock frequency. Hence, a frequency downscaling is necessary. The following calculation allows an identification of the necessary downscaling of the DCO frequency so to provide a representative model of the ASIC DCO.

Given that the tuning step of clock period of the FPGA DCO is equal to the period of the external clock $T_{clk\,DCO}^{fpga}$, the frequency step of the FPGA DCO ΔF_n^{fpga} is related to the FGPA DCO nominal output frequency F_n^{fpga} by the following relation :

$$\Delta F_n^{fpga} = F_n^{fpga} - \frac{1}{\frac{1}{F_n^{fpga}} + T_{clk\,DCO}^{fpga}}. \tag{5.3}$$

At the same time, we want the relation between the nominal frequency and the frequency gain to be the same in the FPGA and in the ASIC DCO (cf. Eq. (5.1)). Hence, the second

relation between F_n^{fpga} and ΔF_n^{fpga} is:

$$\frac{F_n^{fpga}}{\Delta F_n^{fpga}} = \frac{F_n^{asic}}{\Delta F_n^{asic}} \tag{5.4}$$

These two equations has one free parameter $T_{clk\,DCO}^{fpga}$. It cannot be superior to the minimal period of the FPGA internal clock, whose typical frequency is a hundred of MHz, depending on the used FPGA platform. Its actual value depends on the constraint about the scaling of the TDC, which is discussed in the next subsection.

We want the FPGA based DCO to have the same number of steps as the ASIC DCO. Hence, the maximal and minimal frequencies of the FPGA DCO are given by

$$\begin{array}{rcll} F_{min}^{fpga} & = & \frac{1}{1/F_n^{fpga}+127T_{DCO}^{fpga}} & \text{prototype}-1 \\ & = & \frac{1}{1/F_n^{fpga}+511T_{DCO}^{fpga}} & \text{prototype}-2 \end{array} \tag{5.5}$$

and

$$\begin{array}{rcll} F_{max}^{fpga} & = & \frac{1}{1/F_n^{fpga}-128T_{DCO}^{fpga}} & \text{prototype}-1 \\ & = & \frac{1}{1/F_n^{fpga}-512T_{DCO}^{fpga}} & \text{prototype}-2. \end{array} \tag{5.6}$$

It should be noticed that since the frequency-code characteristic of the FPGA based DCO is nonlinear, the scaling (5.1) is not valid for F_{max}^{fpga} and F_{min}^{fpga}.

5.2.2 Synthesizable TDC

The proposed TDC for FPGA prototyping is a digital chronometer counting the number of external high frequency clock cycles during the interval to be measured. The interval length is specified by the MODE pulse duration (Fig. 5.4). The pulse is applied to the counter enable input *EN*. The counter output increments till the end of the pulse. The XI1 register stores the result synchronously with the falling edge of the input pulse. At a small delay time, the *RESET* input of the counter receives an active level, so preparing the counter to a new measurement cycle.

Comparing the counter-based TDC with a delay line based TDC, we can notice that they both quantize the input interval duration. However, in the counter-based TDC, the start of the input pulse isn't synchronized with the clock of the counter. It means that in a counter-based TDC, the first quantization step can be less than one period of the TDC clock. On the contrary, in the delay-based TDC the first quantization step is always equal to the delay of the first delay element, since its operation is always synchronous with the input pulse. This results in a different behavior for small errors inferior to the quantization step, in which case the counter-based TDC may sometimes identify as '1', whereas the delay line based TDC always outputs '0'.

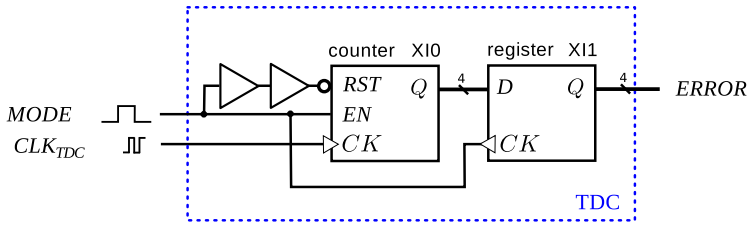


Figure 5.4: **Conventional phase detector**: (a) circuit diagram, (b) state diagram (c) waveforms and (d) transfer function

It is obvious that such a TDC cannot measure synchronization errors of gigahertz frequency signals, since the frequency of the TDC clock should be much higher that the frequency of the signals to be measured. Hence, again, a frequency downscaling is necessary. The only time parameter of the TDC is the quantization step τ_{TDC}^{fpga}, which must be related to the ASIC TDC quantization step τ_{TDC}^{asic} by the same scaling factor α as the DCO time/frequency parameters of both ASIC and FPGA systems:

$$\tau_{TDC}^{fpga}/\tau_{TDC}^{asic} = F_n^{asic}/F_n^{fpga}. \tag{5.7}$$

The free parameter of the Equation (5.7) is the clock period of the FPGA based TDC τ_{TDC}^{fpga} (or $T_{clk_{TDC}}^{fpga}$). From the Eq. (5.7), Eq. (5.3) and Eq. (5.4) and from the parameters of the blocks designed for ASIC, the ratio between the clock frequencies of the DCO and TDC was calculated:

$$\begin{aligned} F_{clk_{DCO}}^{fpga}/F_{clk_{TDC}}^{fpga} &= 8.915 \quad (\text{prototype}-1) \\ &= 4.25 \quad (\text{prototype}-2) \end{aligned} \tag{5.8}$$

where $F_{clk_{TDC}}^{fpga} = 1/\tau_{TDC}^{fpga}$.

Hence, the $F_{clk_{DCO}}^{fpga}$ should be set at the maximal clock frequency available in the used FPGA platform, the other frequency parameters are calculated through the Eq. (5.7), Eq. (5.3), Eq. (5.4) and Eq. (5.8). Tab. 5.1 and Tab. 5.2 summarize the parameters of the TDC and of the DCO implemented for the first and second FPGA prototypes of the ADPLL network.

The complete description of the synthesizable VHDL code of the proposed DCO and TDC can be found in Appendix A.

Table 5.1: **Parameters of the 1^{st} generation FPGA and ASIC implementations**

Parameter	ASIC	FPGA
F_n	218 MHz*	48.904 kHz
ΔF_n	185.25 kHz*	38.236 Hz
F_{min}	125 MHz*	34.916 kHz
F_{max}	310 MHz*	81.486 kHz
τ_{TDC}	32 ps	143 ns
$F^{fpga}_{clk_{DCO}}$	–	62.5 MHz
$F^{fpga}_{clk_{TDC}}$	–	7.01 MHz

* for DIV clock

Table 5.2: **Parameters of the 2^{nd} generation FPGA and ASIC implementations**

Parameter	ASIC	FPGA
F_n	265.516 MHz*	77.93 kHz
ΔF_n	330 kHz*	97.05 Hz
F_{min}	232.2 MHz*	67.28 kHz
F_{max}	295.9 MHz*	92.73 kHz
τ_{TDC}	20 ps	68.065 ns
$F^{fpga}_{clk_{DCO}}$	–	62.5 MHz
$F^{fpga}_{clk_{TDC}}$	–	14.7 MHz

* for DIV clock

5.3 Experimental results

5.3.1 Stability and prevention of mode-lock

Two FPGA prototypes of the network whose architecture are given in Fig. 5.1 were implemented. The TDC and DCO blocks were designed as described in the last section. The digital processing block and the programming interface were synthesized from the same code as that used for the ASIC design.

The system was implemented on the Altera evaluation test board with Cyclone II EP2 C70F672C6 chip. The basic information about this board, as well as the information about measurement set can be found in Appendix D. The synthesis and implementation were performed in Altera Quartus II environment. The FPGA was programmed, and controlled on the fly by Altera USB blaster cable. The reference signal for the input of the clock network was synthesized by external generator with high precision and temporal stability.

The behavior of the system was observed with help of a digital oscilloscope at the points indicated in Fig. 5.5. The signals at each output of the PFD and DCO were observed. In such a way, we have the information about the phases of the local clock signals and by consequence, we know the phase error between them.

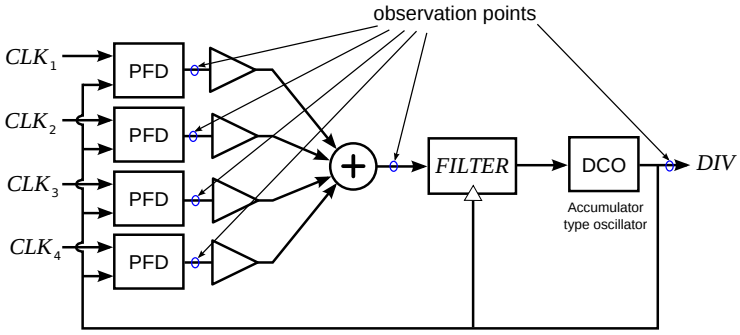


Figure 5.5: **Block diagram of the node in a FPGA prototype with observation points**

On the first 4×4 network prototype, we have performed two experiments to demonstrate the system stability and the effect of dynamic configuration on mode-lock prevention.

The first experiment was done with an ideal network, in which all DCOs have the same initial frequency (corresponding to 512 DCO code) and the same initial phase. In this configuration, the network converged to a synchronized state. The input (reference) clock frequency is set to 51.13 kHz and corresponds to DCO control integer code 566. The network has bidirectional configuration and with the following values of the signal processing block: gains K_{w_1,w_2,w_3,w_4}=1, K_p=1 and $K_i = 0.0028$. In Fig. 5.6 we can see that with these conditions, after the transitional process, all clocks have the same frequency and phase as reference clock.

However, this behavior may be non-representative of the real behavior of an ASIC net-

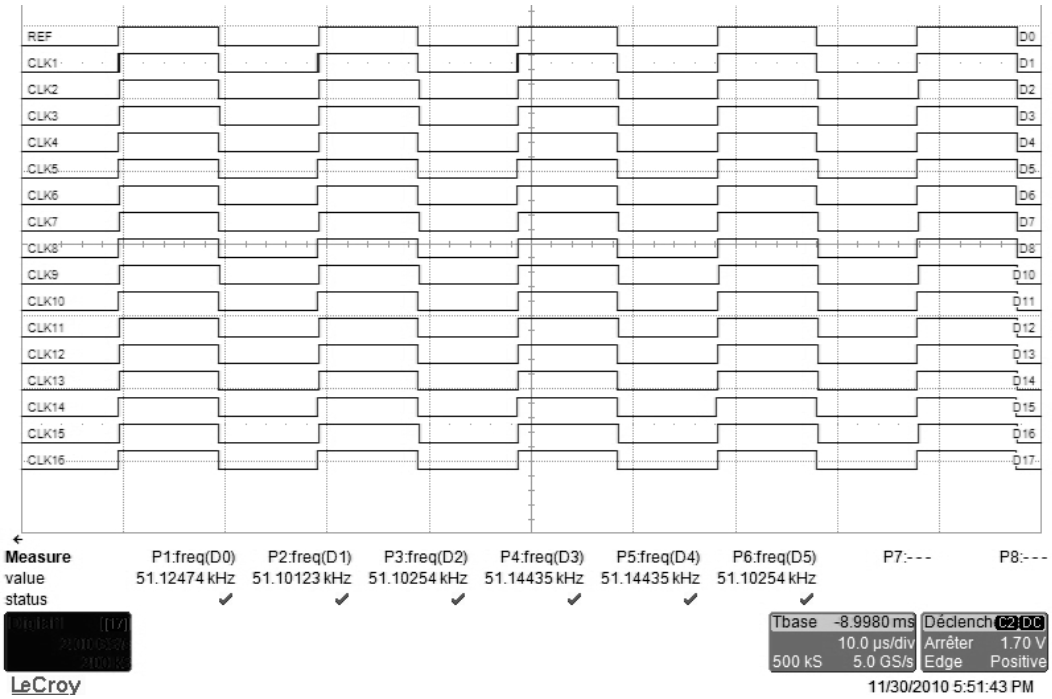


Figure 5.6: **Local clock signals together with reference (Prototype-1)**: bidirectional configuration, initial frequencies of nodes are equal

work. Indeed, in VLSI implementation, due to the local variations and local drop of the supply voltage, the local frequencies of the nodes differ and the local oscillators do not start with the same phase. In particular, in the idealized configuration used in the experiment, undesirable synchronized modes (i.e. mode-locks) were not observed.

In the next experiment we randomized the initial conditions of the network, by diversifying the initial phases of the local oscillators. The initial frequencies of the DCO are set to the values distributed around the nominal value with a dispersion ± 20 %. Fig. 5.7 presents the outputs of the local oscillators after the transient process. A static phase error between clock signals is observed. Fig. 5.8 demonstrates the waveforms of the clock signals from the neighbors of the Node 10 and total error signal in Node 10 processed by its filter: whereas the phase errors are non-zero, the total error is zero, and the frequencies of all local oscillators are the same. This state is a typical for a mode-lock.

The next experiment aims at a verification of the proposed mode-lock elimination technique, which is based on a dynamic reconfiguration of the clock network. The set-up of the network is achieved in two stages. In first stage, the network operates in unidirectional mode. The phase/frequency information from the reference clock is propagated to the opposite corner of the network, without any feedback. The value of the filter coefficients remains the same as in previous experiments, the value of the control block gains are K_{w_1,w_4}=0 and K_{w_2,w_3}=1. Fig. 5.9 presents the operation in unidirectinoal mode after a transient process. There exist static accumulative phase errors. The phase errors increase from the corner node directly coupled with the reference signal, toward the opposite corner where error reaches

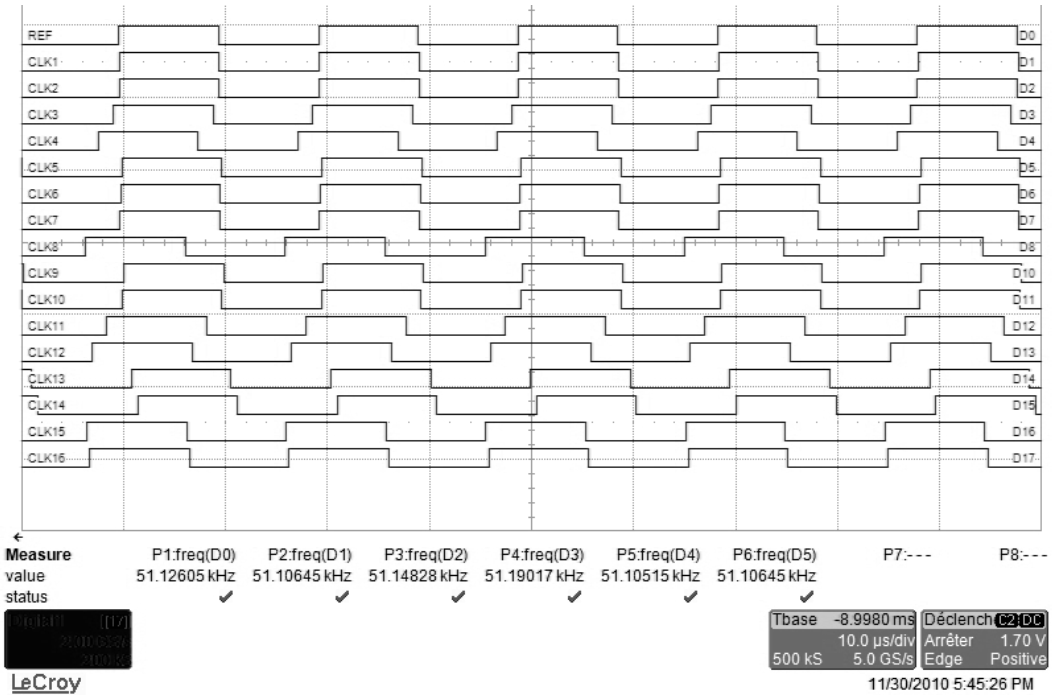


Figure 5.7: **Local clock signals together with reference (Prototype-1)**: bidirectional configuration, initial frequencies of nodes are different

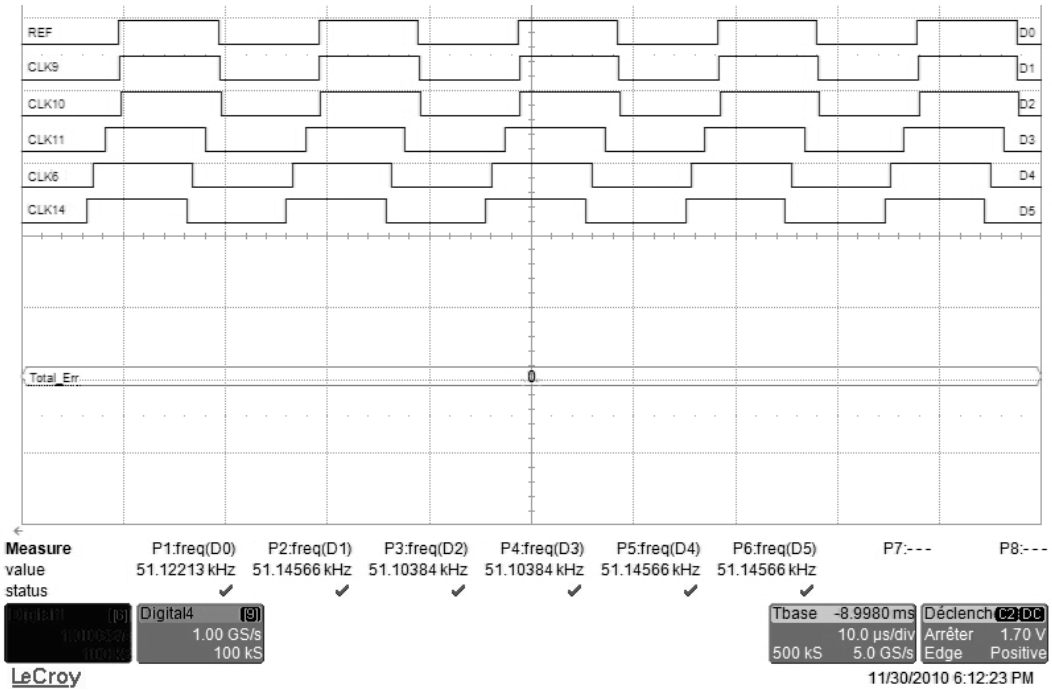


Figure 5.8: **Local clock signals around Node 10 together with reference and integer sum of the node errors (Total_Err), Prototype-1**: bidirectional configuration, initial frequencies of nodes are different, undesired synchronized state

the highest value. However, these errors are smaller than in a mode-lock state as one can observe by comparing Fig. 5.10 showing the neighbor clocks of the Node 11, and Fig. 5.8.

In the second stage, the network is configured to operate in the bidirectional mode. The

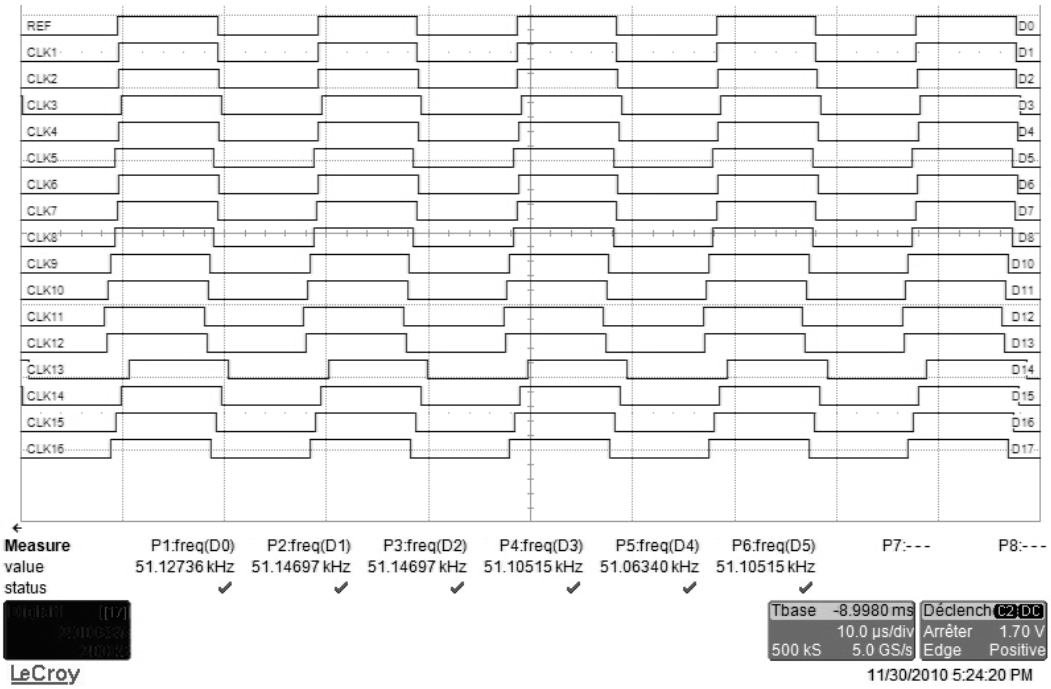


Figure 5.9: **Local clock signals together with reference (Prototype-1)**: unidirectional configuration, initial frequencies of nodes are different

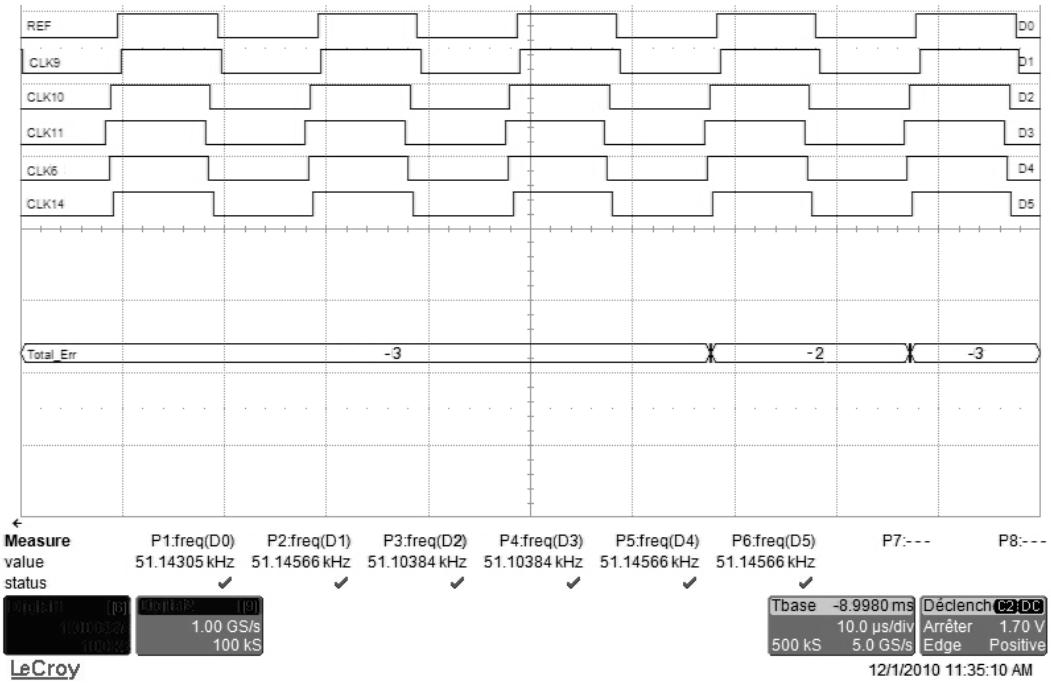


Figure 5.10: **Local clock signals around Node 10 together with reference and integer sum of the node errors (Total_Err), Prototype-1**: unidirectional configuration, initial frequencies of nodes are different, static errors exist

bidirectional mode is switched on once the unidirectionally configured network set up in a steady state mode. The mode switching is done by setting to 1 the gains K_{w_1}, K_{w_4}. Fig. 5.11 shows the steady state operation in the second stage. These plots indicate that the static residual phase errors appeared in a previous unidirectional mode are compensated and min-

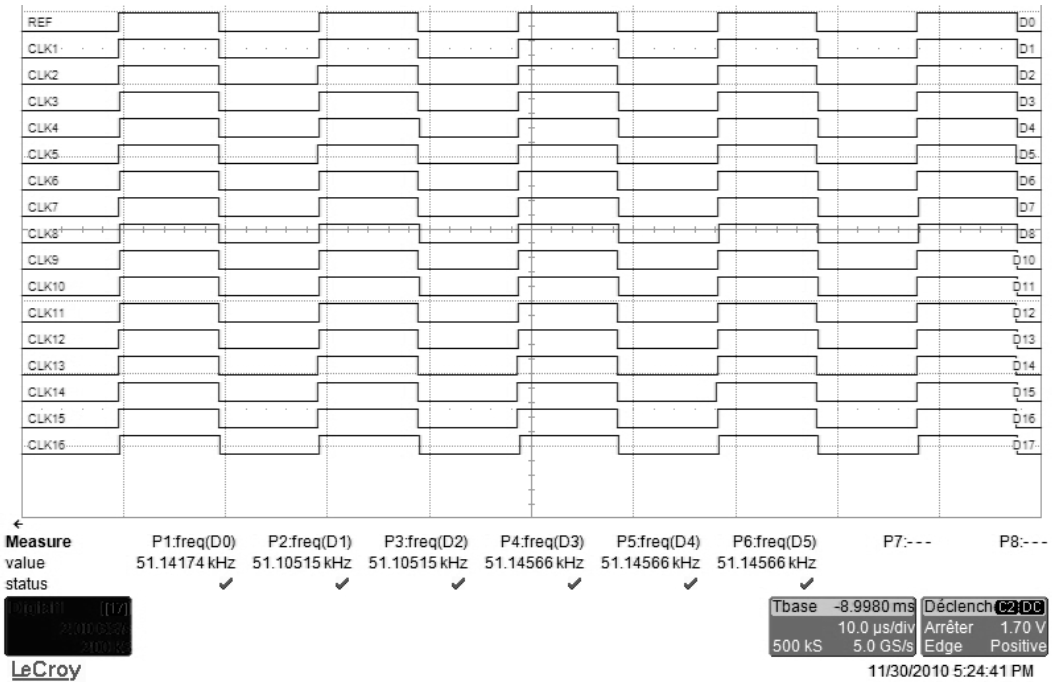


Figure 5.11: **Local clock signals together with reference (Prototype-1)**: bidirectional configuration, initial frequencies of nodes are different

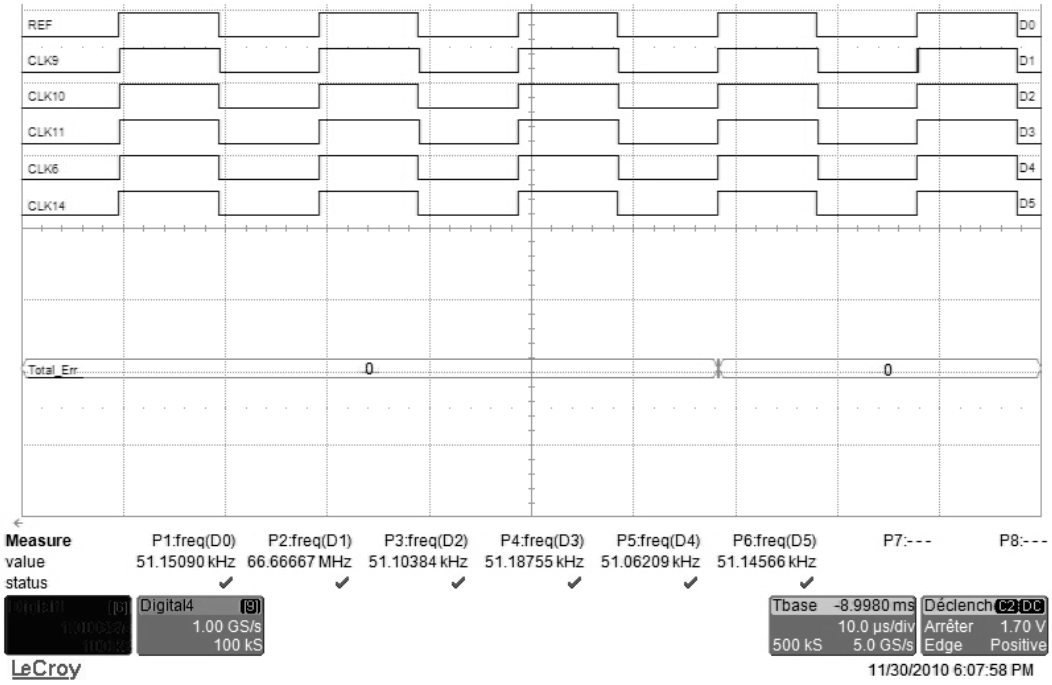


Figure 5.12: **Local clock signals around Node 10 together with reference and integer sum of the node errors (Total_Err), Prototype-1**: bidirectional configuration, initial frequencies of nodes are different, synchronized state with zero total error

imized. The plots of Fig. 5.12 demonstrate that they are small. Observation of the PFD outputs demonstrates that the errors between neighboring oscillators do not exceed 2 phase errors quantization steps.

The same experiments are also performed on the second prototype with a 10×10 di-

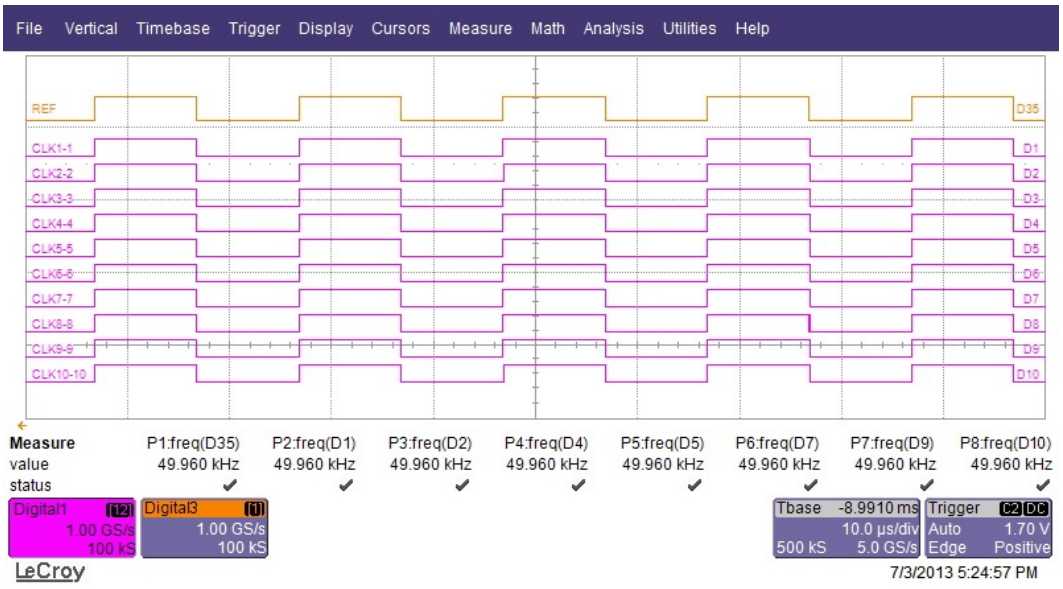


Figure 5.13: **Local clock signals together with reference (Prototype-2)**: bidirectional configuration, initial frequencies of nodes are different

mension. Because of the limit of pin numbers, we cannot observe all the 100 local clock signals at the same time. Fig. 5.13 shows the 10 clock signals in the diagonal nodes of the network. In this experiment, nodes in the network have divers initial frequencies, but when the network synchronizes, all the local clock signals share the same frequency and are well aligned in phase, which demonstrates a good performance of the proposed architecture in a large network.

In large networks, the length of unidirectional chains in the unidirectional mode may be large, so that the accumulated phase errors aren't small enough to prepare the network to the synchronization in a bidirectional mode. To study this problem, we present two topology of unidirectional networks in Fig. 5.14. To compare them, we introduce a parameter D standing for the phase error propagation distance between two clock domains, which is equal to the number of clock domain borders that the information passes through from one node to the other. Criteria of choosing the best unidirectional configuration is that the parameter D from the reference clock to each node in the network should be as small as possible. The topology in Fig. 5.14(a) uses a zigzag chain to connect all the nodes together. In this way, the value of D increases linearly as the geometry gets larger. In a 4×4 network, the distance between the last node at the end of the chain and the reference is 16, while in a 10×10 network, the value is 100. Fig. 5.14(b) shows a network with a comb topology. In this case, the distance D between each node and the reference is the Manhattan distance. The Manhattan distance between a node X=(X1, X2) and a node Y=(Y1, Y2) is defined as: $|x_1 - x_2| + |y_1 - y_2|$, which is the shortest distance between two intersections in a grid. In a 4×4 network, the longest distance is 7. For a 10×10 network using this configuration, the longest distance is 19. Both configuration topologies have been implemented in FPGA and tested.

Fig. 5.15 demonstrates the clock signals of the main network diagonal nodes, when the

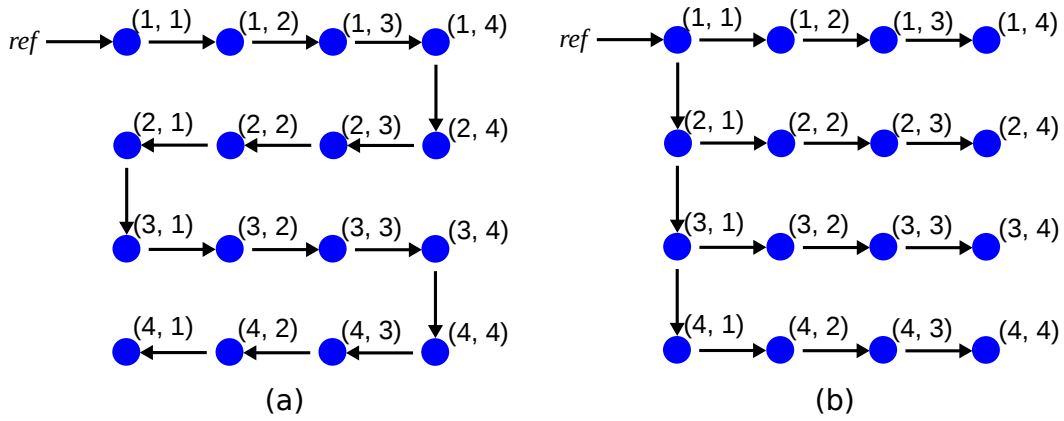


Figure 5.14: **Unidirectional mode topology**: (a) type 1 - zigzag; (b) type 2 - comb (the arrows show the connection in the network and the error propagation direction)

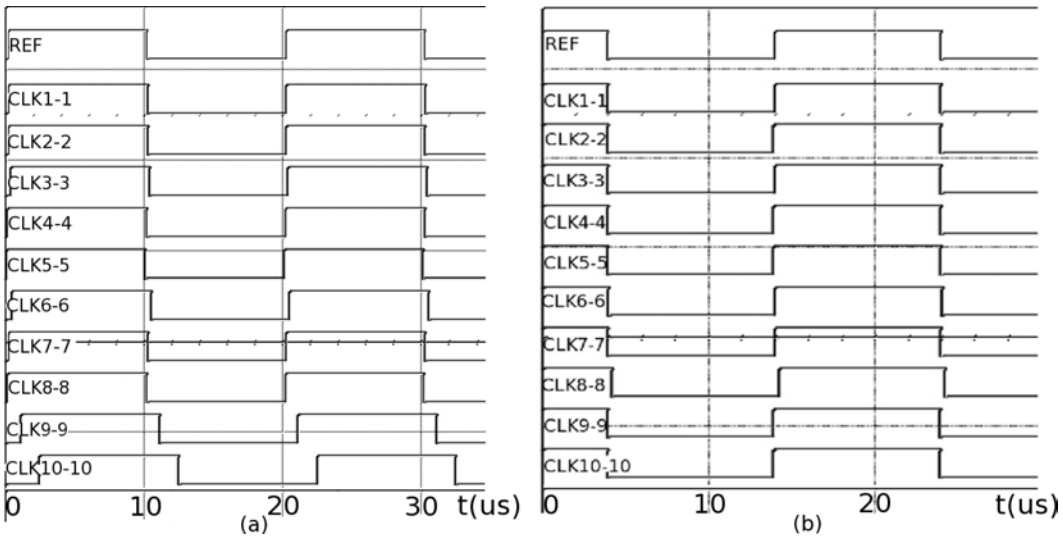


Figure 5.15: **Clocks in unidirectional mode at steady state**: (a) type 1 - zigzag; (b) type 2 -comb

network is configured in unidirectional mode. For the reasons explained above, the phase error between CLK10-10 (clock generated at SCA10-10 in Fig. 5.1) and the reference clock REF is smaller with comb shaped configuration compared with the zigzag type. For this reason, the comb shaped topology is chosen for unidirectional mode as the first phase of the dynamic configuration process.

5.3.2 Phase error between two remote local clocks

In order to validate that in such a coupled ADPLL network the phase errors are not accumulated as in a conventional clock tree, we measured on each clock cycle the phase difference between local clock signals in the diagonal nodes and the reference clock (as shown in Fig. 5.16). Then we can draw the curve of phase error maximum and RMS values in function of the distance to the reference clock. It should be noted that the *distance* mentioned here is not the physical distance in microns, but the Manhattan distance D the reference phase information has to travel before arriving at a local node. For instance, the shortest distance

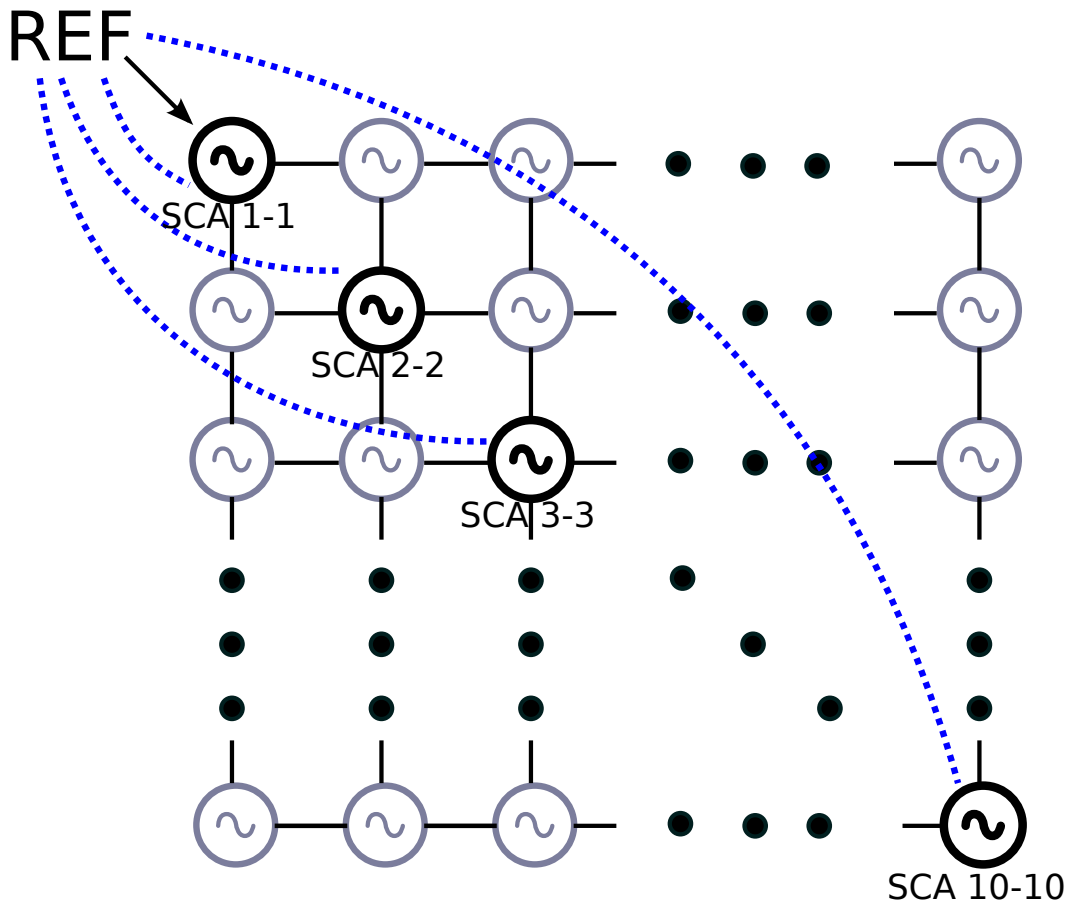
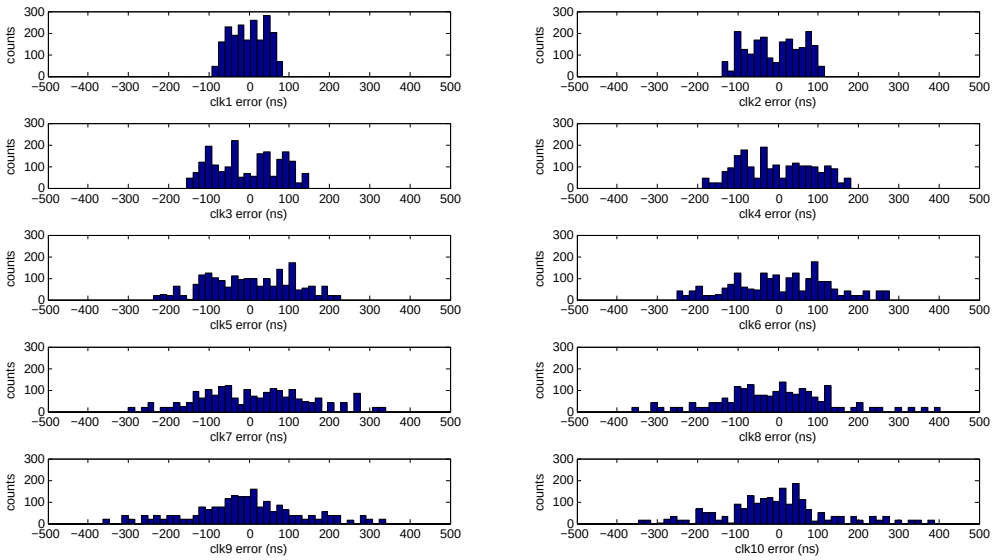


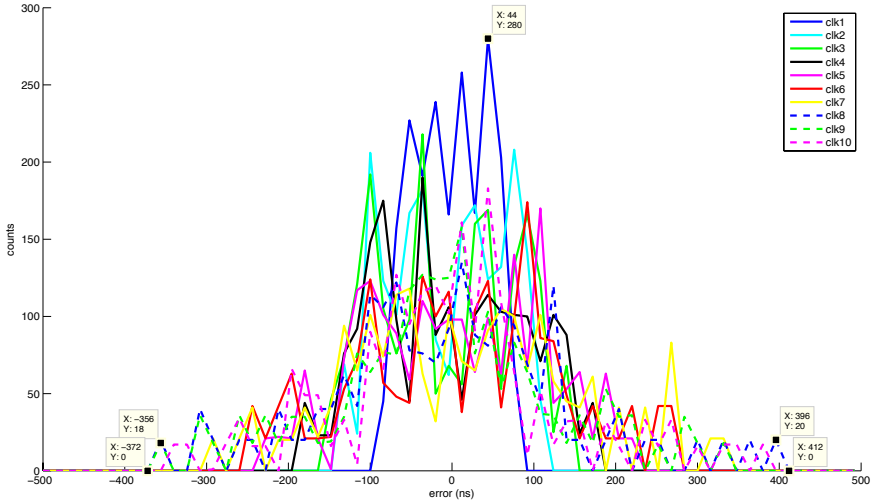
Figure 5.16: **Experiment principle diagram (Prototype-2)**: bidirectional configuration, initial frequencies of nodes are different

in Fig. 5.16 is from the reference and SCA1-1, which equals to 1, and the longest one is to SCA10-10, which equals to 19.

To measure the phase error of each cycle on the fly, a built-in measurement block is embedded in each node. This block has the same principle and architecture of PFD block but with a higher precision. The measurement result is captured by the digital analyzer and stored in a data file. Then we can observe the distribution of phase error from its histogram. Fig. 5.17 presents the histograms of each measured phase error in unidirectional mode (clk1: clock signal in SCA1-1...clk10: clock signal in SCA10-10). Fig. 5.18 shows the case of bidirectional mode. By comparing the two groups of histograms, we can observe that in unidirectional mode phase error is accumulated as the reference phase information travels further, just like in a conventional clock tree. While in bidirectional mode, which is the mode at which the network works at steady state, phase errors between all the nodes in the network and the reference are well constrained within ± 3 times PFD quantification steps. This can be better observed in the two tables which summarize the minimum/maximum values (Tab. 5.3) and RMS values(Tab. 5.4). Fig. 5.19 and Fig. 5.20 trace the maximum absolute value and RMS value in function of the distance, from which we can get the same conclusion as previous analysis.

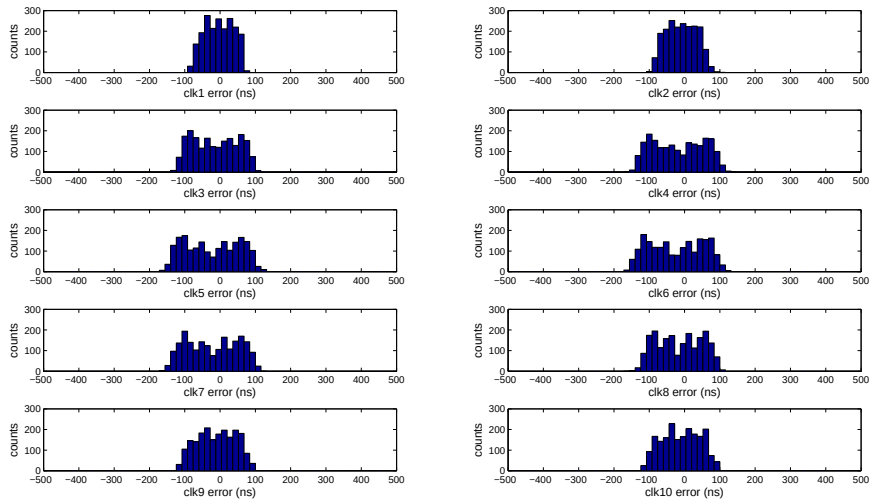


(a)

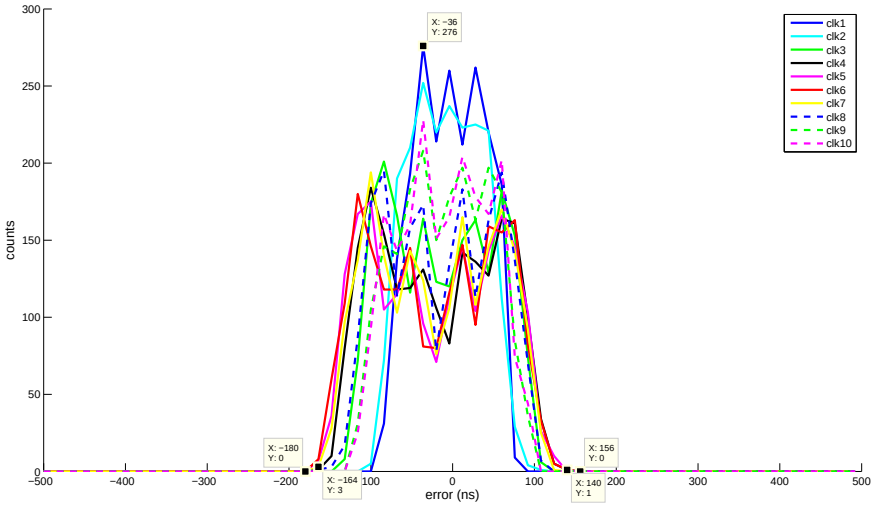


(b)

Figure 5.17: **Histogram of phase errors between local clock signals and the reference in** 10×10 **prototype**: unidirectional



(a)



(b)

Figure 5.18: **Histogram of phase errors between local clock signals and the reference in** 10×10 **prototype**: bidirectional

Table 5.3: **Minimum/Maximum values of phase errors between local clock signals and the reference in 10×10 prototype (*ns*)**

	Phase error (*ns*)	
Node	Unidirectional	Bidirectional
SCA 1-1	$-80 \cdots 80$	$-80 \cdots 80$
SCA 2-2	$-128 \cdots 112$	$-80 \cdots 96$
SCA 3-3	$-144 \cdots 144$	$-144 \cdots 128$
SCA 4-4	$-176 \cdots 176$	$-144 \cdots 160$
SCA 5-5	$-224 \cdots 224$	$-144 \cdots 160$
SCA 6-6	$-240 \cdots 272$	$-144 \cdots 176$
SCA 7-7	$-288 \cdots 336$	$-144 \cdots 160$
SCA 8-8	$-352 \cdots 400$	$-144 \cdots 128$
SCA 9-9	$-352 \cdots 336$	$-96 \cdots 112$
SCA 10-10	$-336 \cdots 384$	$-80 \cdots 112$

Table 5.4: **RMS values of phase errors between local clock signals and the reference in 10×10 prototype (*ns*)**

	Phase error (*ns*)	
Node	Unidirectional	Bidirectional
SCA 1-1	43.9191	39.9564
SCA 2-2	68.4472	42.2609
SCA 3-3	81.8778	63.4479
SCA 4-4	91.4903	72.7273
SCA 5-5	107.3229	76.9506
SCA 6-6	123.9706	77.3304
SCA 7-7	136.3794	73.5529
SCA 8-8	140.8230	64.3260
SCA 9-9	136.3400	53.9767
SCA 10-10	135.0824	53.7999

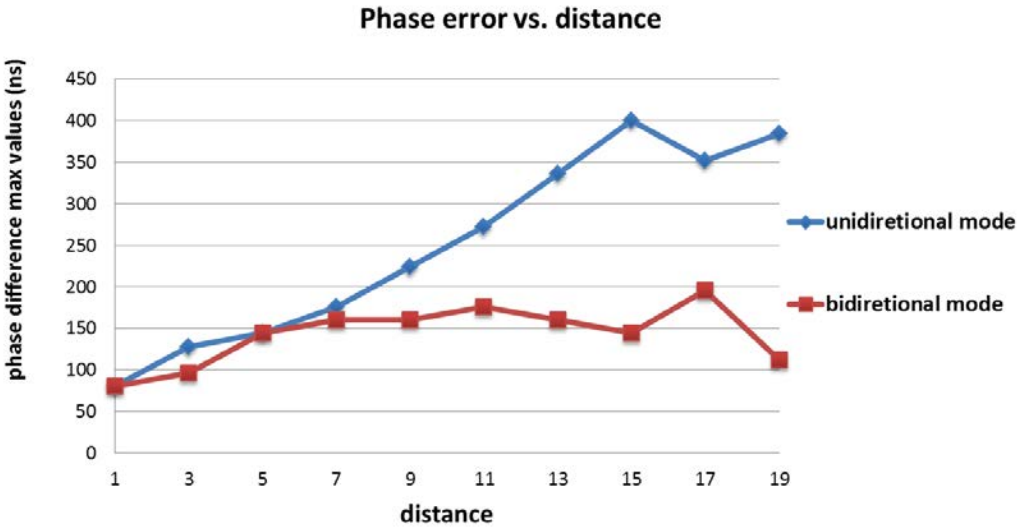


Figure 5.19: **Maximal value of phase error in function of distance to the reference clock**

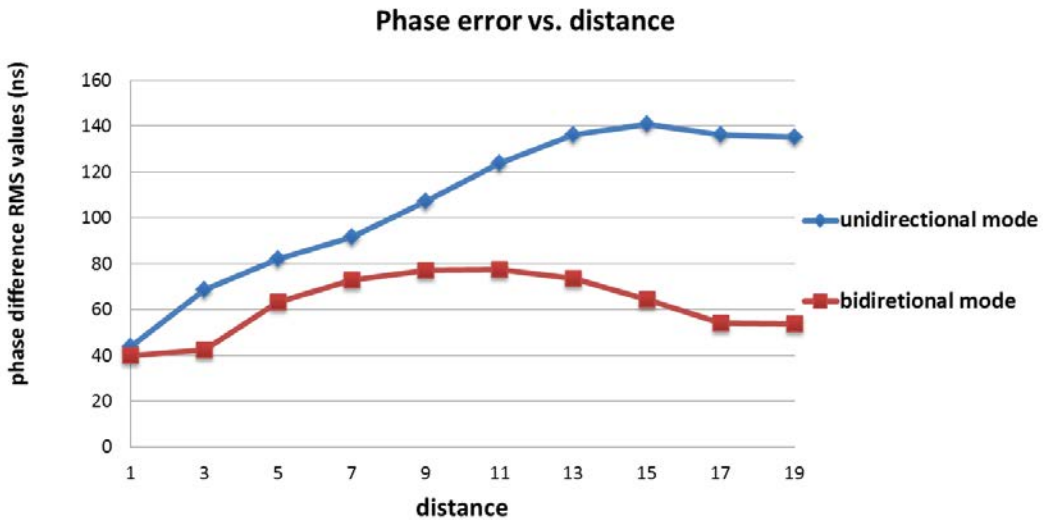


Figure 5.20: **RMS value of phase error in function of distance to the reference clock**

5.4 Conclusion

In this chapter we presented two FPGA prototypes implemented on Altera Cyclon II platform for the proposed clocking system. Both have validated the theoretical study results of the clocking solution.

The first prototype modelizes a 4×4 ADPLL network, which was designed before the implementation of the first ASIC circuit with the same architecture. The principal challenges of this experiment are the implementation and emulation of the mixed-signal blocks such as TDC and DCO in a 100 % digital environment. A frequency downscaling allowed an implementation of ADPLL network isomorphic compared with the original ASIC based implementation, and having similar behavior although operating at lower frequencies. The measurement results show that the operation of the proposed clock generator is similar to what predicted in theoretical studies (presence of mode-locking, numeric parameters of the transient, validity of the developed methods allowing a desirable state selection, etc.). The results of the work on this FPGA implementation were published in proceedings of FPT2011 conference [49].

To demonstrate scalability of the proposed architecture, a second FPGA prototype was designed before the second generation ASIC chip. This prototype has a dimension of 10×10 and parameters downscaled from those of the new ASIC circuit. Experimental results show the feasibility of this idea in such a large network. Other experiments performed on this prototype demonstrate that phase errors in the distributed clock generator are not accumulated, which also provides an argument for the scalability of the proposed solution. The results of this work were published in proceedings of Reconfig2013 conference [53].

Chapter 6

Clock network silicon implementation

Contents

6.1 Introduction 129
6.2 Methodology of chip design 130
6.3 Implementation of local clock generator (NODE) 134
6.4 Floorplan of the chip 138
6.5 Design for test(DFT) 139
6.6 Chip layout 145
6.7 Simulation results 146
6.8 Conclusion 148

6.1 Introduction

This chapter discusses practical issues of the 10×10 ADPLL network AISC prototype design in a 65 nm CMOS technology, using the PFD, DCO, digital error processing block and built-in test circuit presented previously.

The chip design applies an IP reuse strategy and mixed-signal digital-centric design flow, which are presented in Section 6.2. Then Section 6.3 presents implementation of the basic component of network — NODE. Chip floorplan is introduced in Section 6.4. Since this is a test chip, controllability and observability are very important for validation of circuit functionality and performance. The DFT issue is discussed in Fig. 6.5. At last, the top layout is displayed in Section 6.6 and simulation results are presented in Section 6.7.

6.2 Methodology of chip design

Since the chip design is hierarchical (cf. Fig. 6.1), each block in Fig. 6.1 can be regarded as an IP (Intellectual Property) cell, an IP reuse strategy is applied. The methodology explained below in this section is based on this strategy.

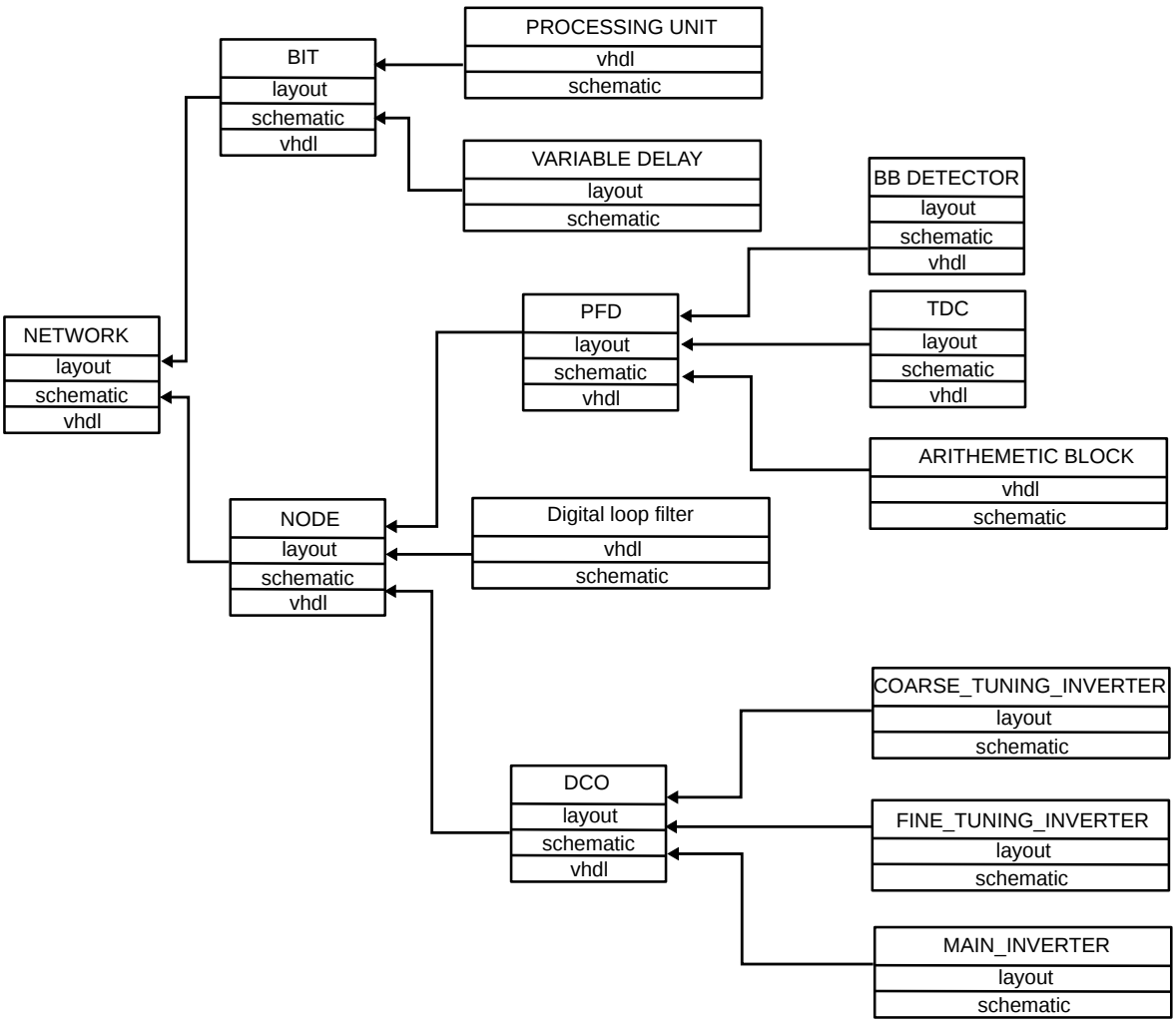


Figure 6.1: **Design hierarchy**

As explained previously, the performance of the designed clock generator is sensitive to the timing properties of some cells used for the PFD, DCO and the test circuit. For this reason, these critical blocks are designed in a custom or semi-custom way: some steps of the standard "compiled" design flow of digital circuits were made manually. In this way, the physical implementation of the ADPLL network deals with 4 kinds of physical cells:

- Standard cells provided by the Design Kits: basic logic cells, pads, etc.
- Full custom cells specific to the project. The schematic and layout are designed manually. These cells are DCO components, delay cell, C-element and arbiter for the PFD, delay cells in the on-chip test circuit, etc.

- Complex functional blocks obtained by standard digital synthesis flow: the filter, PFD encoder, processing unit in the on-chip test circuit, etc. These blocks use the standard digital cells of the DK.
- Semi-custom blocks, whose design uses the standard digital design synthesis flow but with some steps performed manually (e.g., the PFD, whose schematic level design is realized manually, and it uses both full custom cells and standard cells). These blocks are DCO, PFD, NODE, test circuit, etc. This type of the cells are defined as *hard IPs*, and are then used for the construction of the architecture of the ADPLL network at the top level.

In order to assemble the hard IPs together, we chose the approach "digital on top". It means, that all blocks of the network are integrated in a digital design flow, and the final assembly of the chip is done in the digital design environment.

As shown in Fig. 6.2, the IC design environment of Cadence was used for this chip design. We used the Virtuoso tool for the custom design of the critical cells, and Encounter for automatic place and route of the complex blocks and on the top-level assembly. Custom cells are regarded as black boxes. Basic geometry information of the custom and hard IP cells, especially layer and location of pins and stripes, is extracted from custom layout and is stored in the cell LEF file. This information is enough for automatic floorplanning and placement by Encounter. The complete layout data of a custom cell is stored in the GDSII format and is imported to Virtuoso for the final tasks of the chip integration and verification.

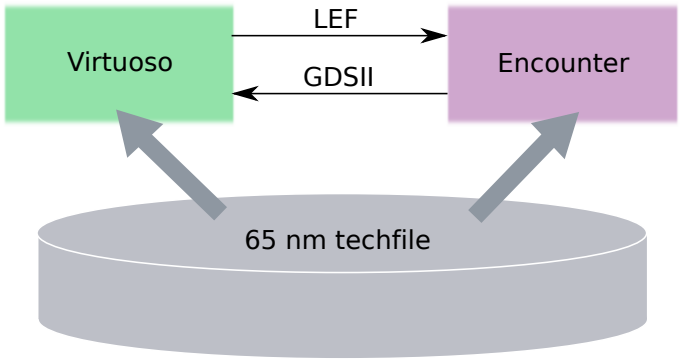


Figure 6.2: **Design environment**

More details about the design flow is provided in Fig. 6.3. The diagram presents a top-down design flow based on digital-centric mixed signal design methodology.

On the left side, the green flow diagram describes the design flow of full custom cell IP. In this chip, this kind of cells are the DCO cells and the DCO core, some cells used in the PFD and voltage controlled variable delay in the test circuit. These cells are designed in schematic level, then the layout is drawn in Virtuoso. The extracted LEF file are provided to higher level block. LIB file supplies timing information of the circuit for higher level timing optimization and routing.

The middle diagram presents the design flow of hard IPs. They are obtained from standard and custom cells, and they use some steps of standard digital flow, in particular, the automatic place and route. Their schematic is done manually.

The design flow of the top circuit is similar, except that it uses the hard IP cells as building blocks for the top-level layout.

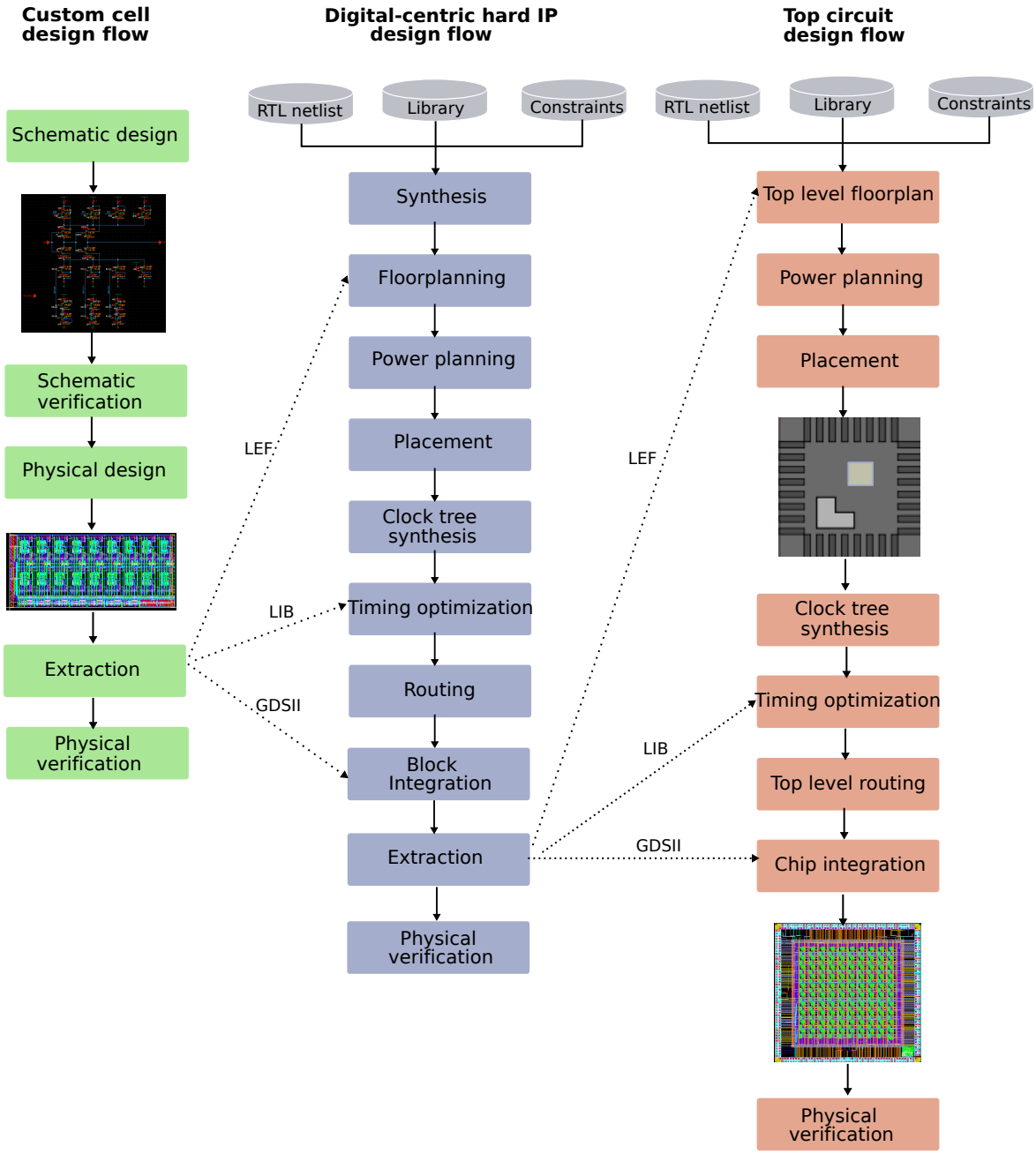


Figure 6.3: **Top-down design flow**

Verification is performed along with the design in different steps. As shown in Fig. 6.4, the arrow direction signifies the dependency in the flow. Since the bottom blocks like custom cells and pure digital cells can be designed in parallel, their verification can also be performed independently. Then the functionality of higher-level blocks and of the whole chip can be verified by behavioral simulation. Two kinds of models are used for behavioral modelling of

the custom cells: at the early design stage, the model is built with use of the desired block parameters given by the specification; at the final verification stage, the behavioral model uses the post-schematic or post-layout simulation results, so to describe block functionality with precision. If the design passes behavior level verification, we can performed more precise and time consuming *mixed* simulation. The mixed simulation mixes the models defined at different levels (VHDL, transistor, extracted schematic view, etc.), in order to verify the impact of the realistic behavior of critical blocks on the system operation.

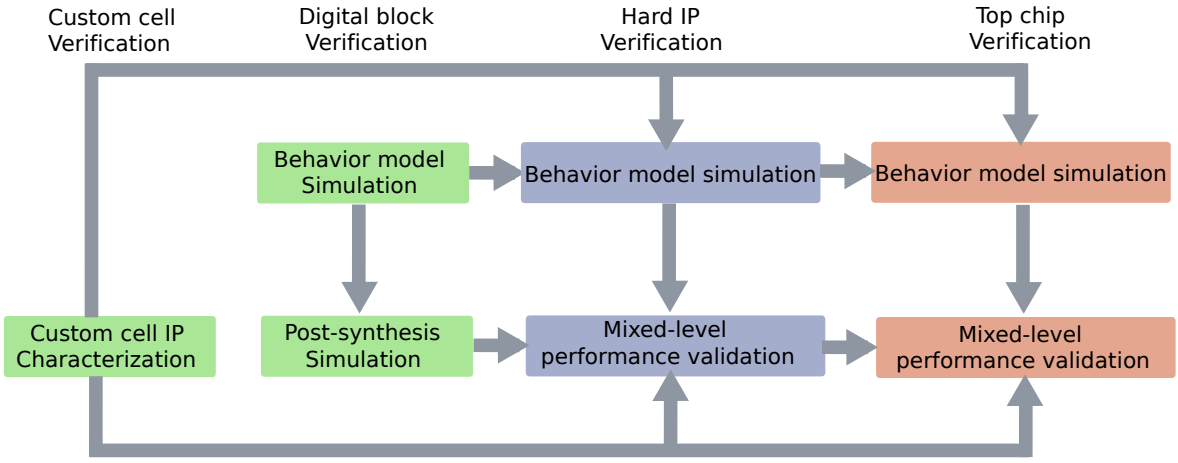


Figure 6.4: **Top-down verification flow**

6.3 Implementation of local clock generator (NODE)

Since the network has a regular mesh structure, and the local clock generators in different isochronous zone possesses the same components (DCO, PFD, filter, etc.), the physical implementation of the network follows the methodology of an hierarchical IP reuse. A mixed IP block named NODE is designed as the basic node element of the network (Fig. 6.5). It contains the following blocks:

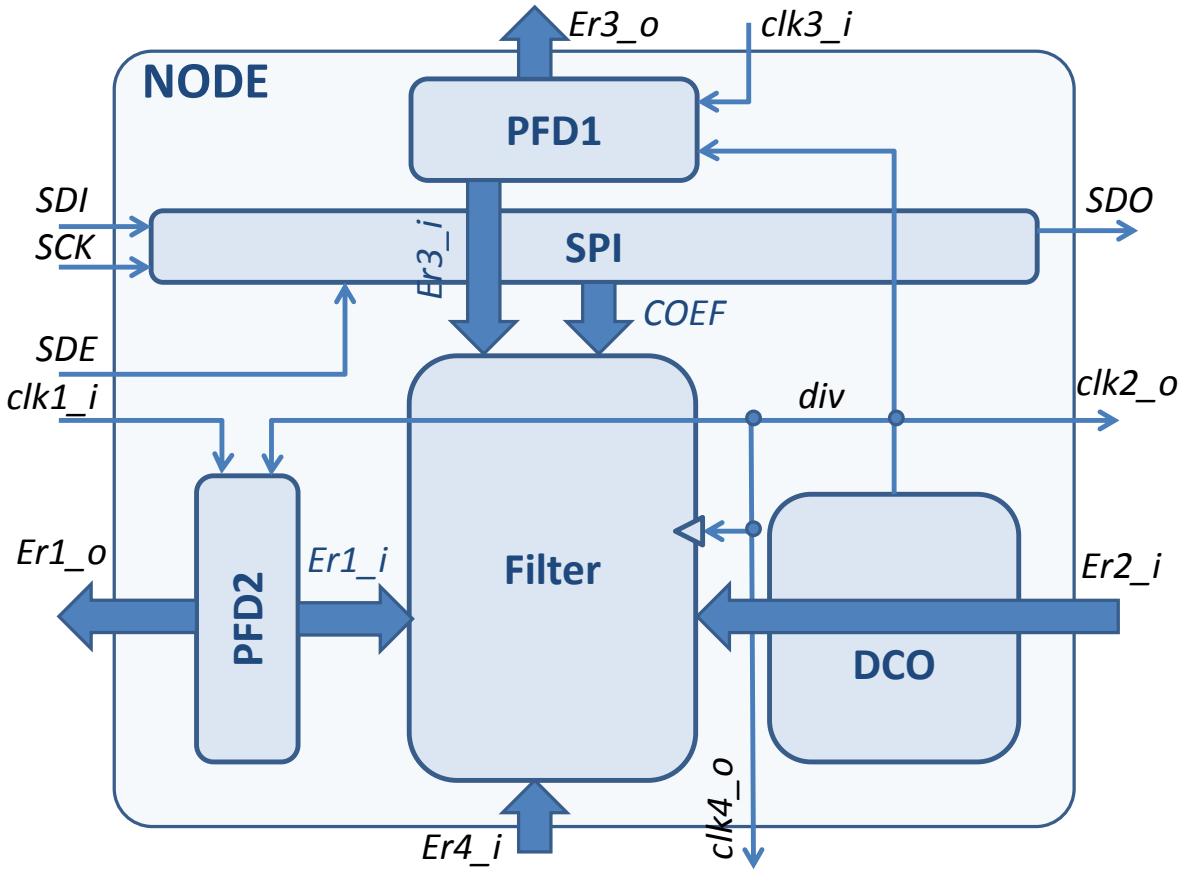


Figure 6.5: **Structure of a NODE**: one DCO, two PFDs and digital processing block (SPI and loop filter)

- Two PFDs: PFD1 and PFD2 compare the local divided clock signal with clocks generated by the neighboring NODE on top and the one on left, respectively. These two blocks are shared with its neighbors when we combine several NODEs together. Since the mesh is two dimensional, two PFDs are enough: one for vertical neighbors, the other for horizontal ones;
- Loop filter: the digital processing unit receives the output of local phase frequency comparators PFD1 and PFD2 and also comparison results with the other two neighboring clocks from two primary inputs of NODE (Er2_i and Er4_i). It processes the phase errors and generates a control word for DCO;
- DCO: the local clock generator. Its signal, after the frequency division, is used for the phase comparison, and for cadencing the filter of the node.

- Serial programming interface (SPI): a shift register streaming in the 1-bit series data from one side and streaming out on the other side for the next NODE. The data correspond to the values of the filter coefficients and to the connectivity parameters defining the operation of each node. When the enable signal SDE is activated, it stores the data in a parallel register, and the values are permanently available on its parallel output ports *COEF*.

Fig. 6.6 shows the layout of NODE. It should be mentioned that the location of pins is chosen so that when the network of ADPLL is assembled, the neighboring nodes formed by the NODE block can be connected by abutment or by short wires. For example, *clk1_i* and *clk2_o* are on the same horizontal level, *Er4_i* and *Er3_o* have the same vertical coordinate. It is the same case for *clk3_i* and *clk4_o*, *Er2_i* and *Er1_o*, *SDI* and *SDO*. The dimension of the NODE block is 149 $\mu m \times 145.3\ \mu m = 0.014 mm^2$.

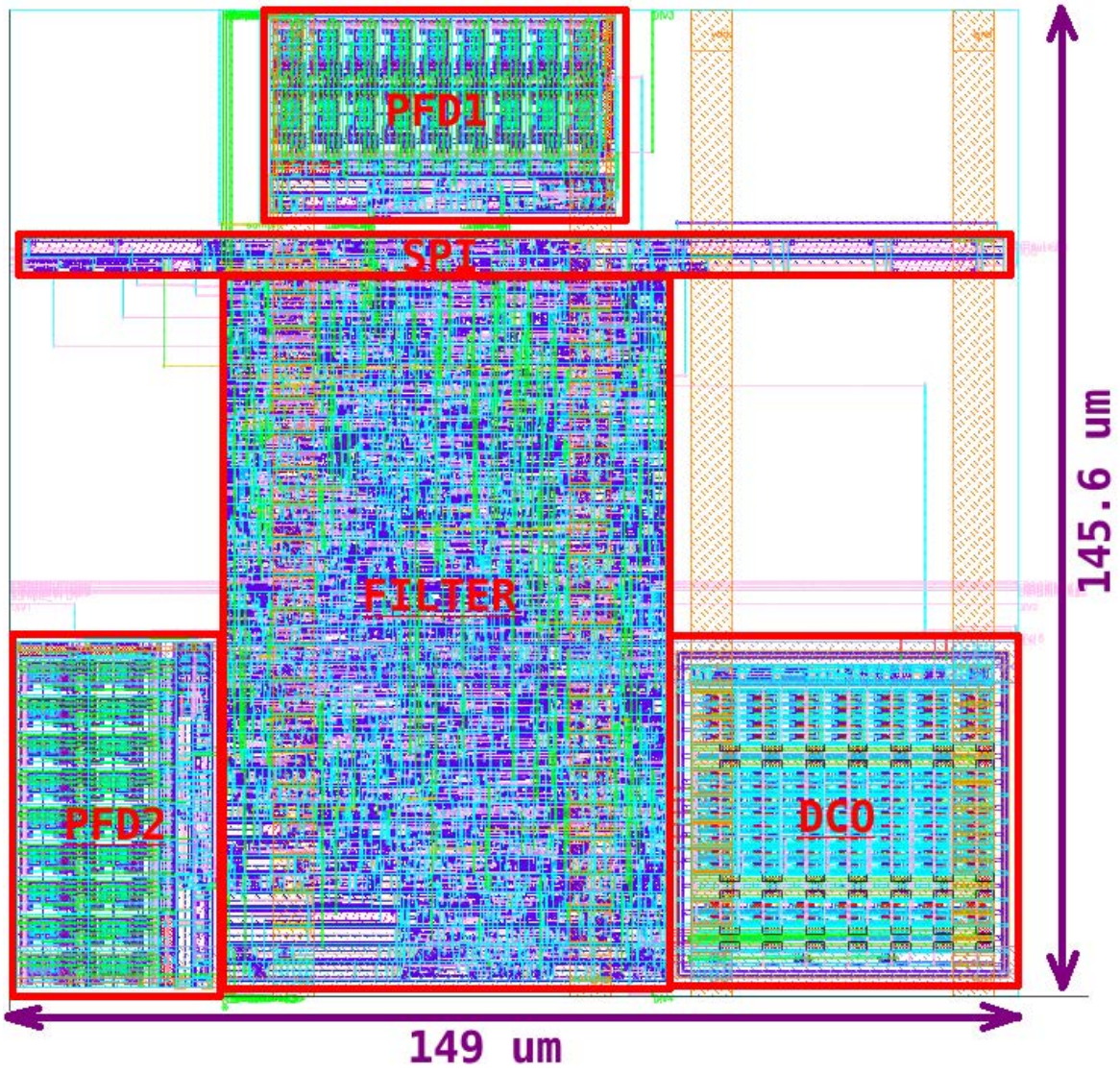


Figure 6.6: **Layout of the block NODE**: one DCO, two PFDs and digital processing block (SPI and loop filter)

Table 6.1: **Simulation parameters and conditions**

Parameter	Value
F_{ref}	265.8 MHz
α	1
β	0.005
K_p	0
K_i	20
Kw_1,Kw_3	1
Kw_2,Kw_4	0
F_{div} initial	258 MHz

To verify functionality and performance of a single NODE, a transistor-level simulation is performed using post layout extracted model of the NODE. The parameters of this modeling experiments as well as the programmed coefficient values are summarized in a Tab. 6.1. To test the operaiton of the sub-system of the NODE block, two identical reference clock signals drive the two ports *clk1_i* and *clk3_i* in Fig. 6.5. A programming sequence defining the filter coefficients and the connectivity of the node is generated at the beginning of simulation.

The plots of Fig. 6.7 present the waveforms of the divided frequency of the DCO, of the filter output code and of the output of the PFD obtained by a post-layout transistor level simulation. After the power up, the node is in the *programming mode* (Section A in Fig. 6.7): the correct values of the programmable parameters are being loaded, and the correct values are only set at the end of the programming mode. The output signals of the ADPLL blocks are irrelevant during this mode. The signals SDI, SCK and SDE are generated by voltage sources whose waveforms are read from a data file. The datafile used for the programming is generated by a script (cf. Appendix C). When the filter coefficients have correct values, the *reset* signal is sent: the ADPLL starts the *frequency and phase acquisition mode* (Section B in Fig. 6.7), where it adjusts the frequency and phase of the local clock according to the reference signal. In the *phase tracking mode* (Section C in Fig. 6.7), the ADPLL output signal is in phase with the reference signal. It should be mentioned that this is a test of the node functionality, which doesn't target an optimal operation. The desired performance can be achieved according to specification by modifying the coefficients, for instance, if a shorter frequency acquisition time is needed, we should program a larger K_i.

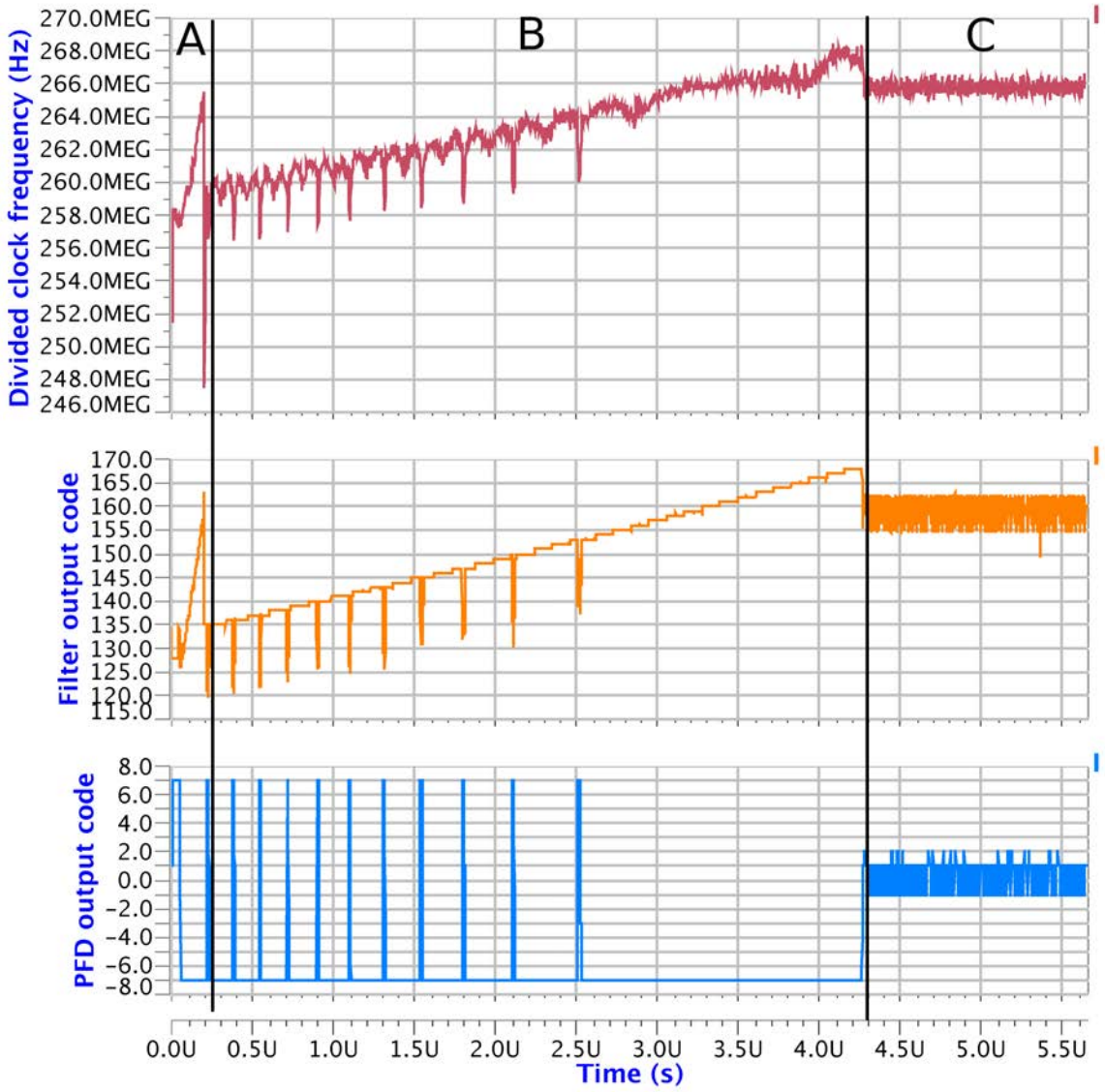


Figure 6.7: **Post-layout simulation of one NODE**

6.4 Floorplan of the chip

The chip is mainly composed of 100 assembled NODE blocks presented in the previous section. They constitute the *core* of the implemented chip, responsible for the clock generation. Four built-in test circuits (BIT) are implemented around the core for the phase error characterization. A preliminary floorplan of the implemented circuit is given in Fig. 6.8. The core area is partitioned into SCAs where the NODEs are placed. Between the neighboring NODEs, a free space of $35\mu m$ width is reserved for the routing: in particular, for the the communication lines between NODEs and for the connections to the chip pads of the signals we want to observe. In the real conditions, the free space inside the NODE and the routing channel will be used by the functional circuit. Since the number of the test signal we route to the chip pads is big, our chip is "I/O constrained": that means that the size of the chip is constrained by the size of the pad ring, and not by the size of the core circuit. The free space left on the chip is filled with fillers, decoupling capacitances or left empty.

The floorplan of the chip is defined by a parametrizable TCL script, which generates automatically the block placement in the Encounter environment. The parameters of the script are: the size of blocks and pads, the number of pads at each side, the space between blocks and the minimum space between chip border and the core. The parametrizable script allows a parallel design of the blocks and the global chip floorplanning: if the layout of a cell is modified, the chip floorplan can be immediately regenerated without extra work. Details of this script can be found in Appendix E.

Several challenges exist in the test chip floorplan.

First, the programming interface is distributed over all error processing blocks using the technique presented in Fig. 3.19 and Fig. 3.20. While on the top chip level, how to program the 100 NODEs in an effective way without blocking the routing channel is the first issue to be think about in the physical design stage.

Second, since the network has a large dimension and each NODE has several functional blocks, in order to verify the chip functionality we cannot observe all the signals. Moreover, although the built-in test circuit doesn't cost much, it cannot be put everywhere in the chip, because it needs pads to output the test results, and the number of pads is the dominant factor of the chip size. How to test the chip with limited number of pads directly decides the cost of test chip.

The two floorplan challenges consist in chip controllability and observability, which are two main issues in design for test study.

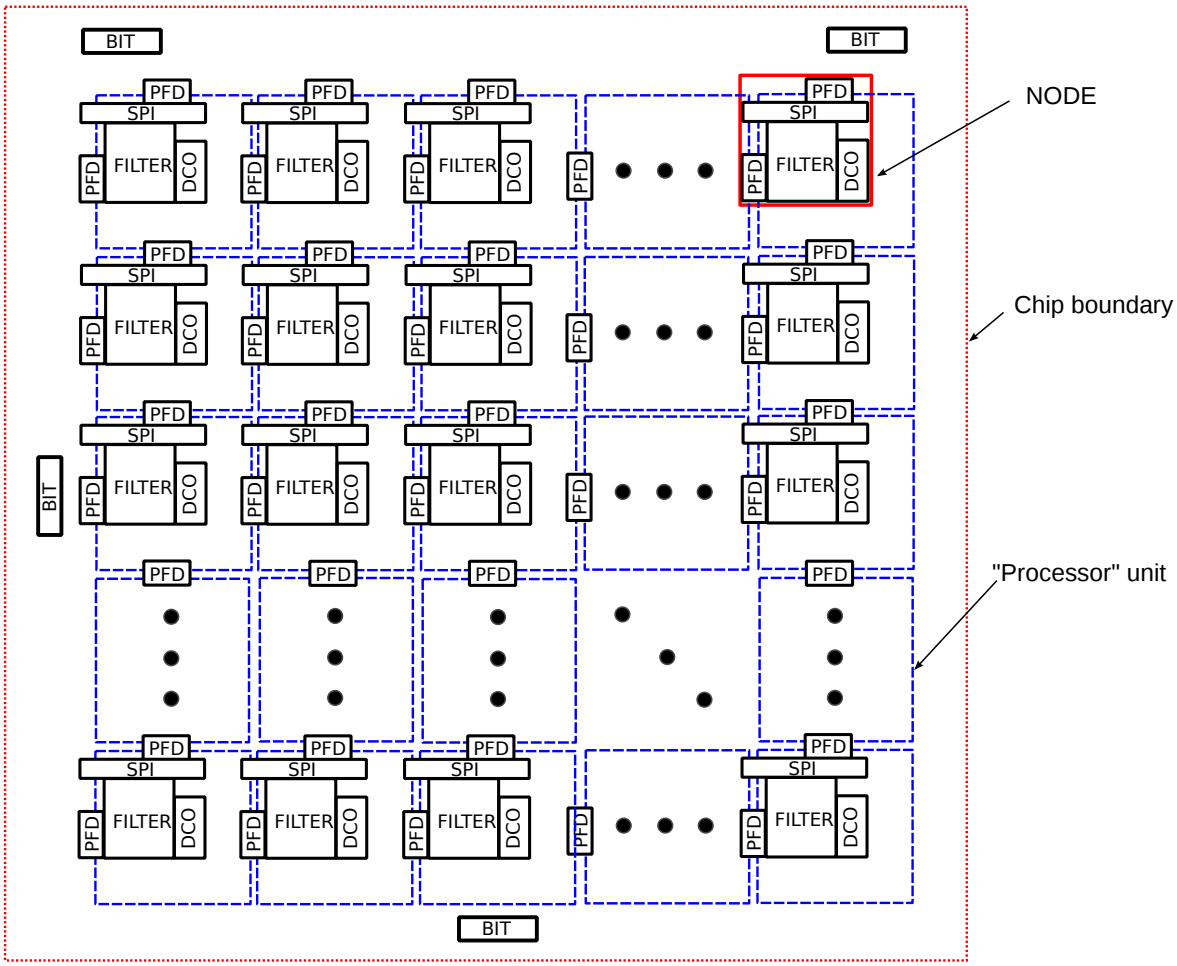


Figure 6.8: **Preliminary floorplan of the test chip**

6.5 Design for test(DFT)

As introduced in [32], DFT is an important stage of the chip design. The term DFT designates all issues, techniques and strategies making possible test, validation and verification of the chip after the fabrication. The DFT related functions in this test chip are the following:

- The possibility to program the chip parameters. This allows a reconfiguration of the network topology and a choice of the loop filter coefficients. It multiplies the number of configurations in which the chip can be tested, so it increases the impact of the chip test on the validation of the concept of ADPLL network based clock generation.

- The routing of critical and relevant internal circuit signals toward the chip pads, so to make them observable ([66],p.114). Since the number of pads is limited, the selection of the internal signals for observation depends on the strategy of the chip test, aiming a maximization of the knowledge about the chip operation.

The next two subsections present these two issues.

6.5.1 Chip programming

The programming interface for each block was presented in Section 3.3.4. As explained in that section, the interface is designed so to be extensible: by cascading the programming interfaces, any number of blocks can be programmed with the single programming process. The required number of the external input chip pads (3 pads) does not depend on the number of cascaded blocks.

The topology of the programming interface cascading on the designed chip is given in Fig. 6.9. The corresponding programming sequence for the developed network consists of S bits

$$S = N \cdot (S_i + S_p + S_{kw} \cdot 4 + S_{num}) \tag{6.1}$$

where N=100 is number of the network nodes, S_i=8 is width of the integral path coefficient, S_p=2 is the width of the proportional path coefficient, S_{path}=2 is the width is the input gain control coefficients, $S_{num} = 2$ signifies the number of neighboring clock inputs taken into consideration. The length of one node is 20 bits, while for the designed network, the total length of a programming sequence is 2,000 bits. They can be transferred with a high rate (tens of MHz) and in this way a total time for the network programming procedure will be negligible (200 μs @ 10MHz).

The connection topology of the programmable blocks of the network is depicted in Fig. 6.9. It is programmed in a zigzag way to avoid long routing wires from the last node of one row to the first node of the next row. As explained in Section 6.3, each node has SPI interface: the input programming signal pin *SDI* on one side and the output pin *SDO* on the other side. This allows programming the nodes effectively in sense of the shortest routing distance. However, since the nodes of odd and even rows are programmed in inverse directions, we need two kinds of NODEs with exactly the same functionality and floorplan but different pin locations: one type of NODE in odd rows with *SDI* on the left and *SDO* on the right; the other type in even rows with opposite pin position.

A correct configuration of the chip programming is the first thing to verify before other functional verification. In order to control the correctness of the programming, we reserved four chip pads connected to four bits of the programming interface registers, situated at fours network corners (observation points in Fig. 6.9).

6.5.2 Built-in test circuits placement

The PFD blocks at the node boundary only compare the two neighboring clock signals. For reasons explained in Subsection 1.2.10, a special analog circuit named built-in test circuit (BIT) is designed for measuring phase error between two clock signals in distance. Due to

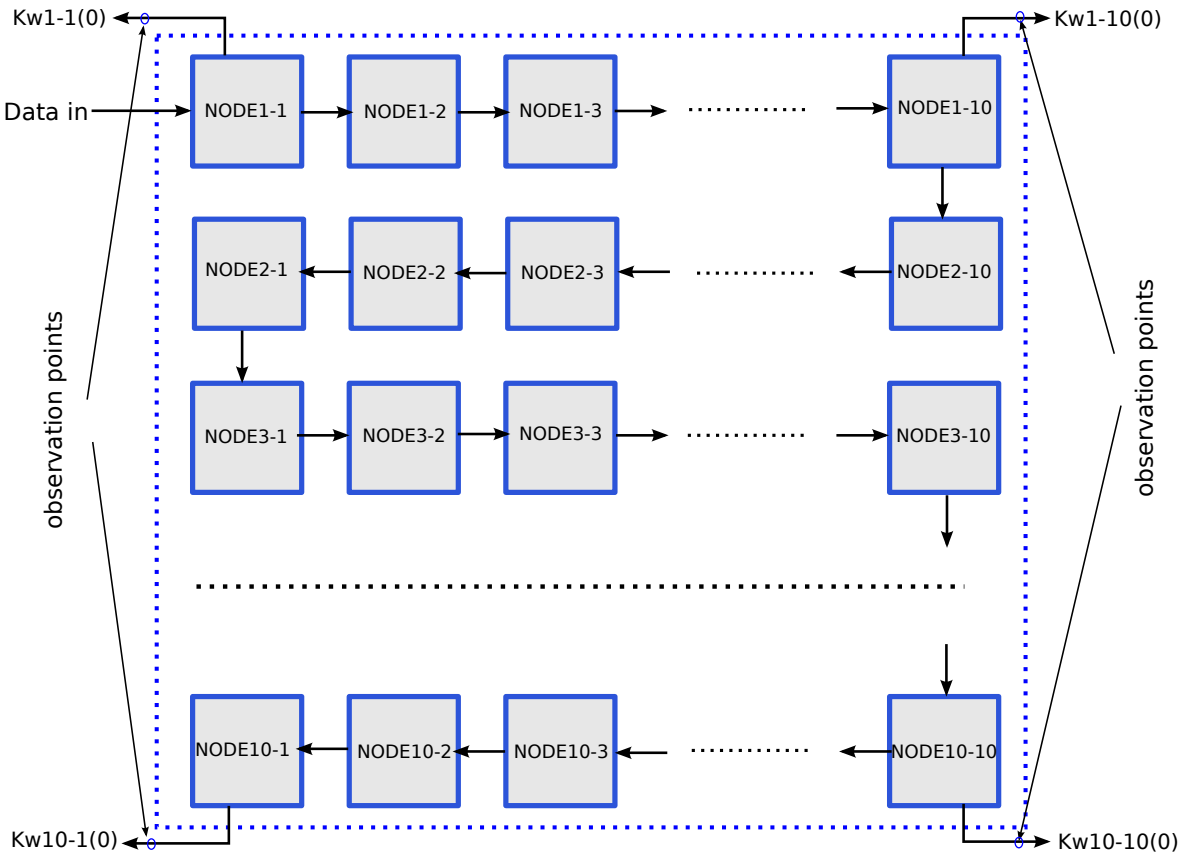


Figure 6.9: **The connection sequence of the programmable blocks of the network**

the limited chip area and pad number, we decided to implement four BIT blocks in the test chip (Fig. 6.8) for the following tasks:

- one BIT block at the northwest corner of the chip measuring the phase error between the reference clock and NODE1-1;
- the second BIT block at the west border of the core measuring the phase error between the reference clock and center node NODE5-5;
- the third BIT block at the northeast corner of the chip measuring the phase error between the reference clock and NODE10-10;
- the last BIT block at the center of south border measuring the phase error between NODE10-5 and NODE10-6.

The first three BIT blocks in the above list allow collecting phase error information as a function of distance from the reference clock signal in the northwest corner. The last BIT block tests the clock difference between two neighboring nodes.

6.5.3 Definition of the input/outputs of the chip

The pins are classified in five categories: control signals, input signals, output signals, built-in test circuit signals and power supplies.

1. control signals for network (8 pads: 4 digital input pads + 4 digital output pads)

Control signals are used for programming and configuration of the network.

- RST (1 bit): a global reset signal.
- SDA (1 bit): a stream of programming data.
- SCK (1 bit): programming clock signal.
- Enable (1 bit): the signal which allows loading programming bits to local coefficients register.
- Digital output pads (4 bit): output 4 bits of programmed coefficients for verification.

2. Input signals of network core (4 pads)

The only input signals of the network are the four reference clock signals. As presented in previous chapters, normally only one reference clock (the one in the northwest corner) is used. Due to the utmost importance of this signal, reference clock inputs are reserved at each chip corner to improve the chip reliability after fabrication. Another reason is that for some configurations, more than one reference clocks are needed, which is presented later in Chapter 7.

3. Output signals of network (60 output pads)

- Replicas of reference (4 pads): Reference clock signals at the node input ports are connected to output pad for verification during chip test.
- Divided DCO clock signals (36 pads): Since the network has 100 locally generated clocks, if we want to observe all of them, one or more multiplexers are needed. They will introduce additional delay and skews, which is unfavorable for chip test and debug. Fortunately, the network is symmetric, we decided to output certain typical local clock signals in this chip: the 20 clock signals in two diagonals, 8 clock signals in a vertical border, and 8 clock signals in a horizontal border, thus 36 pads totally.
- PFD output (20 pads: 4×5): The outputs of 4 PFDs are readout for test. Each PFD has 4 signed bits and another 1-bit signal (inner reset signal in Bangbang detector) should be used for synchronization. Hence 5 pads for each PFD and 20 pads in total. The four PFDs are the follows:

  1. PFD between reference and NODE1-1
  2. PFD between NODE6-1 and NODE7-1
  3. PFD between NODE9-1 and NODE10-1
  4. PFD between NODE10-9 and NODE10-10

4. Built-in test circuit signals (52 pads: 12 inputs + 40 outputs)

Input signals (3 pads)

Table 6.2: **Summary of IO pads**

category	input	output	sum
control signals	4	4	8
PLL network	4	60	64
BIT circuit	12	40	52
total	28	104	124

- testEnable (1 bit): to enable the test. Otherwise, the device is shut down to save energy.
- voltage control signal (2 pads): one voltage signal for the control of each variable delay.

Output signals (10 pads: 8 digital output bits + 2 oscillator clock signals)

- digital output bits (8 pads): the number of errors is counted by a 8-bit counter and the result is sampled every 2 power 8 cycles.
- oscillator clock signals (2 pads): to measure the exact delay values, the replicas of each delay is connected as a ring oscillator. The oscillator clock signals are measured outside of the chip to get the delay values.

We plan to put 4 test circuits in different locations of the chip, thus the total number is 52 pads (13×4).

5. Power supplies (76 pads: 24+4+10 pairs)

- VDDE/GNDE (24 pairs): among all the output pads above, the dynamic output pads are divided clock signal (36 pads), PFD output (20 pads), test circuit output data (32 pads) and test circuit oscillator clocks (8 pads). The total number is 96 pads. If we use one VDDE/GNDE pair for each four pads, we need 24 pairs.
- VDDA/GNDA (4 pairs): one pair for each built-in test circuit
- VDD/GND (10 pairs): global power supply for the core

Therefore, we need 32 pairs (24+4+4) of power supply, thus 64 pads.

The summary of I/O and power supply pads are displayed in Tab. 6.2 and Tab. 6.3. In summary, the chip has 28 input pads, 104 output pads and 76 power supply pads. It is totally 200 pads.

Table 6.3: **Summary of power supply pads**

category	number of pairs	number of pads
VDD/GND	10	20
VDDE/GNDE	24	48
VDDA/GNDA	4	8
total	38	76

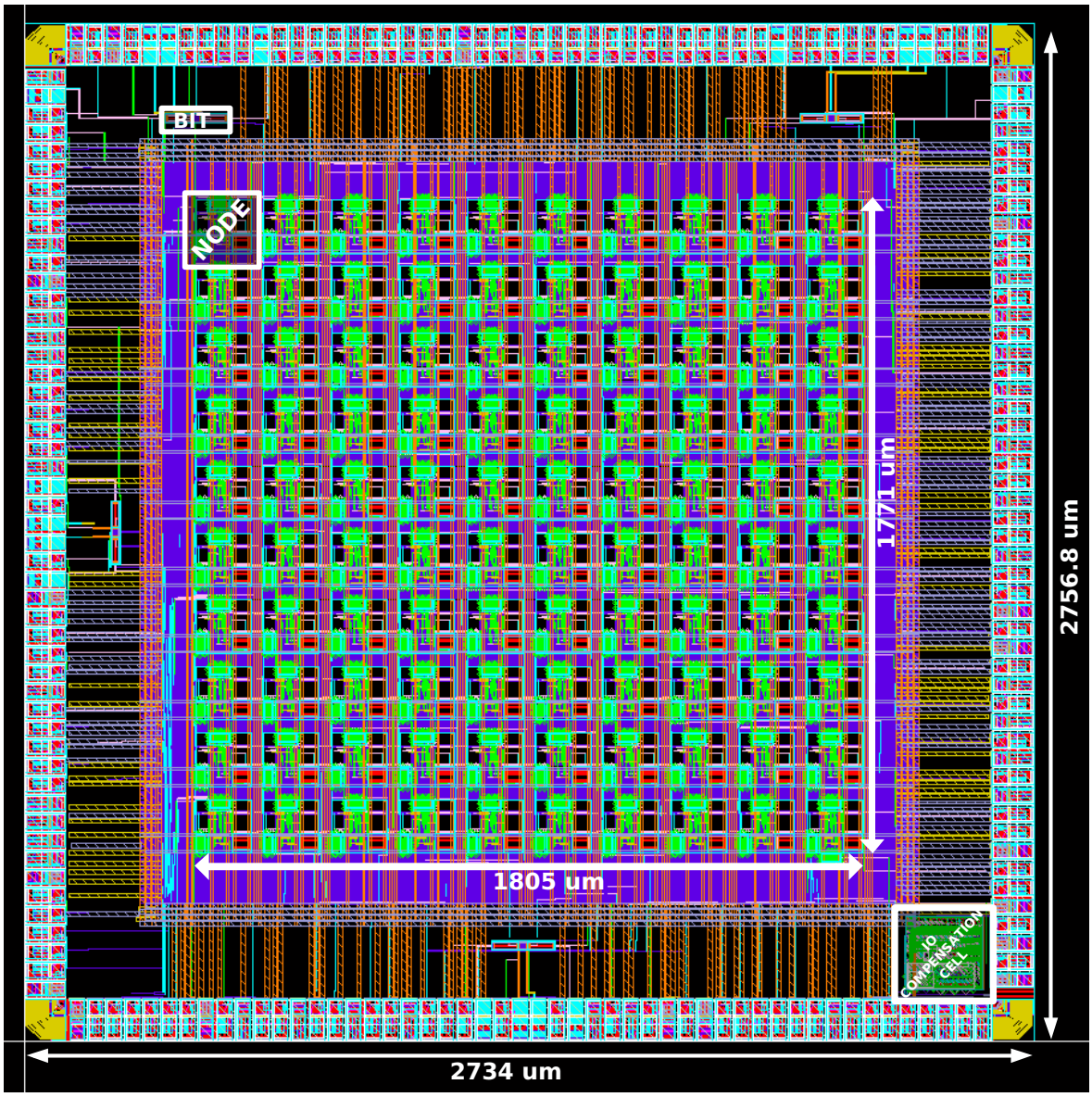


Figure 6.10: **Layout of the test chip of the clock network**

6.6 Chip layout

The assembling of the clock network blocks has been done in a standard digital design flow with a help of EDA tools. The actual layout of the chip is given in Fig. 6.10. The size of the chip is $2734 \times 2756.8\ \mu m^2$ where clocking network core occupies $1805 \times 1771\ \mu m^2$.

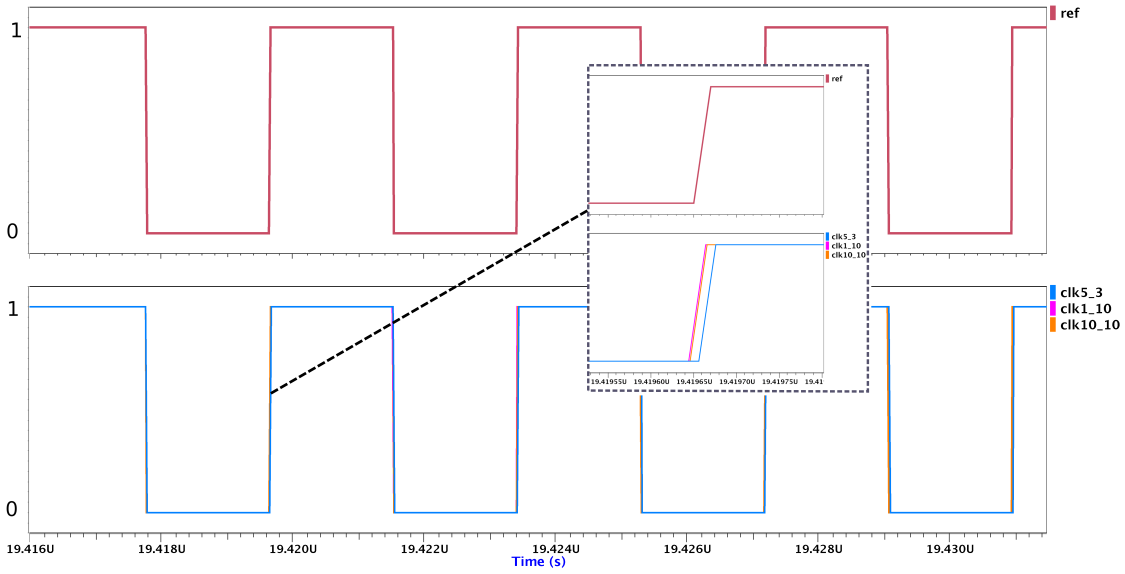


Figure 6.11: **Clock signals in the network in steady mode**

6.7 Simulation results

During the edition of this manuscript, this test chip is under fabrication, thus no measurement result is available. Due to the long time of transistor level simulation for a whole network, only the network start-up is simulated, which allows a verification of the programming interface and the basic functionality of the blocks. In order to check the performance of network in steady state, VHDL level simulations have been performed.

Fig. 6.12 shows the reference clock wave and some locally generated clock signals in steady mode. We can observe that all the signals are synchronized in frequency and in phase. Fig. 6.12 illustrates the outputs of PFDs around NODE5-3. When the whole system starts working after programming period, there is a short period of time (about 500 ns) during which the signals varies sharply. This is the frequency acquisition process. After that the phase tracking starts and phase errors are attenuated.

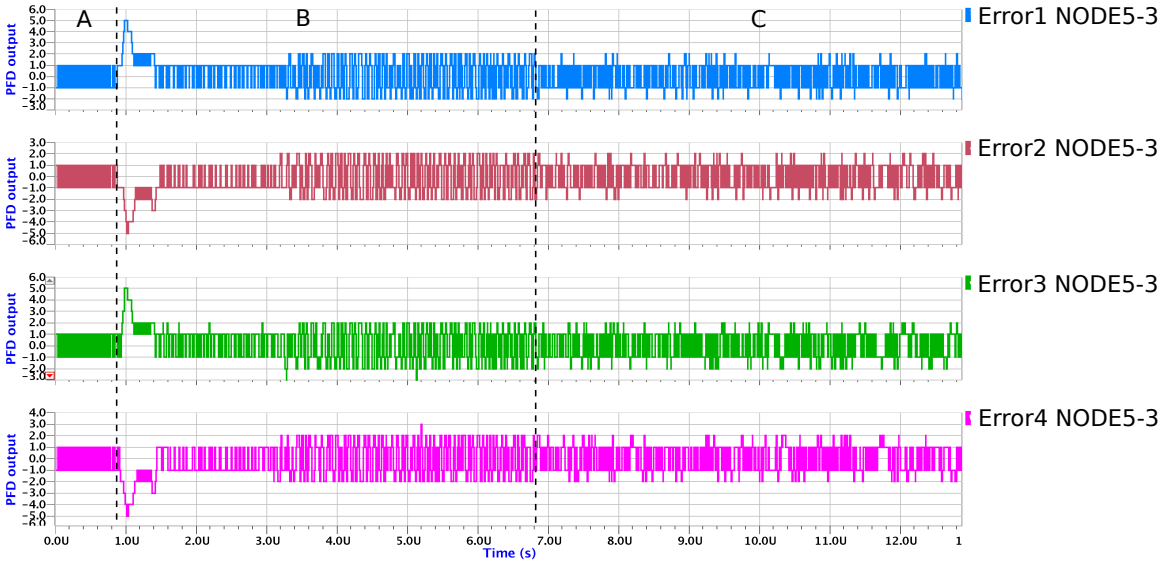


Figure 6.12: **PFD outputs between NODE5-3 and its neighboring nodes:** A) programming mode; B) unidirectional mode; C) bidirectional mode

6.8 Conclusion

In this chapter, the implementation of a 10×10 network ASIC prototype is presented. This chapter can be divided into three parts. The first part presents the design methodology and the used design flow. The second part presents the topology and layout of the basic construction block of network: the NODE block, which is repeated 100 times in the network. The last part from Section 6.4 to Section 6.7 discusses practical implementation of the chip. This chip is presently under fabrication. Its functionality was validated by simulation at different abstraction levels.

Chapter 7

"Swimming pool"-like distributed architecture

Contents

7.1 Introduction 149
7.2 Modeling of infinite ADPLL network by a continuous wave propagation medium 150
7.3 ADPLL network with limited surface 154
7.4 Simulation results 156
7.5 Conclusion 160

7.1 Introduction

This chapter studies the robustness of the ADPLL network with regard to the perturbations, when the network is globally synchronized. If a node of a large synchronized ADPLL network is get perturbed (i.e., for some reason, its phase or frequency becomes very different from the neighbors), the error may propagate, affecting several nodes of the network in the nearby area. If the perturbation is large enough so that phase error propagates to the border of the network before it is attenuated locally, since the network has a limited dimension, phase error may be reflected and results in a wave interference and even in a standing wave.

The work presented in this chapter is inspired by the theory of wave propagation on a liquid surface. In Section 7.2, we start by studying one local clock generator in a large unlimited network. Then, we introduce an analogy between the surface of the phase error in a network of ADPLL and a water surface. In Section 7.3 we study the phenomena of error wave propagation and reflection in an ADPLL network with limited surface, and we also explain the advantage of proposed "Swimming pool"-like architecture in preventing error wave reflection. Simulation results are presented in Section 7.4 to demonstrate the performance of proposed architecture.

7.2 Modeling of infinite ADPLL network by a continuous wave propagation medium

This section shows that a network of ADPLL as it is designed in this PhD project can be modeled by equations describing the waves propagating on a liquid surface, or an elastic membrane. The link between the corresponding wave equation and the parameters of the ADPLL network is demonstrated.

The derivation of the equivalent model starts from the assumption that the ADPLL network is a space-discrete model of a continuous wave propagation medium. This vision is presented in the fig. 7.1, which puts again the architecture of the ADPLL network and the structure of one node. The dynamic quantity of the media is the phase of each oscillator, which, in the scale of the network, depends on time and on the coordinates. In this section, we study an infinite ADPLL network, or an area of the ADPLL network which is far from the borders.

In this analysis, we consider the continuous time model of the ADPLL network, since the mathematical tool related to the wave propagation are essentially developed for continuous time.

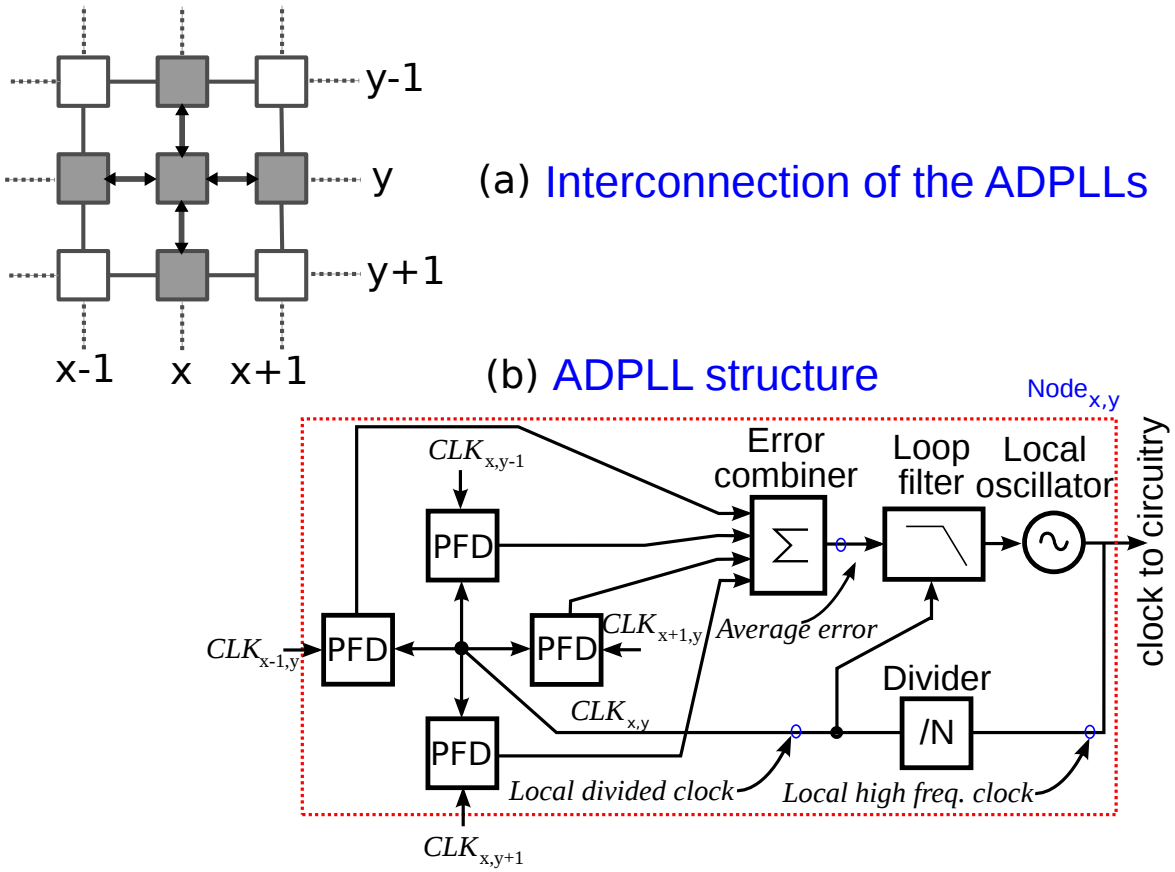


Figure 7.1: **Interconnection and structure of the ADPLLs**

7.2.1 From a discrete network to a continuous medium

In this section we show how the discrete space structure of the ADPLL network can be modeled by partial differential equations defined in continuous space.

Each node of the network modeled in the continuous time can be described by ordinary differential equations. These equations can be obtained if one considers the transfer function of each block expressed in the Laplace domain:

$$\begin{aligned} H_{\text{PFD}} &= K_{\text{PFD}} = \frac{1}{\delta_{PFD} \cdot 2\pi f_s} \\ H_{\text{Filter}} &= (K_p + \frac{K_i}{s}) \cdot e^{-s\tau} \\ H_{\text{DCO}} &= \frac{K_{\text{DCO}}}{s} \end{aligned} \tag{7.1}$$

where K_{PFD} and K_{DCO} are the gains of PFD and DCO. δ_{PFD} is the timing resolution of PFD and f_s is the sampling frequency. K_p and K_i are the gains of the proportional and integral paths in the PI filter, respectively. τ is the delay in loop filter. The closed loop transfer function of feedback system can be expressed as:

$$\begin{aligned} H &= \frac{H_{\text{PFD}} \cdot H_{\text{Filter}} \cdot H_{\text{DCO}}}{1 + H_{\text{PFD}} \cdot H_{\text{Filter}} \cdot H_{\text{DCO}}} \\ &= \frac{K_{\text{PFD}}(K_p + \frac{K_i}{s})e^{-s\tau}\frac{K_{\text{DCO}}}{s}}{1 + K_{\text{PFD}}(K_p + \frac{K_i}{s})e^{-s\tau}\frac{K_{\text{DCO}}}{s}} \\ &= \frac{K_{\text{PFD}}(K_p s + K_i)e^{-s\tau}K_{\text{DCO}}}{s^2 + K_{\text{PFD}}(K_p s + K_i)e^{-s\tau}K_{\text{DCO}}} \\ &= \frac{M(K_p s + K_i)e^{-s\tau}}{s^2 + M(K_p s + K_i)e^{-s\tau}} \end{aligned} \tag{7.2}$$

where $M = K_{\text{PFD}}K_{\text{DCO}}$.

The ADPLL receives the phase of input signal ϕ_{input} The phase comparison part consisting of four PFDs compares ϕ_{input} with the phase of output clock signalϕ_{output}, and generates the average value of phase errors (Fig. 7.1(b)). The input of feedback loop ϕ_{input} for the node (x,y) can be regarded as the mean value of the phase of the four neighboring clocks $(\phi_{x+1,y} + \phi_{x-1,y} + \phi_{x,y+1} + \phi_{x,y-1})/4$. The output ϕ_{output} is the phase of local oscillator $\phi_{x,y}$. Each node in the network satisfies the following equation:

$$\begin{aligned} \phi_{\text{output}} &= \phi_{\text{input}} \cdot H \\ \phi_{x,y} &= \frac{\phi_{x+1,y} + \phi_{x-1,y} + \phi_{x,y+1} + \phi_{x,y-1}}{4} \cdot H \end{aligned} \tag{7.3}$$

The sum of phase error is:

$$\begin{aligned}\Sigma_{\text{error}} =&(\phi_{x+1,y}-\phi_{x,y})+(\phi_{x-1,y}-\phi_{x,y})+\\ &(\phi_{x,y+1}-\phi_{x,y})+(\phi_{x,y-1}-\phi_{x,y})\\ =&\phi_{x+1,y}+\phi_{x-1,y}+\phi_{x,y+1}+\phi_{x,y-1}-4\phi_{x,y}\end{aligned} \tag{7.4}$$

Thus Eq. (7.3) can be rewritten as

$$\phi_{x,y} = (\frac{\Sigma_{\text{error}}}{4}+\phi_{x,y})\cdot H \tag{7.5}$$

Eq. (7.4) coincides with the discretization (cf. Eq. (7.7)) of the Laplacian (cf. Eq. (7.6)) of a scalar field Φ in 2-dimensional continuous space.

$$\Delta\Phi_{x,y} = \frac{\partial^2\Phi}{\partial x^2}+\frac{\partial^2\Phi}{\partial y^2} \qquad \text{(Definition of Laplacian)} \tag{7.6}$$

$$\begin{aligned}\Delta\Phi_{x,y} \simeq&(\phi_{x+1,y}-\phi_{x,y})-(\phi_{x,y}-\phi_{x-1,y})+\\ &(\phi_{x,y+1}-\phi_{x,y})-(\phi_{x,y}-\phi_{x,y-1}) \qquad \text{(Discretization of Laplacian)}\\ =&\phi_{x+1,y}+\phi_{x-1,y}+\phi_{x,y+1}+\phi_{x,y-1}-4\phi_{x,y}\end{aligned} \tag{7.7}$$

Hence, we can perform a "reverse discretization" process, and describe the discrete network in continuous medium. Eq. (7.5) becomes

$$\begin{aligned}\Phi_{x,y} &= (\frac{\Delta\Phi_{x,y}}{4}+\Phi_{x,y})\cdot H\\ &= (\frac{\Delta\Phi_{x,y}}{4}+\Phi_{x,y})\cdot\frac{M(K_ps+K_i)e^{-s\tau}}{s^2+M(K_ps+K_i)e^{-s\tau}}\end{aligned} \tag{7.8}$$

By performing a reverse Laplace transform on Eq. (7.8), we arrive at the following differential equation:

$$\frac{\partial^2\Phi}{\partial t^2} = \frac{MK_p}{4}\frac{\partial\Delta\Phi(t-\tau)}{\partial t}+\frac{MK_i}{4}\Delta\Phi(t-\tau) \tag{7.9}$$

To simplify the equation, we make two approximations. First, during a very small variation of time t, $\Delta\Phi(t-\tau)\simeq\Delta\Phi(t)-\frac{\partial\Delta\Phi(t)}{\partial t}\cdot\tau$; Second, a variation of local clock phase Φ introduces a change of phase differences $\Delta\Phi$ with its neighboring nodes clocks. Since the

loop filter processes the mean value of four phase errors, according to Eq. (7.7), during a small variation of time, $\frac{\partial \Delta\Phi}{\partial t} \simeq -4\frac{\partial \Phi}{\partial t}$.

$$\begin{aligned} \frac{\partial^2\Phi}{\partial t^2} &\simeq \frac{MK_p}{4}\left(\frac{\partial\Delta\Phi}{\partial t} - \frac{\partial^2\Delta\Phi}{\partial t^2}\tau\right) + \frac{MK_i}{4}\left(\Delta\Phi - \frac{\partial\Delta\Phi}{\partial t}\tau\right) \quad -\text{first approximation} \\ &\simeq -(MK_p - MK_i\tau)\frac{\partial\Phi}{\partial t} + MK_p\tau\frac{\partial^2\Phi}{\partial t^2} + MK_i\Delta\Phi \quad -\text{second approximation} \end{aligned} \tag{7.10}$$

Hence, Eq. (7.9) can be approximated as:

$$\frac{\partial^2\Phi}{\partial t^2} \simeq -\frac{M(K_p - K_i\tau)}{1 - MK_p\tau}\frac{\partial\Phi}{\partial t} + \frac{MK_i}{4(1 - MK_p\tau)}\Delta\Phi \tag{7.11}$$

7.2.2 An analogy with damped wave equation

Up here, we have performed a reverse discretization passage from ADPLL mesh ϕ to a continuous surface of phase errors Φ. Eq. (7.11) is the same as the damped wave equation describing water surface movement with dissipation [64]:

$$\frac{\partial^2 h}{\partial t^2} = -k\frac{\partial h}{\partial t} + c^2\Delta h \tag{7.12}$$

Here h is the height of the water, c is the wave speed and k is the damping constant. We can make an analogy between the level of water and the phase error of synchronous network. By comparing Eq. (7.11) and Eq. (7.12) we get the k and c parameters of the synchronization errors surface.

$$k = \frac{M(K_p - K_i\tau)}{1 - MK_p\tau},\ c = \sqrt{\frac{MK_i}{4(1 - MK_p\tau)}} \tag{7.13}$$

The transient process of an unlimited ADPLL network in phase domain can be seen as analogous to the wave movement in a vast expanse of water. In equilibrium, the whole water surface is flat. Similarly, when the ADPLL network is synchronized, all the locally generated clocks are in phase. However, if there is a local perturbation, a wave may appear, propagating through the network. This is an undesirable phenomena. The solution aimed to limit it is proposed in the next section.

7.3 ADPLL network with limited surface

When an infinite network is synchronized in phase, in a local micro region, the difference of phase level between the nodes (x+1,y) and (x,y) approximates the inverse of phase difference between nodes (x-1,y) and (x,y), thus the Laplacian of local phase approaches zero, which means the local region $\langle \phi_{x+1,y}, \phi_{x,y+1}, \phi_{x-1,y}, \phi_{x,y-1} \rangle$ can be regarded as flat.

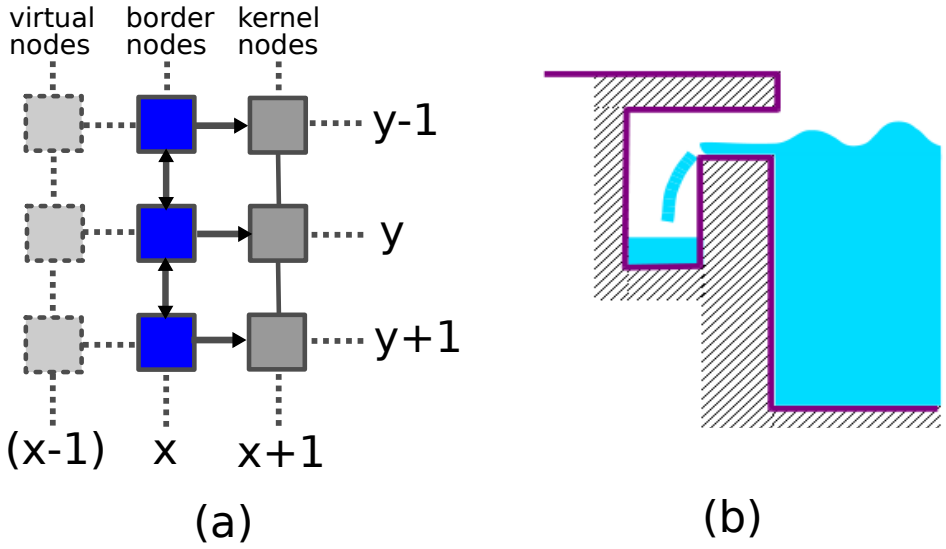


Figure 7.2: **(a) Interconnection of nodes at the border of network**: ($\rightarrow$: unidirectional; $\leftrightarrow$, $-$: bidirectional) **(b) overflow channel of a swimming pool**

However, on the boundary of a limited network as in Fig. 7.2(a), since a node (x,y) on the left border of the network has no neighboring node on its left, $\phi_{x-1,y}$ does not exist, the local phase region is not flat in x-direction. The phase difference $\phi_{x+1,y} - \phi_{x,y}$ makes node (x,y) changes its frequency, which will result in a fluctuation of phase in this local region, including the node (x+1, y) itself. In other words, the error wave reflects on the border. As explained previously, to have a flat surface of phase and suppress the reflection, we need an infinite network. To have such a network, we need additional virtual nodes beyond the border to balance the phase in x-direction, as shown in Fig. 7.2(a). Since node (x-1,y) doesn't exist, it is enough to eliminate the link from node (x+1,y) to the border node (x,y) and keep the link from the border to the kernel.

These anti-reflection considerations lead to isolate a border only distributing its clock to a kernel surface in which the nodes are connected as in the case of unlimited network (Fig. 7.3).

All ADPLLs (border and kernel) are used to generate local clocks. The network border can be regarded as an independent and synchronous ring exciting the inner kernel as a membrane. This ring, with the reference clock signals at its 4 corners, produces a reference for ADPLLs in the kernel and absorbs the error waves. In the "Swimming pool"-like analogy, the ring of the ADPL network acts as the overflow channels of a pool (cf. Fig. 7.2(b)).

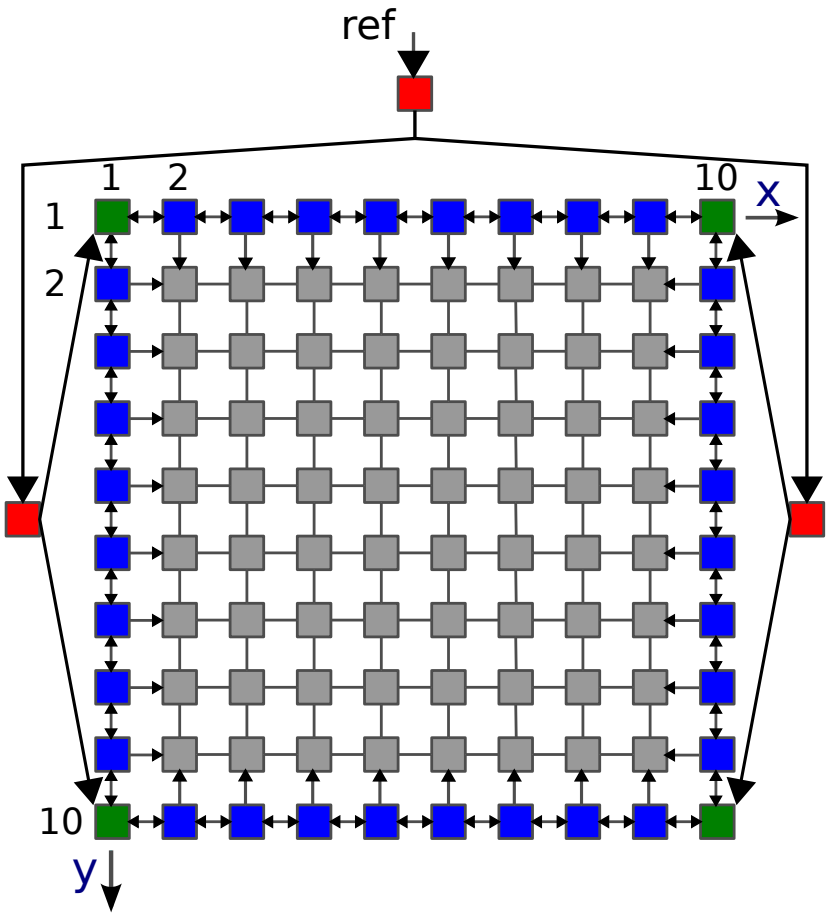


Figure 7.3: **Proposed network topology** ($\rightarrow$: unidirectional; $\leftrightarrow$, $-$: bidirectional)

7.4 Simulation results

A 10×10 network as shown in Fig. 7.3 is modeled in VHDL. A PFD with a resolution of 20 ps and a DCO with a nominal frequency of 1 GHz and mean frequency step of 2.26 MHz are used in this work. The simulations allow studying the behavior of network with different parameters and validating the theoretical analysis.

To observe the process of synchronization and phase error attenuation, phase errors of local clocks with respect to the reference clock are sampled each cycle and used to create a 3-D animation. The four images in Fig. 7.4 show the surface of phase error in the network at four different moments, giving us a direct idea of the synchronization process.

Fig. 7.4(a) shows the the phase errors before the network is synchronized in phase. The network works in the frequency/phase acquisition mode. We observe that the border is very stable with a relatively low amplitude of errors, while the kernel acts like a membrane fluctuating up and down with an amplitude smaller and smaller until the whole network gets in phase.

When the network is synchronized in phase, the surface is flat. We add a perturbation on one node in the network as shown in Fig. 7.4(b). The phase error propagates to the border like a wave(cf. Fig. 7.4(c)) . Since the border is isolated from the kernel, instead of affected by the error wave, it attenuates the amplitude of wave until the surface returns to a flat level (cf. Fig. 7.4(d)). We can say the error is absorbed by the border like the water flows into the overflow channel of a pool.

When the whole network is in phase, the phase difference between two neighboring clocks is within ±40 ps, which is two steps of PFD resolution. We measure the phase error of each clock with respect to the reference clock REF so as to obtain the clock error distribution histogram of nodes in the kernel (Fig. 7.5(a)) and that of nodes in the ring border (Fig. 7.5(b)). It is obvious that the border clocks have smaller errors than kernel clocks, which agrees with our previous analysis.

A significant advantage of the proposed circuit is its good performance of perturbation attenuation. To prove it, we compare the proposed architecture shown in Fig. 7.3 with a conventional 10×10 fully connected topology without ring. When the network is in phase, we add an artificial perturbation on CLK35 at the node (3,5), and observe the transient response on nodes (3,5), (2,5) and the nearest border node (1,5).

In the conventional circuit, it is obvious that CLK15 is affected by the perturbation put on CLK35. The reflection produces some wavelet on node (1,5) (Fig. 7.6(a)). In the proposed "Swimming pool"-like topology, we can observe that thanks to the strong ring border, CLK15 is not affected and there is no more wavelet on CLK35 (Fig. 7.6(b)).

According to Eq. (7.13)), the wave speed and damping constant depend on design pa-

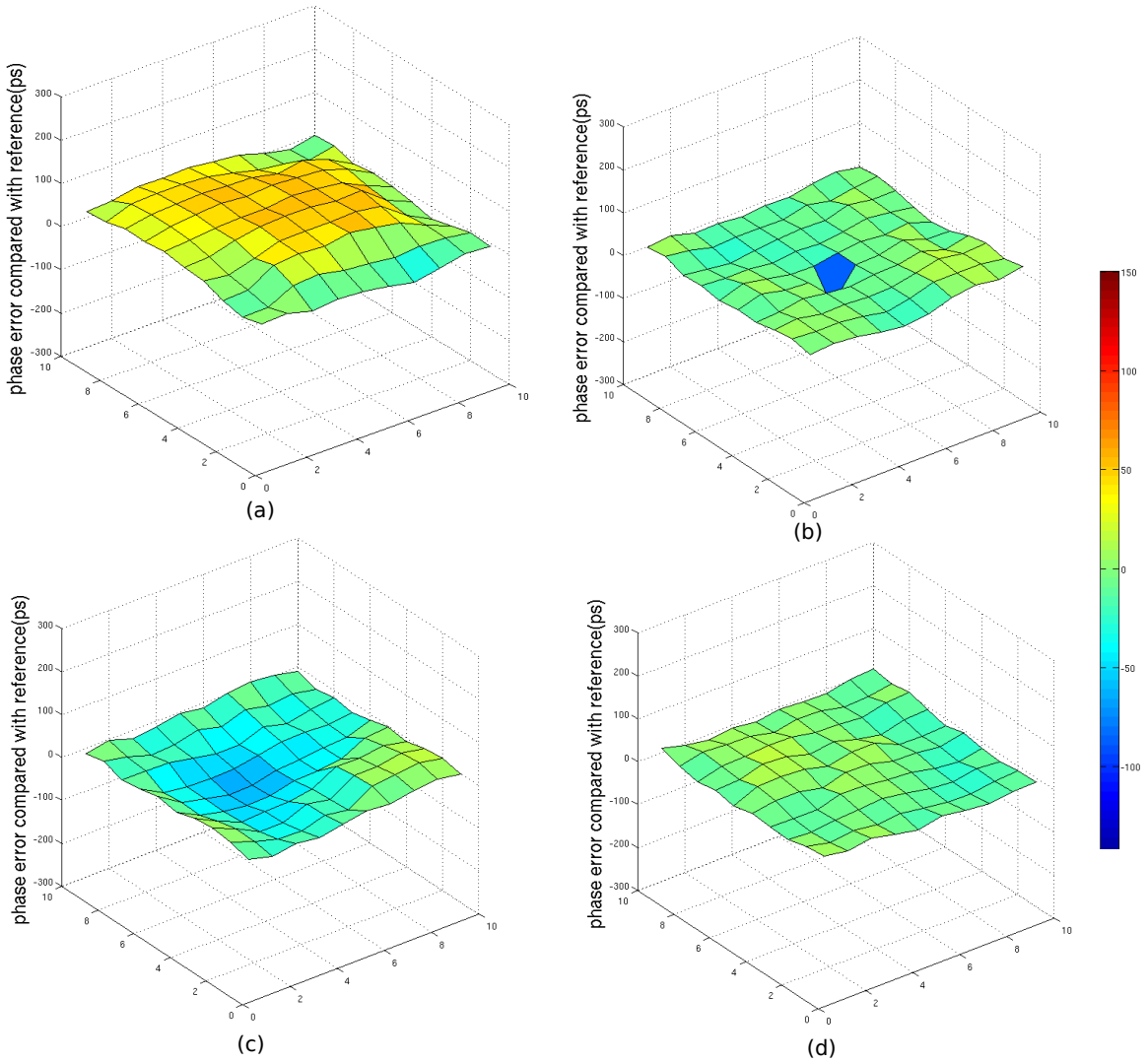


Figure 7.4: **The surface of phase error between each local clock signal and reference:** (a) in frequency/phase acquisition mode; (b) a perturbation happened; (c) the propagation of phase error; (d) the stable state is re-established

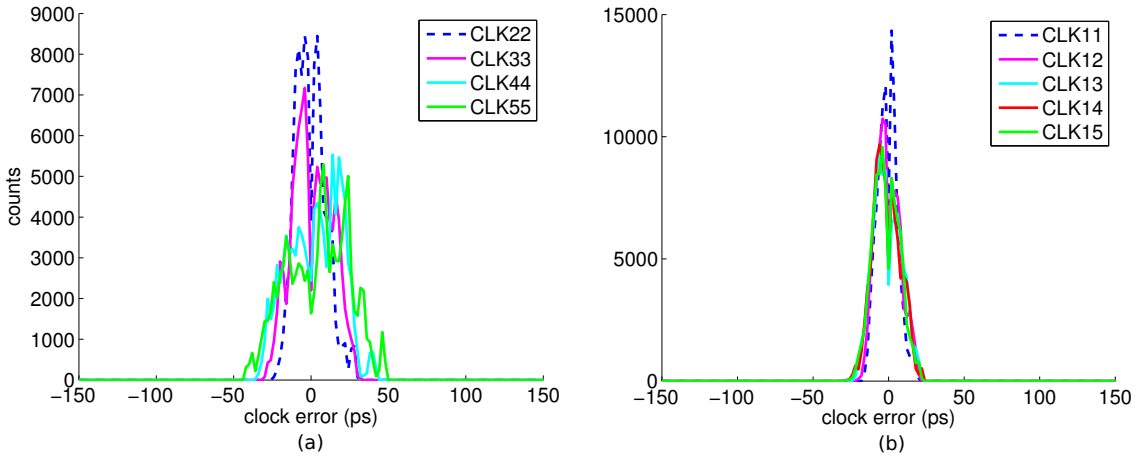


Figure 7.5: **Histogram of absolute phase errors at steady state:** (a) inner clock signals; (b) border clock signals

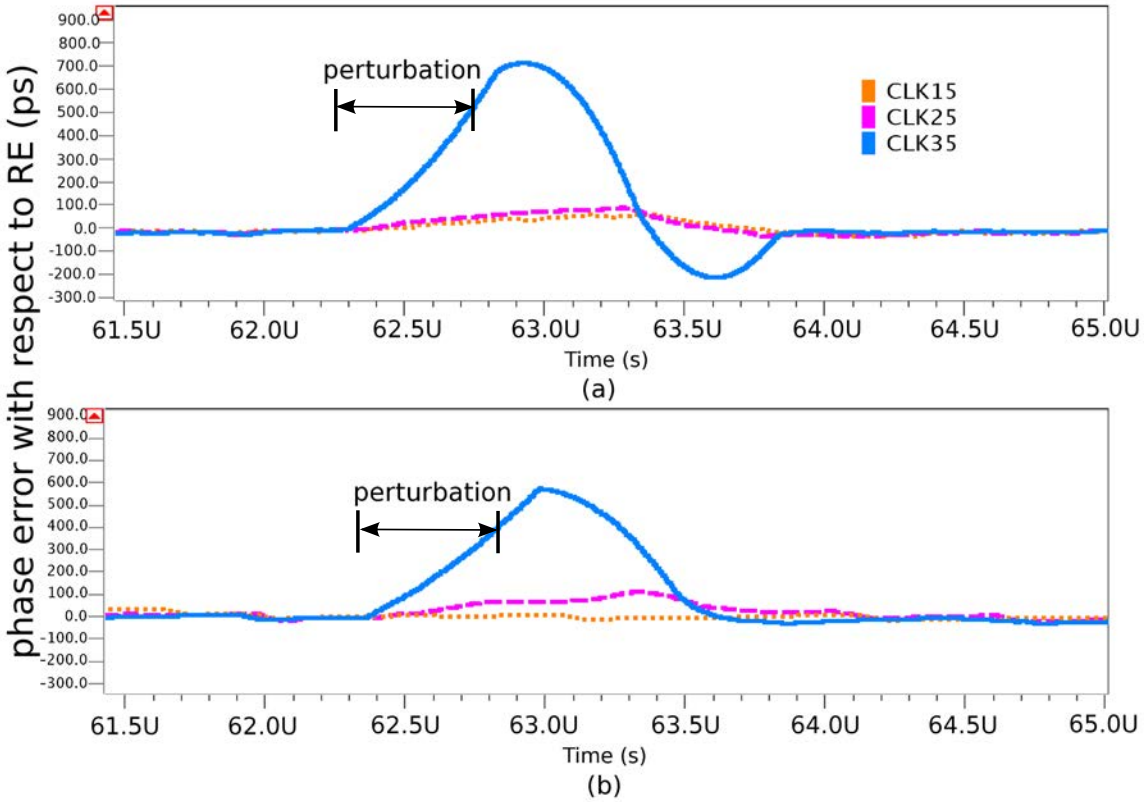


Figure 7.6: **Transient response of perturbation:** (a) conventional 10×10 network (b) "Swimming pool"-like 10×10 network

rameters (gains of PFD and DCO, filter coefficients, etc.). The reconfigurability of loop filter allows modifying features of the control system according to the specification[21]. Fig. 7.7 shows phase errors of clock signals in the principal diagonal of the proposed network with two different parameter sets. We can observe that if the system is overdamped, the system takes a shorter time to acquire the reference frequency, but the phase error in steady state is relatively larger (± 100 ps in Fig. 7.7(a)) compared to an underdamped system (± 50 ps in Fig. 7.7(b)). Designers can choose the appropriate parameters to meet requirements of convergence speed and maximum error limit.

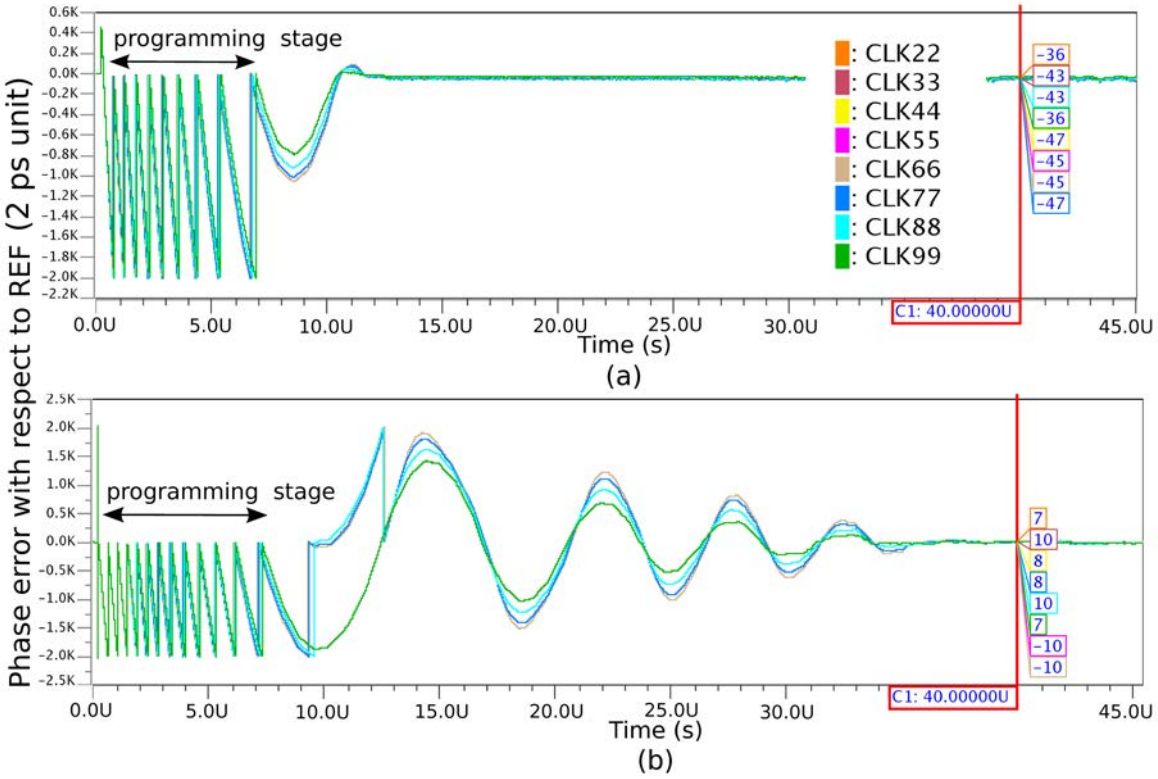


Figure 7.7: **Absolute phase errors of clock signals in proposed network with different parameters:** (a) overdamped; (b) underdamped

7.5 Conclusion

An ADPLL network with "Swimming pool"-like topology is proposed for synchronization in large many-core SoC. This topology is based on the architecture of coupled ADPLL network. However, this configuration enhances the stability of system by improving its capacity of perturbation attenuation. Theory analysis presented in this chapter supports design of the proposed architecture for different specifications. Simulation results proved the feasibility of a fully synchronized circuit with a large size. This possibility allows the use of large globally synchronous architecture, so offering a possibility of designing and verifying SoCs with well-established and secure design methods.

Thanks to the reconfigurability of ADPLL, we can realise this topology on the circuit presented in Chapter 6. The work presented in this chapter may serve as an example of innovative topology exploration by using the coupled ADPLL network as a research and development platform. Other topologies may be inspired for specific needs or improvement.

Chapter 8

ADPLL with sliding-window for wide range frequency tracking

Contents

8.1 Introduction 161
8.2 State of art 163
8.3 "Sliding window" architecture 166
8.4 Comparison with conventional PLL 174
8.5 Clock distribution network using "sliding window" ADPLL 176
8.6 Conclusion 182

8.1 Introduction

This chapter presents a study aiming to improve the dynamic and power performance of the digital PI filter of the ADPLL. In particular, the goal of the study is to reduce the time of the frequency acquisition of the ADPLL network, in comparison with the conventional architecture of the PI filter presented in Chapter 1.

This study was carried out at an early stage of the PhD, and the resulting solution for the architecture of the filter was a candidate for use in the silicon prototype of the ADPLL network. However, the proposed filter architecture was quite new, additional parameters have been introduced by new functional blocks in the proposed structure, which increases the degree of freedom of the non-linear system. When this solution is used in a coupled network, the non-linearity issue is more complex. This makes the solution risky to be used in the 10x10 network of ADPLL operating at high frequencies. Hence, in the prototype, we preferred to use the conventional architecture of filter presented in Chapter 3.

However, the study of the alternative solution for the filer was an important part of the PhD project, the results were very satisfying, and the solution was validated at different

levels. For this reason, we include the presentation of this study in this PhD report.

The large frequency acquisition time is related to the limited output range of the PFD: when the phase error is very large (the frequency are different), the PFD outputs the maximal value allowed by the capacity of the output digital word (± 15 in the first prototype presented in Chapter 1 and ± 7 for the second prototype presented in Chapter 6). For this reason, the frequency acquisition time is roughly proportional to the difference between the reference frequency (the input) and the divided DCO frequency. The acquisition time may be very high if the DCO frequency range is large.

The response to this problem is a different processing of frequency acquisition and phase acquisition. We implemented it with an architecture which we called “PI filter with sliding window”, which is described in this chapter. The chapter is organized as follows: in Section 8.2, previous work aiming at the same targets are described, and their drawbacks are indicated. Section 8.3 explains in detail the principle of proposed architecture and functionality of each block. Then Section 8.4 presents simulation and synthesis results of both conventional and proposed circuit, which demonstrate high performance of the new architecture. At last in Section 8.5, the "sliding window" ADPLL is implemented in a 4×4 network. Verification with non-ideal cases consideration is performed. The power consumption of a network using the proposed technique is compared with a network composed of conventional ADPLLs.

8.2 State of art

As presented in previous chapters, the structure of a typical PLL is presented in Fig. 8.1 [73]. A PFD detects the phase/frequency difference between the locally-generated clock and a reference clock, and generates a signed binary code. This code is then processed by a loop filter, so as to generate a control word for the Digitally Controlled Oscillator (DCO). The frequency divider is used to generate a clock with a frequency higher than the one at which the error phase information is processed.

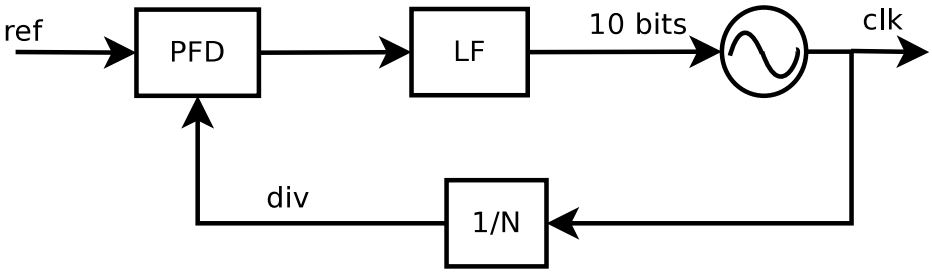


Figure 8.1: **Conventional PLL structure**

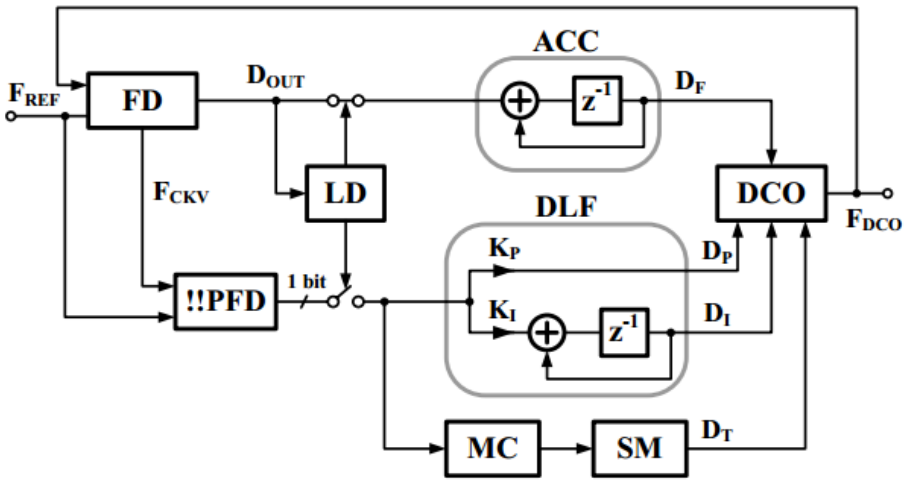


Figure 8.2: **Tierno PLL structure[30]**

A traditional ADPLL with PI filter has two disadvantages in the case where the DCO frequency range is wide, as required for the modern clocking and RF systems. One is long frequency acquisition time, because a single PFD is not efficient enough to give the frequency relation between two clocks under comparison. The other is high power consumption of high-frequency multi-bit arithmetic operations related to the PI filter processing. The second disadvantage is more serious in our case. Because coefficients in the loop filter are programmable, to achieve a calculation with precision, large number of bits are reserved for coefficients, thus the arithmetic operators (adder, multiplier, integrator...) cost much area and power. Moreover, in steady state, since the PLL already converges, normally only one or two LSBs are updated each cycle. It is not necessary to calculate the whole control word at high rate.

Fast lock can be achieved using a dual-loop architecture: one loop for frequency tracking, the other for phase error correction, with different filter characteristics. The block diagram of a DPLL architecture proposed in [30] is shown in Fig. 8.2. It consists of two loops: an FLL (Frequency Locked Loop) and a PLL. On power-up only the FLL is active. The frequency detector (FD) determines the frequency difference in digital format (D_{OUT}) and provides it to the accumulator (ACC). The output of ACC controls the frequency of a digitally controlled oscillator (DCO) via a 10-bit current mode digital-to-analog converter (DAC), which allows for a wide tuning range. The lock detector (LD) detects when the frequency lock is acquired, and then deactivates the FLL and activates the PLL. The sign of the phase-frequency difference in the PLL is determined by a bang-bang phase-frequency detector (!!PFD) that consists of a conventional PFD followed by a sampling D flip-flop. The one-bit digital output of the bang-bang PFD is filtered by a digital PI loop filter (DLF). The proportional path drives the DCO directly through a high-speed path consisting of only a 1-bit DAC (PDAC). In order to improve the tracking range of the DPLL, a phase-frequency error monitoring circuit (MC) and a state machine (SM) have been incorporated in parallel with the DLF.

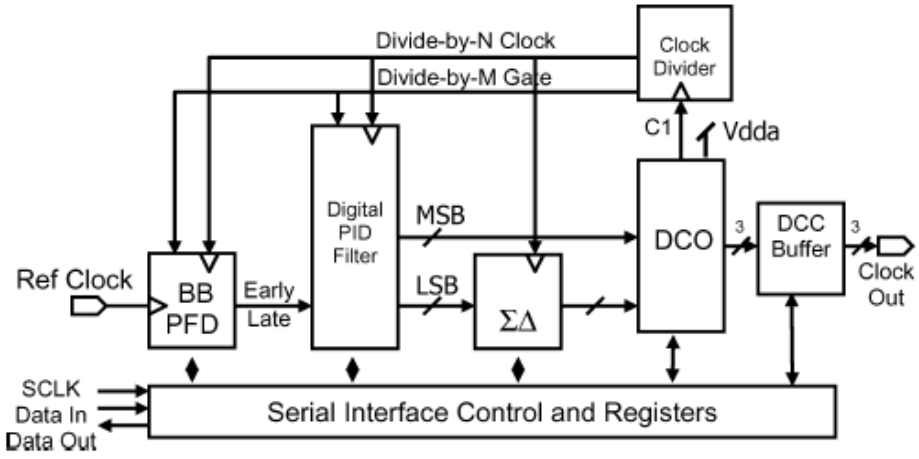


Figure 8.3: **Tierno PLL structure[60]**

As we can see, this approach is very expensive in term of hardware resources: in addition to common PLL blocks (PFD, DLF and DCO), other blocks like FD, ACC, MC PDAC and SM are needed. This may result in a potential increase of circuit area and power consumption. For certain applications, for instance, a clocking network using many ADPLLs, this approach is not appropriate.

A simpler solution is proposed by J.A. Tierno [60]. This method divides DCO dynamic range into several segments: the most significant bits (MSB) for frequency tracking (at low rate), which are sent directly to the digitally controlled oscillator (DCO), and the least significant bits (LSB) for phase tracking (at high rate), which are sent to a third-order sigma-delta modulator and are used to enhance the frequency resolution of the DCO. However, a problem of this approach is that boundaries of segments are predefined by the lengths of MSBs and LSBs. If the code corresponding to the reference frequency happens to be at the boundary

between two segments, MSBs value will hop between two adjacent code, the MSBs must change with high frequency in order to achieve the phase tracking.

A new method with floating frequency segment is proposed in this study to solve this problem. As in some previous works [60], actual DCO control code is obtained as sum of a large coarse value and a small signed correction code. Generally, the coarse code represents the bits with high weight, and the correction (fine) code represents the bits with small weight. However, in the proposed architecture, the LSB of the coarse code and correction code have the same weight. Coarse frequency code is dynamically updated at a relatively slow frequency. A PFD and a filter calculate a signed correction code, which is periodically added to the coarse frequency code. It allows fast updating of DCO control word within a certain range around the coarse frequency. This structure implements a 4 bit range frequency window slowly sliding on 10 bits range. An update of the coarse frequency code provides an immunity to slow variations of the DCO initial frequency due to temperature, etc. This method provides a substantial economy of high frequency arithmetic operations, which results in a power saving.

8.3 "Sliding window" architecture

The architecture of proposed PLL is shown in Fig. 8.4. A Reference Frequency Indicator (RFI) block, apart from the main PLL circuit, gives a code ($code_{ref}$) corresponding to the reference clock frequency. This code is then sent to PLLs in clock distribution network. Phase tracking is achieved by a 3-bit PFD and a 4-bit filter generating a 4-bit signed code based on the phase difference between two clocks. Frequency tracking is achieved by a mean filter, which receives a stream of 4-bit codes and calculates the average of eight most recently received ones. The sign of the average value ($+1$, -1, or 0) estimates the frequency relation between two clocks, and is used by the Coarse Frequency Adjustment (*CFA*) block for coarse frequency code adjustment. The output code of *CFA* block plus the 4-bits filter output forms DCO control word.

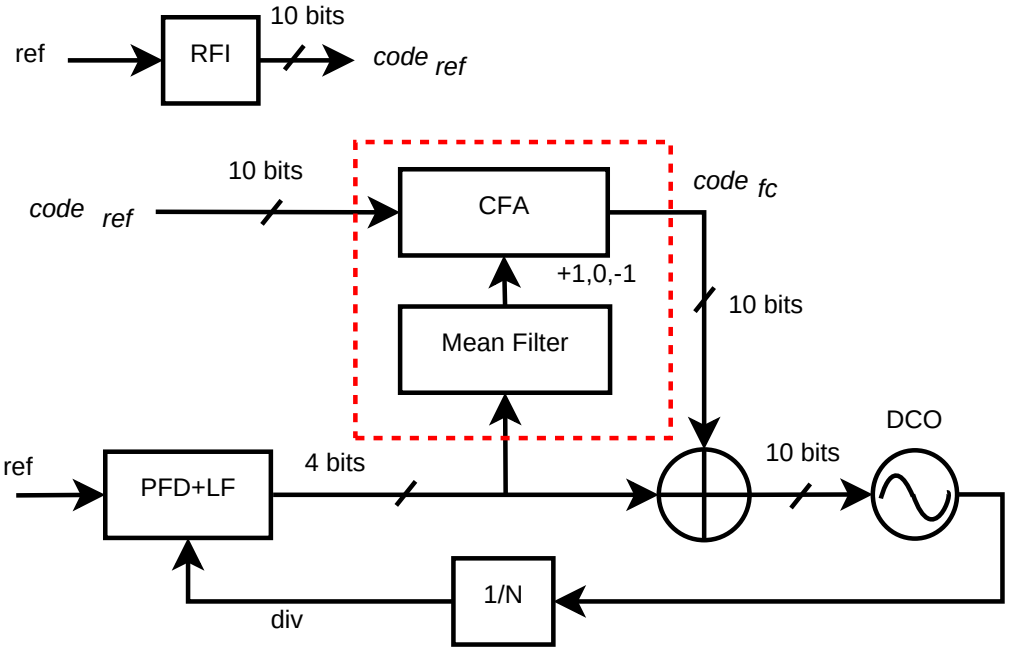


Figure 8.4: **Proposed architecture for PLL**

The advantage of this structure is that *CFA* block can always update the coarse frequency (fc) of DCO according to the reference-local clock relation, and this adjustment is performed at a frequency lower than sampling frequency of system (div). PFD and filter, working at frequency div, tune DCO frequency around fc. This tuning process corrects phase error and at the same time works together with *CFA* to make sure that the reference frequency is always in the tuning interval. As the interval is relatively small, a big number of bits for PFD and filter is no longer necessary.

8.3.1 Reference frequency indicator (RFI)

A RFI is an extra block for one PLL or a clock distribution network. It gives the system a 10-bits coarse estimation code of the reference frequency ($code_{ref}$) at starting stage. A RFI block can be implemented as a Look-Up Table (LUT), which takes reference frequency and

some other parameters like temperature as inputs and gives corresponding code with certain precision at a fast speed. The implementation of this block is not an emphasis of this work.

8.3.2 Coarse frequency adjustment

The coarse frequency acquisition is achieved by the mean filter block and *CFA* block (red rectangle in Fig. 8.4).

Mean filter

A mean filter uses a FIR (Finite Impulse Response) architecture which calculates the average value of last 8 values at the input. The equation of a mean filter with input $x[i]$ at a given moment i is:

$$sum[i] = x[i-7]+x[i-6]+\cdots+x[i] \tag{8.1}$$

$$avg[i] = sum[i]/8 \tag{8.2}$$

where $sum[i]$ and $avg[i]$ are sum value and average value. If we define a signal $new[i]$ for the new input data and another signal $old[i]$ for the oldest data value at that moment, $sum[i]$ can also be represented by Eq. (8.3).

$$sum[i] = sum[i-1]-old[i]+new[i] \tag{8.3}$$

There are two options to design such a filter: shift register and circular buffer, whose principles are presented in Fig. 8.5.

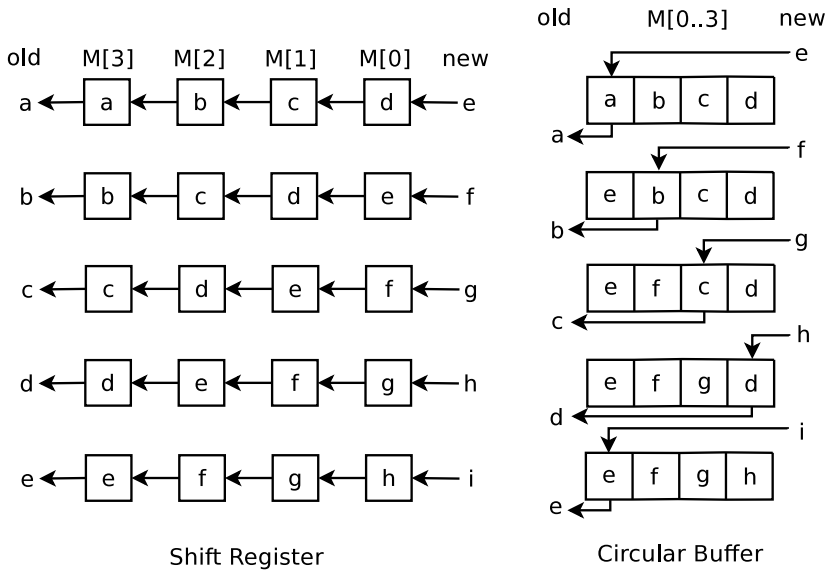


Figure 8.5: **Sliding window algorithm**

A shift register stores the last N data dynamically. At each clock event, all the data are left shifted. The data in the leftest register, which is the oldest, is popped out. And a new

data is pushed in the rightest register. The block always calculates the sum and average value of data in all the N registers.

A circular buffer [40] is a memory storing the last N output data of filter. An index points out register which stores the oldest data. The data stored in the register pointed by the index is substituted with the new input. Instead of shifting the data, the index is shifted at each clock event. In this case, only one register need to be updated, thus a circular buffer requires less power than a shift register.

For this reason, circular buffer is chosen to implement the mean filter. To avoid read-write conflicts, the mean filter works in two phases, one phase (S1) for reading the oldest data and subtracting it from *sum*, and the other phase (S0) for writing new data and adding it to last calculation result. Transitions of phases are sensitive to edges of *clk* signal, as shown in Fig. 8.7. A controller, implemented as a finite state machine (FSM), verifies transition conditions of working state, generates the write enable signal (WE) and new index for the circular buffer (idx), updates the input for ALU ($IN1$), and calculates the average value (avg) of *sum* and its sign *avg_sign*. The block diagram of the whole mean filter is presented in Fig. 8.6. Tab. 8.1 details definition and output signal values of two different working states.

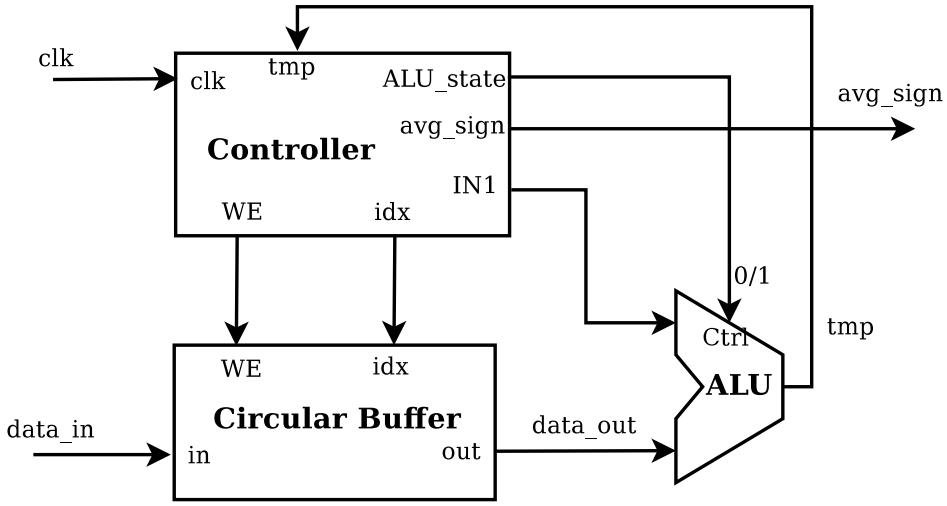


Figure 8.6: **Mean filter implementation**

Table 8.1: **Control table**

clk	**state**	idx_n	**WE**	**ALU_state**
1	S1	$(idx_{n-1}+1) \bmod N$	0	1 (mode $\ominus$)
0	S0	idx_n	1	0 (mode $\oplus$)

The circular buffer storing last N output data of filter is implemented as an array of N registers (N=8 here) with a N-to-1 multiplexer and a 1-to-N demultiplexer. In writing mode, the value of *idx* signal defines which clock signal flips, and hence, which register stores the new *data_in* value. The data update of each register is done only at the rising edge of its

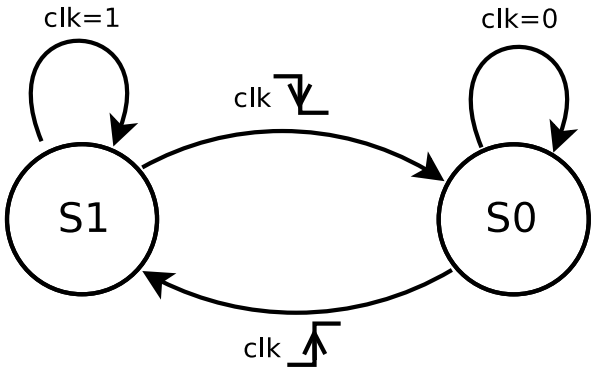


Figure 8.7: **State diagram of controller**

clock signal. In reading mode, the output always takes value of the register selected by *idx*. As shown in the chronograph (Fig. 8.9), *data_in* and *idx* are always prepared half *clk* period before the rising edge of WE.

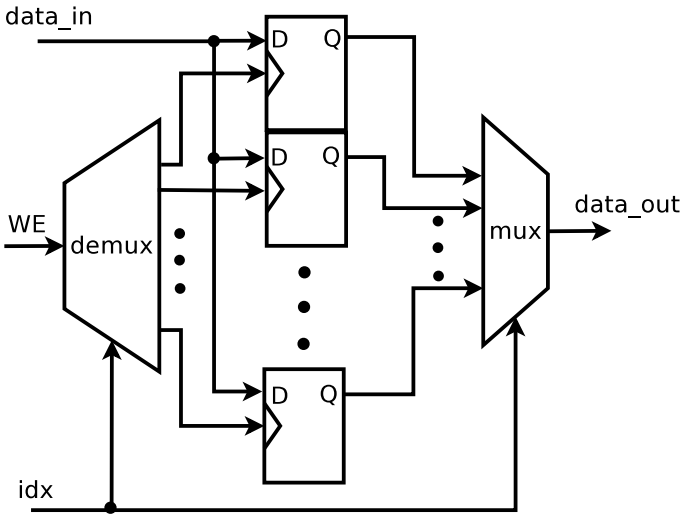


Figure 8.8: **Implementation of memory in mean filter**

This operation of mean filter can be described by the data flow diagram in Fig. 8.10. In general, a full cycle starts with S1 and ends with S0. Before S1, *tmp* stores the sum value of last cycle. *idx* points to the oldest data in last cycle. At the beginning of S1, the average value of last cycle and its sign *avg_sign* are calculated by operation *avg*, which is applied in the controller because it is as simple as a right binary shift. The *idx* is also shifted to point to the oldest data in the current cycle. In the mean time, the controller takes the temporary result *tmp* (calculated by ALU), and prepares one input of ALU $IN1$ for its next operation. The other input of ALU is always the output data of memory (*data_out*) pointed by its current *idx*. However, the value is different in different states:

$$data\_out_{(s1)}[i] = old[i] \tag{8.4}$$

$$data\_out_{(s0)}[i] = new[i] \tag{8.5}$$

Block ALU is an adder/subtracter. Its working mode depends on *ctrl* signal value. The output of ALU *tmp* is either the intermediate calculation result (at state S1) or a new *sum* value (at state S0):

$$tmp_{(s1)}[i] = sum[i-1] - old[i] \quad (8.6)$$

$$tmp_{(s0)}[i] = tmp_{(s1)}[i] + new[i] \quad (8.7)$$

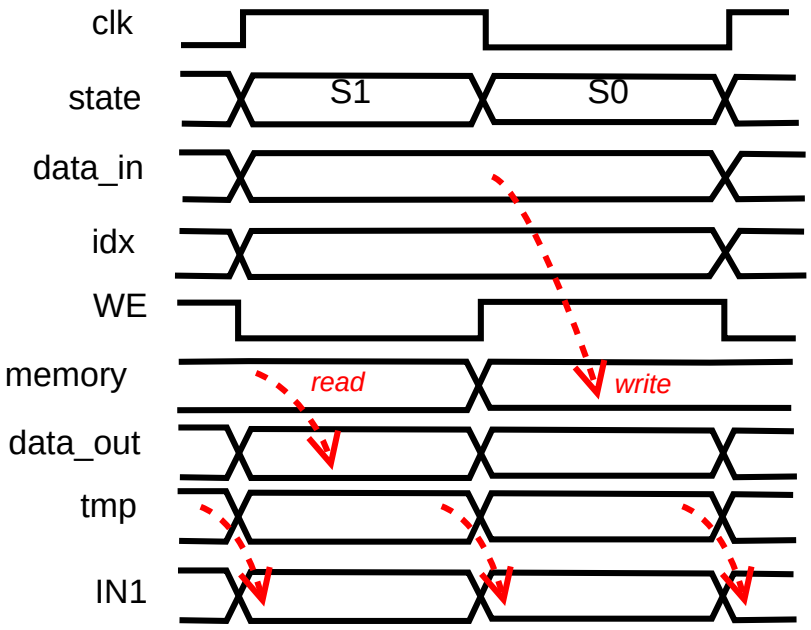


Figure 8.9: **Chronograph of mean filter**

Some timing constraints have to be met for correct functionality of circuit. First, update of signal *idx* and subtraction operation should be carried out within the first state *S*1. Second, new income data is stored in proper register and an addition gives the new *sum* value, which should be finished before the end of the second state *S*0. Each state lasts half period of *clk* signal.

To verify the timing conditions, synthesis is done using Synopsys Design Compiler under a CMOS 65nm technology. Timing report shows that the work of each state is finished within 1 ns under a timing constraint of 2.5 ns for clock period. There is a positive slack of 250 ps in each state. Timing constraints are satisfied.

Coarse Frequency Adjustment block (CFA)

The role of the *CFA* block is to calculate a coarse frequency code $code_{fc}$ based on the output of mean filter in the last few cycles. *CFA* updates the coarse frequency code by ± 1 or 0 at each cycle. To identify an eventual regular error on the coarse frequency, the mean filter needs to accumulate the output of the phase tracking blocks during several cycles. There is a trade-off between the coarse frequency tracking precision and hardware complexity. Apparently, to better estimate the frequency relation between reference clock and generated

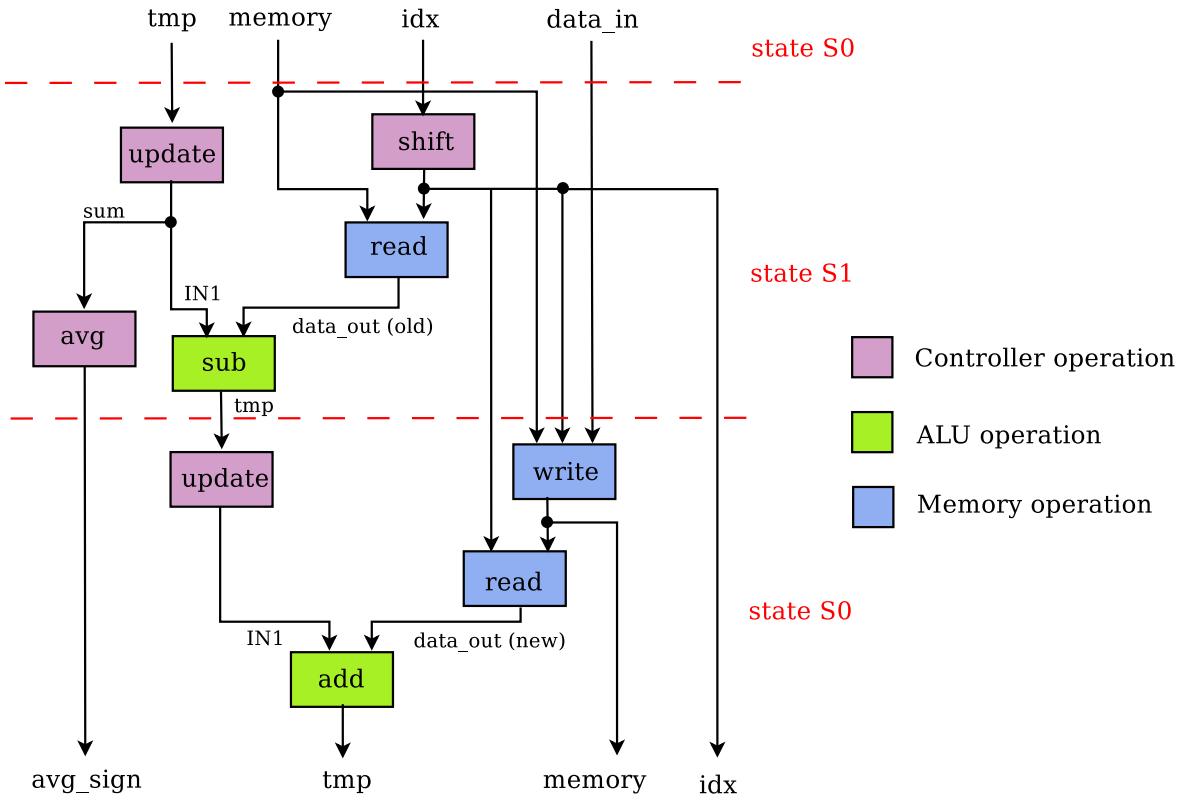


Figure 8.10: **Data flow diagram in the mean filter**

clock, an adequate number of cycles should be taken into consideration, which defines the size of shift register and ALU in Fig. 8.6.

To relax the constraint in mean filter, an integrator and a saturator are used in the CFA structure (cf. Fig. 8.11). The first integrator accumulates mean filter output value. If there is overflow or underflow, this integrator is reset to 0. The second integrator, which is initialized to $code_{ref}$, takes the overflow/underflow value to adjust coarse frequency progressively. The result is combined with phase error correction blocks presented in the next subsection Section 8.3.3 to perform a complete phase locking process.

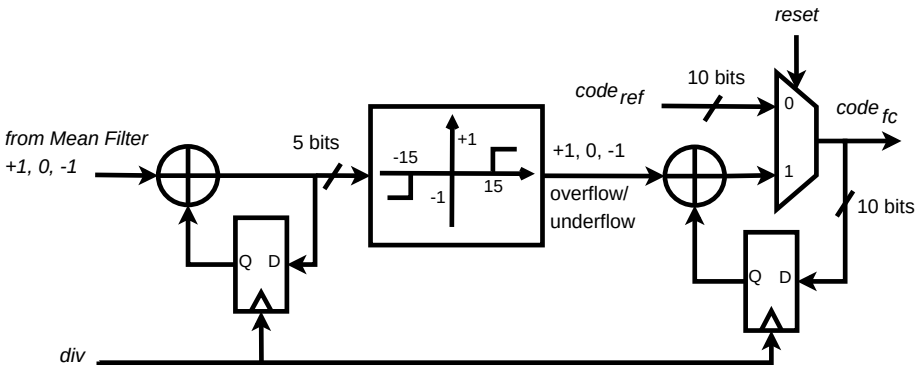


Figure 8.11: **CFA structure**

8.3.3 Phase error correction

Phase error is corrected by a 3-bit PFD and a 4-bit PI filter. To compare the circuit proposed in this chapter with a ADPLL with conventional structure, we use the same structures and principles as the PFD block presented in Section 3.1 and the filter in Section 3.2 except for less number of bits. The use of coarse frequency adjustment cells presented in the last section share the work of frequency tracking, making less bits for PFD and filter possible. However, the 3-bit PFD has a very small absolute phase comparison range, which results in some potential drawbacks.

Drawbacks related to a small PFD range

The PLL transfer function in s-domain is:

$$H(s) = \frac{K_{PFD}K_{DCO}(K_p s + K_i)}{s^2 + sK_{PFD}K_{DCO}K_p + K_{PFD}K_{DCO}K_i} \tag{8.8}$$

where K_{PFD} and K_{DCO} are the gain of PFD and the gain of DCO respectively. K_p and K_i are the proportional and integral coefficients of the PI filter.

If we compare Eq. (8.8) with the common transfer function of 2nd order system, the damping factor ξ is obtained.

$$\xi = \frac{K_p}{2}\sqrt{\frac{K_{PFD}K_{DCO}}{K_i}} \tag{8.9}$$

As shown in Fig. 8.12, due to the dynamic range limit of a 3-bit PFD, there is saturation when phase error is larger than $3\Delta T_{TDC}$ or less than $-3\Delta T_{TDC}$. Hence the gain K_{PFD} is much smaller in the saturation region than the one in quasi-linear region. According to Eq. (8.9), this results in a relatively small damping factor ξ in the saturation region, which causes a relatively slow correction speed at the beginning when the phase error is still large.

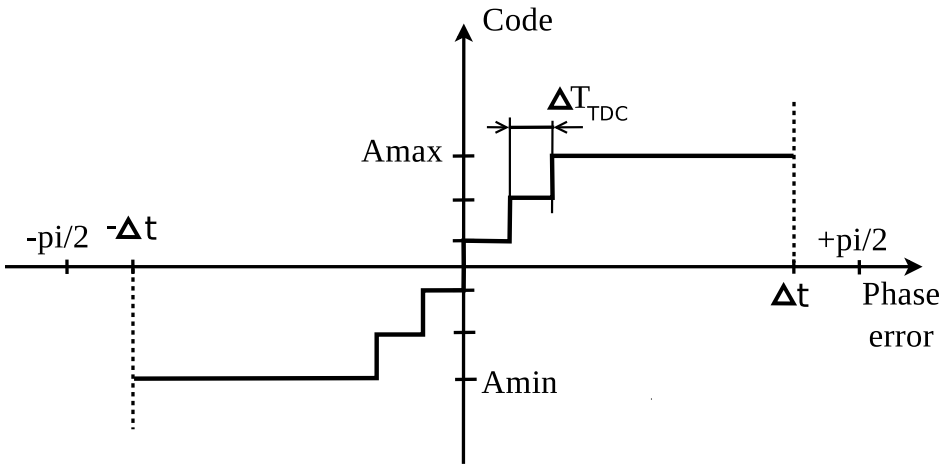


Figure 8.12: **Transfer function of 3-bits PFD**

Solution - Adaptive filter

According to Eq. (8.9), the damping factor ξ is function of K_{PFD}, K_{DCO}, K_p and K_i. Since K_{PFD} and K_{DCO} are defined by design specification, only the filter coefficients can be modified to compensate the effect of K_{PFD} diminution in saturation region. K_i is less effective than K_p, because K_i is under root and it has already a very small value in current design. If this value is reduced furthermore, the integral path performance is also reduced.

Hence, the solution consists in using an adaptive filter with a variable K_p (cf. Fig. 8.13) instead of a regular PI filter in order to change damping factor of system on the fly when PFD works in different regions [67]. In this implementation, we set the value of K_p in saturation region three times as large as that in quasi-linear region.

The key points of implementation consist in two mechanisms: detection of PFD working region and modification of the K_p for the case where the PFD is in the saturation. To detect the saturation at the PFD output, we just need to verify if the PFD output value equals to its minimum/maximum values (± 3). This costs just a few logic gates and tens of picoseconds.

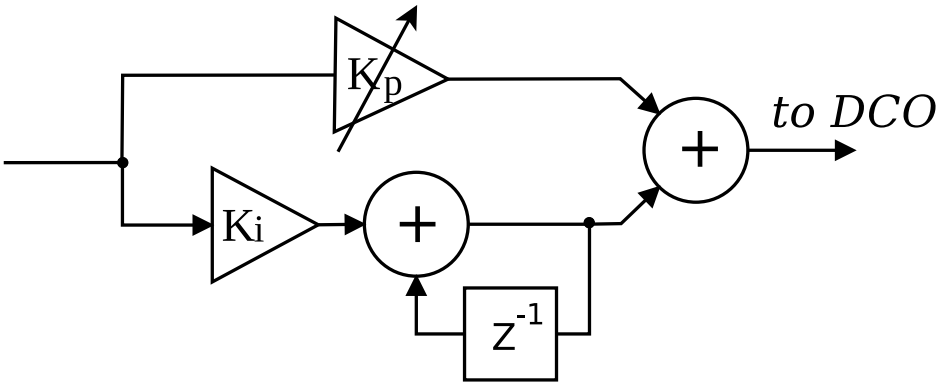


Figure 8.13: Adaptive loop filter structure

Table 8.2: **Parameter summary of two ADPLLs**

Parameter	*"sliding window" ADPLL*	*Conventional ADPLL*
PFD resolution	30 ps	30 ps
PFD number of bits	3 bits	5 bits
LF output number of bits	4 bits	10 bits
DCO Central frequency	1.744 GHz	1.744 GHz
DCO Output divided frequencies	872/218/109 MHz	872/218/109 MHz
DCO Tuning range	999∼2480 MHz	999∼2480 MHz
DCO Gain	1.482 MHz/LSB	1.482 MHz/LSB

8.4 Comparison with conventional PLL

In this section, we compare the proposed "sliding window" ADPLL with a conventional one presented in [73] in terms of functional performance and power consumption. Tab. 8.2 summarizes parameters of the two circuits under comparison.

8.4.1 Functional Simulation results

Fig. 8.14 shows the phase error between the reference clock and clock generated by an ADPLL using a conventional PI digital filter, as in sec. XXX [22]. The reference frequency is 297.3 MHz at first, and it changes to 225 MHz since 10 us. It takes 13.5 us for PLL to be re-synchronize with the new frequency.

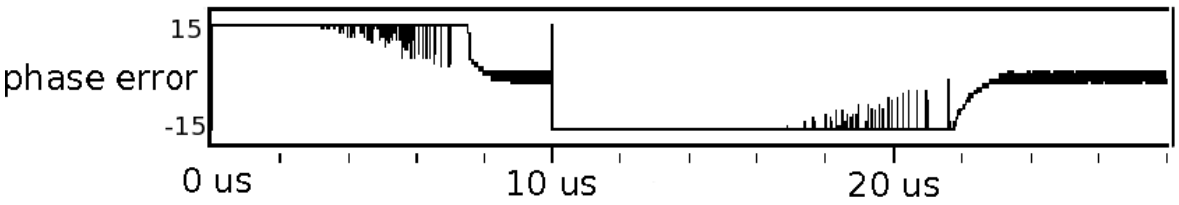


Figure 8.14: **Simulation of conventional architecture** $Kp = 1, Ki = 15/2^{11}$

Fig. 8.15 presents result of a simulation of the new ADPLL structure presented in this study, which use a PI filter without dynamic adaptation of the K_p (cf. explications for fig. 8.13). Same initial condition and stimulus as last one are applied. The coarse frequency code of PLL is reset with the new code after the change of frequency at 10 us. In this case, the re-convergence time is 2 us.

If an adaptive PI filter is implemented in the model as shown in fig. 8.13, the re-convergence time is shortened to 1 us (Fig. 8.16). As expected, the proposed PLL highlights a very high convergence speed.

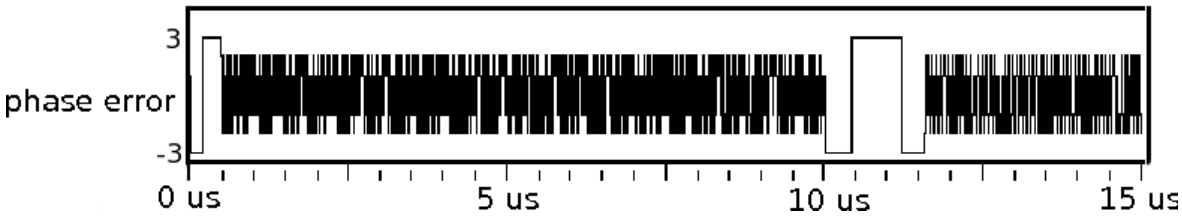


Figure 8.15: **Simulation of new architecture with regular PI filter** $Kp = 1, Ki = 15/2^{11}$

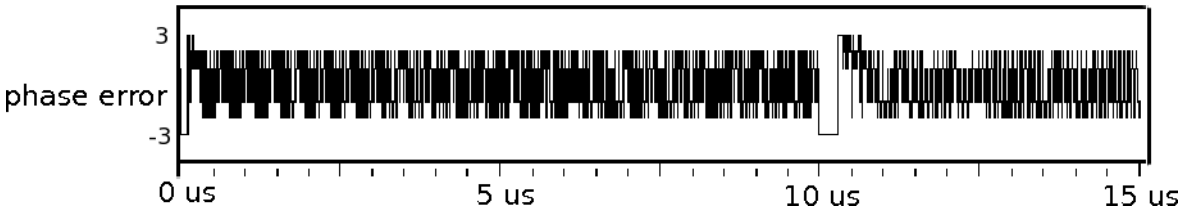


Figure 8.16: **Simulation of new architecture with adaptive filter** $Kp = 1$ **or** 3**,** $Ki = 15/2^{11}$

8.4.2 Power consumption comparison

Syntheses using Synopsys Design Compiler are done for conventional architecture [22] and proposed architecture under ST Microelectronics CMOS 065 nm technology. The *div* signal is chosen to be at PLL nominal frequency. Tab. 8.3 shows the power consumption of two architectures in μW. We can see that 37.4% of total power consumption is reduced by using the proposed architecture.

Table 8.3: **Power consumption of two architectures**

cell	Conventional Architecture	Proposed Architecture
PFD	20.9	16.1
Loop Filter	51.0	12.4
Mean Filter	-	11.7
Incrementer	-	4.8
Total	71.9	45.0

8.5 Clock distribution network using "sliding window" ADPLL

From the analysis above, we can see the proposed "sliding window" ADPLL possesses two advanced features: fast locking speed and lower power consumption. Locking speed and power consumption, especially the latter are two important indexes for evaluation of a clock generation and distribution system. In this chapter, we try to employ this "sliding window" ADPLL technique in a coupled clocking network presented in previous chapters, and evaluate the network performance in terms of functionality and power.

8.5.1 Network structure

In this work, we use a 4×4 network shown in Fig. 8.17, which has the same topology as the network presented in Chapter 1. It is composed of Phase Frequency Detectors (PFD) and 16 Filter/Oscillator (FO) blocks. PFDs are placed on each border between two synchronous clock areas (SCA), measuring the phase error between each couple of neighboring oscillators. The PFD placed in the upper left corner compares the phase of the input reference and the first oscillator in the network.

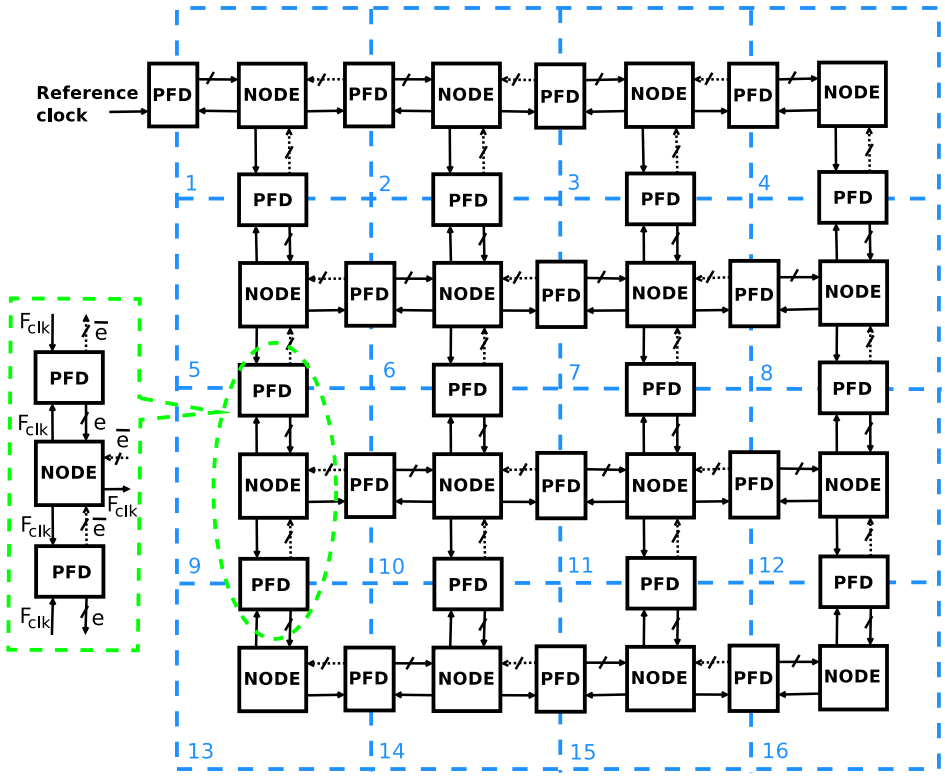


Figure 8.17: PLL network topology

Fig. 8.18 shows the architecture of one typical local clock generation node (NODE6) in Fig. 8.17. Phase tracking is achieved by 3-bit PFDs and a 6-bit filter generating a 6-bit signed code based on the phase difference between local generated clock and its neighboring-node clocks. The neighboring nodes taken into account depends on configuration of the whole

system and can be reconfigured by programming the coefficients $kw1$, $kw2$, $kw3$ and $kw4$, which have 2 bits. In this experiment, in most cases, kw is programmed as 0 or 1.

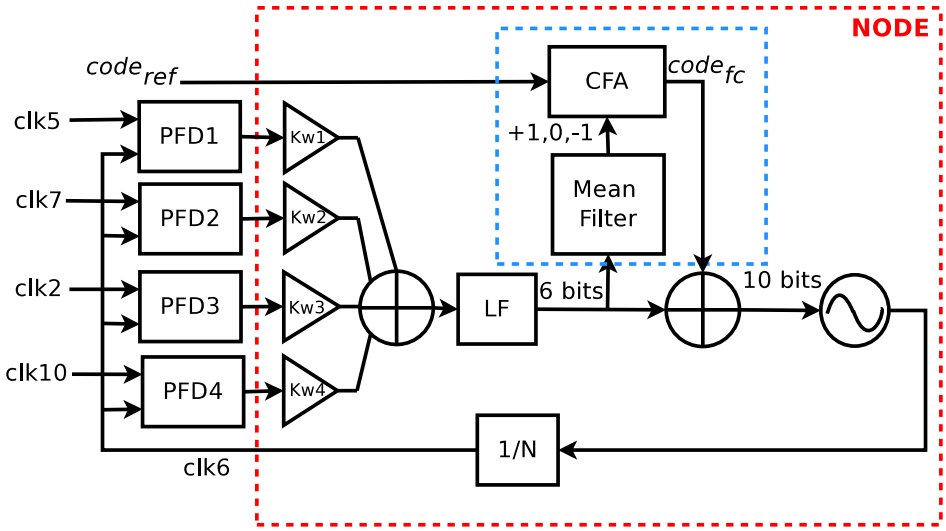


Figure 8.18: **PLL with sliding window**

The coarse estimation code of reference frequency ($code_{ref}$) comes from the outside, through the programming interface. This is necessary for coarse frequency code update by *CFA*, so it should be provided to the nodes in the start stage of system.

8.5.2 Evaluation of functional performance of system

As presented in Section 8.1, the proposed novel structure of digital filter has introduced additional parameters for ADPLL, thus increased the degrees of freedom. Subsection 8.4.1 has demonstrated that in the ideal case, one single ADPLL using this kind of filter is stable and has a fast establishment speed if the frequency of reference clock is constant. An ideal case is defines as follows:

1. Reference code given by RFI block ($code_{ref}$) is equal to the DCO control word corresponding to the desired frequency (reference frequency).

2. No fabrication mismatch. DCOs in different positions of the chip operate have exactly the same characteristics (same gain and transfer function).

3. No change of DCO frequency due to temperature variation.

However, in a network of coupled oscillators , one node receives at most four reference clocks, whose frequencies are not always constant, especially in the frequency acquisition mode. Whether the network is still stable is the utmost question. When the desired stable state is established, if we change the frequency of reference clock, whether the system can re-establish to the new synchronized state should be verified.

Moreover, if the condition of the ideal case (cf. List 3) is not satisfied, the performance of system in terms of convergence speed and residual phase error should also be observed.

For these reasons, we have performed various simulations on the network of ADPLL with "sliding-window", and we observed phase error between each two neighboring nodes. In this section, we only display the values of phase error between NODE16 (the most remote node to the reference) and its two neighbors.

Each simulation has two phases. The first phase is from 0 to 90 us. In this phase, the frequency of reference clock is 200 MHz. In the second phase from 90 us, to observe the re-establishment speed, the reference changes its frequency to 327 MHz at 90 us. The corresponding reference code is reprogrammed. Moreover, to avoid undesired steady state, in each phase, the network is dynamically configured, first in unidirectional mode, then in bidirectional mode(cf. Fig. 1.22). This procedure is displayed in Tab. 8.4.

Table 8.4: Simulation procedure

time (us)	$0 \sim 30$	$30 \sim 90$	$90 \sim 120$	$120 \sim 150$
ref clock (Hz)	200M	200M	327M	327M
code_ref	253	253	900	900
configuration mode	unidirectional	bidirectional	unidirectional	bidirectional

Besides the ideal case simulation, three other kinds of simulations are carried out to evaluate performance of proposed system in non-ideal cases.

The following simulations have been performed to verify the functionality of network in different cases (fabrication mismatch, temperature variation, etc.)

1. Ideal case (cf. List 3): Phase errors between NODE 16 and its two neighboring nodes are displayed in Fig. 8.19. From the simulation result, we can see that there is no undesired steady state, and maximum phase errors are ± 2 as in one single ADPLL.

2. Non-ideal case 1: reference code ($code_{ref}$) has an offset compared with the theoretical DCO control word which corresponds to the desired frequency (reference frequency): the reference code ($code_{ref}$) is estimated by RFI block. However, the frequency of DCO may change due to PVT variations, thus the estimated $code_{ref}$ may has an offset with respect to the theoretical code calculated from the frequency and the transfer function of DCO. Fig. 8.20-Fig. 8.23 show simulation results when reference code has an offset of 1, 2, 3 or 4 compared to the correct code. We can see that in all these simulations, the NODE 16 is synchronized with its neighbors with residual errors less than ± 2 times of PFD resolution. However, compared with the ideal case (cf. List 3), the frequency tracking takes longer time.

3. Non-ideal case 2: static disparity among nodes: due to fabrication mismatch, DCOs in different positions of the chip may generate different frequencies with the same

command code. Fig. 8.24 proves the static offset immunity of proposed system when there is a constant offset error of 5 over transfer function of the DCO in NODE 16.

4. Non-ideal case 3: dynamic disparity among nodes: due to temperature variation, the frequency of some nodes may vary in function of time during the operation of circuit. A dynamic offset (-5 to +5) is added to the transfer function of the DCO in NODE 16. Simulation result shown in Fig. 8.25 demonstrates that in most of the time, the residual errors are less than ± 2 times of PFD resolution.

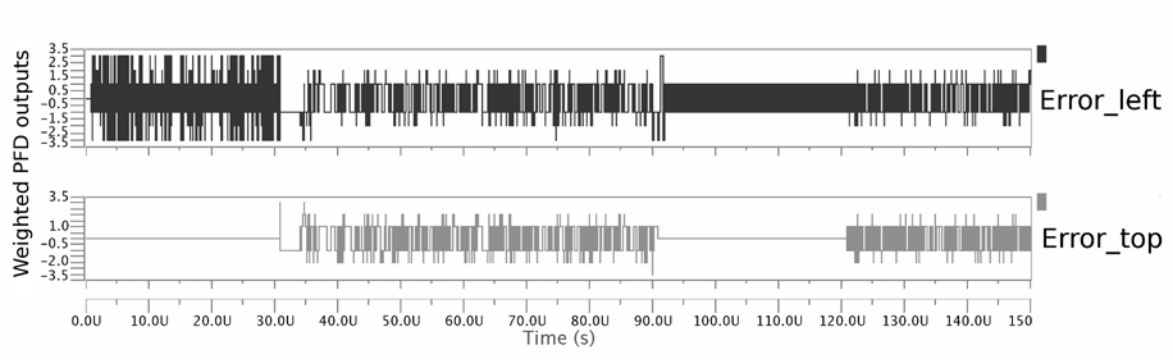


Figure 8.19: Error entries of node 16 mutiplied by *kw* (reference code=theoretical code)

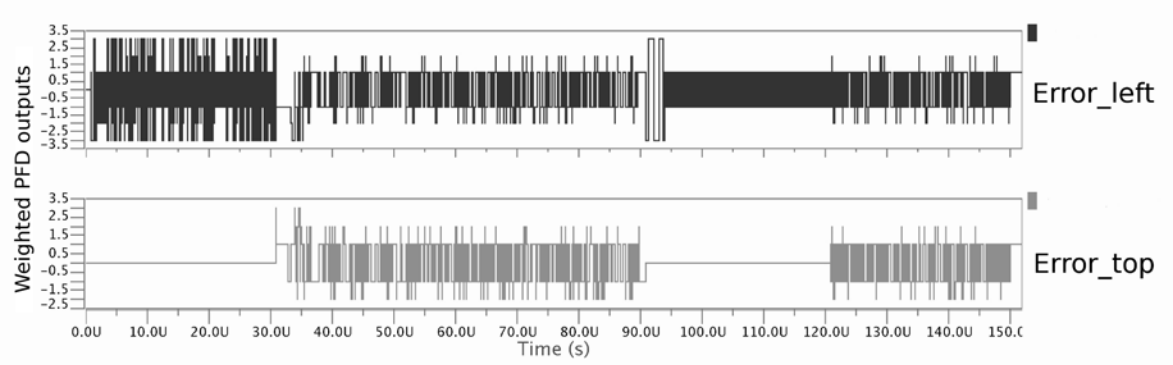


Figure 8.20: Error entries of node 16 mutiplied by *kw* (reference code=theoretical code + 1)

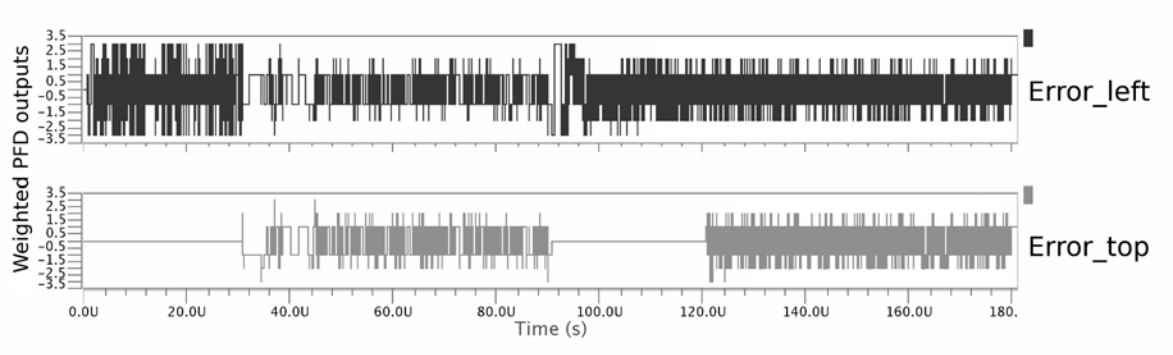


Figure 8.21: Error entries of node 16 mutiplied by *kw* (reference code=theoretical code + 2)

From the simulation results we can get three conclusions: First, in all the above test conditions, the whole network can be synchronized with the reference clock with a certain

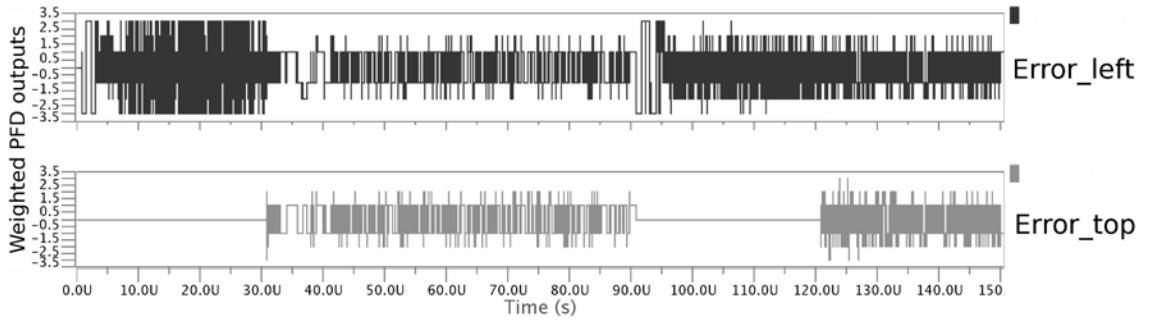


Figure 8.22: Error entries of node 16 mutiplied by *kw* (reference code=theoretical code + 3)

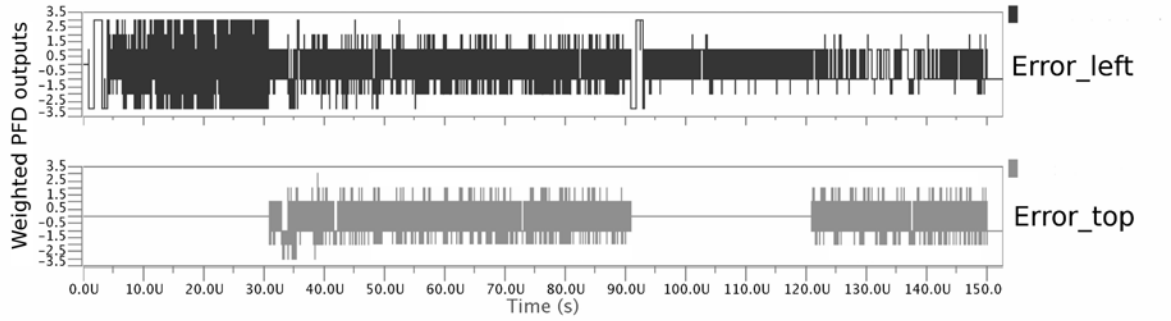


Figure 8.23: Error entries of node 16 mutiplied by *kw* (reference code=theoretical code + 4)

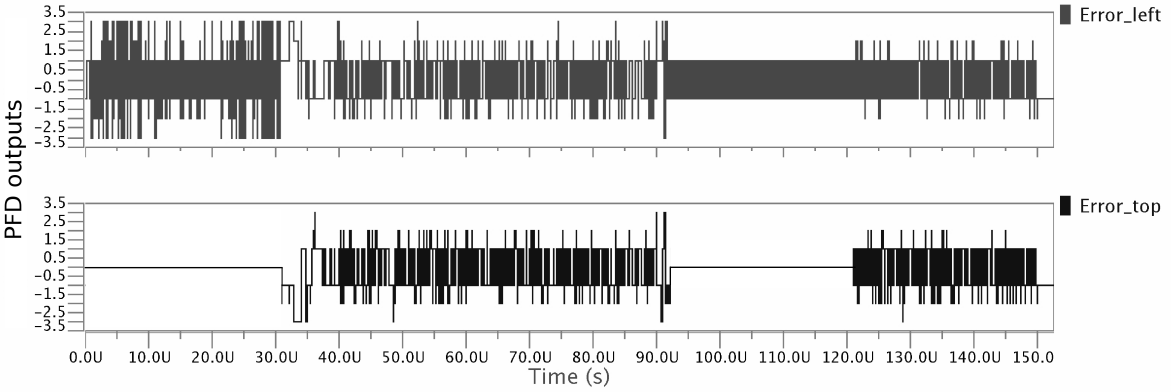


Figure 8.24: Error entries of node 16 mutiplied by *kw* (static offset = 5)

precision (maximum error: ± 2 or ± 3 times PFD resolution). Second, a RFI block with bad precision could degrade the system mainly in frequency tracking speed. Third, PVT variations of DCO may also result in degraded performance in terms of convergence speed and phase error.

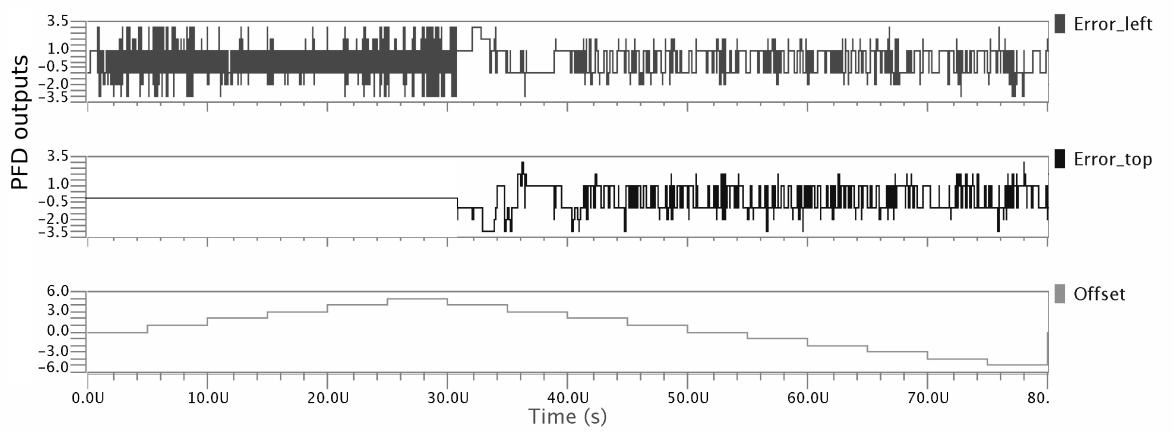


Figure 8.25: Error entries of node 16 mutiplied by *kw* (dynamic offset -5 to 5, only the first phase from 0 to 90 us is displayed)

8.6 Conclusion

In this chapter, an ADPLL architecture with sliding window is proposed to accelerate frequency tracking and to reduce power consumption.

In the first part of this chapter, we discusses the drawbacks of conventional ADPLL and existed approaches for fast frequency acquisition. These solutions mainly suffer from complex structures thus large area and power. Hence, they are not appropriate for clock generation and distribution. To solve this problem, we present a new ADPLL architecture with separated control for frequency acquisition and phase error correction. System structure and block working mechanism are explained. Then behavior level simulations are performed and power consumption is analyzed. Some results of the research on it have been published on the international conference NEWCAS2012[51].

In the second part of this chapter, this "sliding window" technique is implemented in a 4×4 clock distribution network. Simulation results demonstrate a good performance even with DCO disparity.

However, blocks like mean filter and CFA introduce additional parameters to the transfer functional of ADPLL, thus increase the degree of freedom of the non-linear system. To achieve best performance, a theoretical analysis is necessary for choosing appropriate parameters like the number of values stored in the circular buffer of the mean filter and the depth of the integrator in the CFA block.

Chapter 9

Conclusion and Perspectives

Contents

9.1 Thesis summary and conclusions 183

9.2 Future work 186

9.1 Thesis summary and conclusions

Clock generation and distribution are important techniques in the field of VLSI circuits. The quality of the clock signal has a great impact on circuit performance and power consumption. Nowadays, a variety of clocking solutions like clock tree, hybrid clock mesh, asynchronous communication is being used. These techniques exhibit an upper limit of the circuit size which can be synchronized. In deep submicron technologies, these conventional clocking techniques fail the generation of a global clock usable at any point of the chip.

The objective of this research is a study of an alternative clocking solution, which employs a network of oscillators coupled by all-digital PLLs (ADPLL) technique. Such a system is called "network of ADPLL". This PhD project included a theoretical study, modeling, CMOS and FPGA prototyping of networks of ADPLL. In comparison with previous researches, which showed principal possibility of clock generation by network of ADPLL, this study proved a feasibility of a *large* network of ADPLL, and solved some problems specifically related with the large size of network. This PhD report describes in detail the design of a chip containing a Cartesian network of 10x10 nodes, each of them generating a local clock synchronized in phase and in frequency with the neighboring clocks. The modeling of the network at different abstraction levels was an important part of the work. The ADPLL network is a complex system: only its reduced model can be described by equations, and realistic effects can only be accounted for with a numerical modeling. However, we proposed a reduced model used for the design, relating the network parameters to the quality of the synchronization of the local clock signals.

The design of a VLSI prototype of the network used was based on the study previously

achieved in the HODISS project, where a low-size network prototype was implemented on silicon. However, in this PhD, we adapted the design of several blocks of the network to the constraints related to the size of the network, and we modified the phase detector so to improve its resolution (20 ps instead of 32 ps in the previous work). To realize this resolution, a Vernier architecture was used. The DCO block was simplified (reduced size and power consumption). We designed two versions of the filter: one with architecture optimizing the convergence speed and power consumption, and one being a simplified version of the architecture used in the first low size prototype (4×4) of network. The chip containing a 10×10 network of ADPLL has been fabricated in 65 nm technology, and is presently under test.

Manycore circuits are probably the most important potential applications of the proposed clocking solution. For such systems, the question of scalability is paramount: how the performance of the system changes if the size of the system increases? For our clocking solution, the question can be formulated as follows: how will the synchronization quality (the clock error) with the size of the network (the number of nodes) change? Our study provides the following response to this question:

– If the phase error between the neighboring clocks is considered, the response is “the clock error expressed in time units doesn’t change when the size of the network increases”. The clock error between the neighboring nodes is equal to 2 steps of resolution of the phase error measurement. The phase error measurement resolution scales down with sizes of the CMOS transistors (in the same way as the delays of the digital gates).

– If the phase error between any two nodes of the network is considered: we showed that although the phase error increases with the Manhattan distance between the nodes (equivalently, with the size of the network), a saturation is observed, and the global error is limited to 3 steps of the phase error measurement resolution.

These facts were verified experimentally, on a network of size 12x12 nodes implemented with an FPGA platform; hardware limitations prevented us from test on networks with larger sizes.

One of the main advantages of network of oscillators coupled by ADPLL networks is the possibility to dynamically reconfigure its topology, connectivity, parameters of the processing blocks, etc. This allows and implementation of different network behaviors, with the same hardware platform. This property have been successfully used to select of the desirable synchronized mode. As well, we explored an original technique inspired from the hydrodynamics, allowing an elimination of standing waves which may occur in the network. By configuring the network border differently from the network kernel, the network border can be regarded as an independent and synchronous ring. The nodes composing this ring excite the inner kernel and absorb the error waves coming back from the kernel, like the overflow channels of a swimming pool. This "swimming pool"-like topology has been stud-

ied and designed. Its advantage in preventing error wave propagation and reflection has been demonstrated by simulation results.

The measurement of clock error can be required in many cases: for characterization, of for self-test of the clocking system, ... Since clock signal is continuous in time and it is very sensitive to noise, off-chip measurement is not appropriate for high frequency signals. In this project, we designed an on-chip clock error measurement circuit, which measures the clock error statistics between two clock domains in high-speed clocking systems (gigahertz and more). The result of measurement may be read off-chip with low rate. The strategy aims at picoseconds resolution without complex calibration. After a validation of the technique on a discrete prototype, an IC block was designed using 65 nm technology. This mixed signal IP was used in the the test chip containing a large ADPLL network, for measurement of errors between neighboring clocks in 4 places of the network.

The built-in test circuit, together with the clock generation and distribution circuit, constitutes a complete clocking system, which proposes and alternative to existing conventional clocking techniques for large digital systems on chip.

9.2 Future work

The test chip has been fabricated, and the test is being prepared at the time when this document is being written. If the test results are satisfying, the next step will be integrating the proposed clocking system in a real multiprocessor SoC.

Moreover, during the research, we found some challenges or points of improvement with respect to the modeling of such a non-linear system, the robustness of clock distribution and the deployment in 3D integrated circuits.

9.2.1 Modelling of ADPLL for the study of residual phase error in steady state

A challenge in analyzing the behavior of phase error correction of an ADPLL or a network of ADPLLs comes from the nonlinearities of these systems. The non-linearity appears due to the following reasons:

- The characteristic of the phase detector is mod 2π.
- The analog/digital interfaces between the digital filter and the DCO, and between the DCO and PFD.
- The sampling of the DF by the DCO whose signal is controlled by the filter output: a self-sampling

The most common and efficient techniques are those originated from control theory for Linear Time-Invariant (LTI) systems. To apply the analysis tools coming from the LTI system theory, it is necessary to neglect the quantization in the ADPLLs. This leads to inaccurate predictions of residual errors (i.e. when a network is very close to the synchronized state). For example, a continuous LTI model of a PLL with two integrators predicts a zero residual phase error. However, in a realistic ADPLL network, the residual error is at least equal to the quantization step of the PFD, and is a function of time looking like a stochastic process. The mode when the phase errors are small is ”nominal” for an ADPLL network which is close to the synchronization, and hence, the failure of the LTI model to describe this operation mode is a serious drawback.

In Chapter 2, we have proposed a reduced numerical model of ADPLL and a method of analyzing the impacts of characterization of PFD, DCO and digital filter on the residual phase error of an ADPLL in steady state. This method serves as an estimation of the maximum value of residual error with different block parameters (quantization steps of PFD and DCO, filter coefficients, etc.). A more precise and complete model considering the non-linearity is necessary for the study of steady-state behavior of ADPLL in a coupled network.

9.2.2 Exploration of fault-tolerance property

A very serious issue with clock tree or other centralized approaches is the lack of robustness. If due to some manufacturing defect, one internal wire or buffer in a clock tree breaks, the whole sub-tree starts from the broken point will stop working correctly. The proposed clocking system, as a distributed approach, is probably more robust. But since the whole network is coupled, if one link between two clock domains breaks or one oscillator works abnormally, the neighboring nodes and even the whole network may also be affected. A possible solution may be a use of the phase detectors, which always observe the efficiency of the local clock generators. Since the system is digital, it is possible to develop a mechanism detecting the position of the failure, and reconfiguring the network so minimize the impact of the failure on the behavior of the network.

In general, there are two kinds of faults in the network: a broken link and a faulted node. If a link breaks, the PFD cannot well detect the phase error between two clocks or cannot send back the correct results. In this case, this link should not be taken into consideration while calculating the DCO control word. Since all the links have their weight coefficient, it is easy to deactivate a link if it is broken. A faulted node is more difficult to deal with. One possibility is bypassing this node and using the neighboring clock signal provisionally.

These solutions have a hardware cost, and the designers should find a trade-off between the hardware overhead and the system robustness.

9.2.3 Clock distribution for 3-D chip

Three-dimensional (3D) IC is an emerging technology being pursued by industry and laboratories. It offers new levels of efficiency, power, performance, and other advantages. The clock distribution in this circuit must be in three spacial dimensions, and this increases the complexity of the clock generator. Several researches address these problems [34, 35, 38, 6]. Three typical topologies have been summarized in Fig. 9.1[38]. In general, they combine different topologies used in 2-D circuits, such as clock tree, ring and mesh. The common point in the three approaches displayed in Fig. 9.1 is that a clock tree is implemented in the 2^{nd} plane. The root of clock tree is connected with the reference clock. In Fig. 9.1(a), two other clock trees are used in the other two planes. The root nodes in both planes are connected with the root of the 2^{nd} plane by Through-Silicon Via (TSV). This results in a 3-D clock tree with only one real root node, and the skew between two leaf nodes is larger than in a 2-D clock tree. In Fig. 9.1(b), the 1^{st} and 3^{rd} planes use mesh structure instead of tree structure. Each node in the meshes are connected with the 2^{nd} plane node by TSV, thus a large number of TSVs are necessary. This results in a difficulty of physical implementation. In Fig. 9.1(c), clock rings are used in the 1^{st} and 3^{rd} planes, and only the four corner nodes are connected with the corner nodes of the 2^{nd} plane. However, if the plane is large, the synchronization between central nodes in different planes is difficult to be guaranteed. We can see the native problems of these 2-D conventional clocking solutions are multiplied by

the 3-D implementation. As a result, these solutions are of limited efficiency.

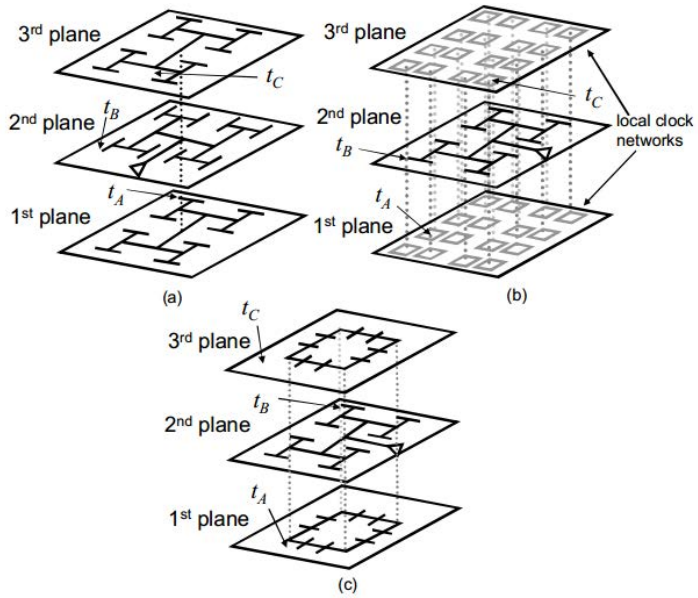


Figure 9.1: **Various 3-D clock distribution approaches within the test circuit of [38]**: (a) H-trees, (b) H-tree and local rings/meshes, (c) H-tree and global rings

The study presented in this PhD project suggest a different solution to the problem of synchronization in 3D chips. This method (cf. Fig. 9.2) is inspired by the approach displayed in Fig. 9.1(c). Instead of using a clock tree in the 2^{nd} plane, a distributed architecture of ADPLL network presented in the document (cf. Fig. 2.1) can be implemented. One corner of the network is connected with the reference clock. Two other networks of ADPLLs are constructed in the 1^{st} and 3^{rd} planes. The four corner nodes in each of these two planes are coupled with the four corner nodes in 2^{nd} plane also by using ADPLL technique. Moreover, since the clocking networks in the 1^{st} and 3^{rd} planes have four reference clock signals, these two can be configured as "swimming-pool" architecture proposed in Chapter 7. This proposed approach exploits the reliability and scalability of ADPLL network compared with conventional approaches used in Fig. 9.1.

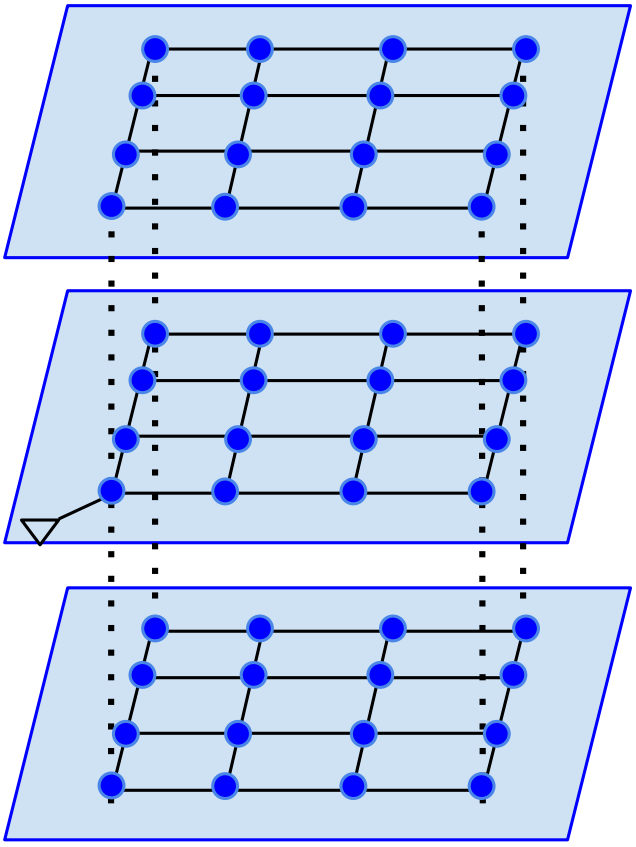


Figure 9.2: **Proposed 3-D clock distribution approach using the network of ADPLL**

Appendices

Appendix A

VHDL models of the ADPLL blocks

This appendix presents the behavioral and RTL-level VHDL models of the ADPLL blocks: a BB-detector, a TDC, a digital signal processing block (i.e. loop filter) and DCO.

Listing A.1: VHDL model of the digitally-controlled oscillator

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.MATH_REAL.all;
use IEEE.NUMERIC_STD.all;

entity DCO_V2 is
 generic (
        TYPEDCO : integer := 1; -- DCO type: 1 - LIP6 TT, 2 - LIP6 SS, 3 - LIP6 FF
        RESOLUTION : time := 1fs;        -- finest time resolution,fs
        DCO_INIT_DELAY: time := 1ps;     -- initial oscillation delay,ns
        DELTA_F: real := 0.0;            -- frequency offset,MHz
        DCO_JRMS: time := 0fs;  -- jitter rms, fs -- 1821fs
        DCO_WRMS: time := 0fs            -- wander (accumulative jitter) rms, ps
        );
 port (
        C : in STD_LOGIC_VECTOR(2 downto 0);
        B : in STD_LOGIC_VECTOR(6 downto 0);
        A : in STD_LOGIC_VECTOR(6 downto 0);
        RST: in STD_LOGIC;
        CLK, DIV, DIV16: out std_logic
        ) ;
end DCO_V2 ;

architecture behavioral of DCO_V2 is
--================================================================
        signal enable : std_logic := '0';
        signal DCO_IN : STD_LOGIC_VECTOR(9 downto 0) := "0000000000";
        signal DCO_CODE : integer range 0 to 1023 := 0;
        signal frequency_dco : real := 1.0;
```

```
signal smp: bit := '0';
signal period0: time := 1ns;
signal Q1, Q2, Q3: std_logic := '0';
signal D1, D2, D3: std_logic := '1';
signal CLK_TMP: std_logic;
type LIP6_TT is array (0 to 255) of real;
signal freq_lip6_TT : LIP6_TT :=(
        0 => 0.9300370537880846,
        1 => 0.9314933063967212,
        2 => 0.9327363758105037,
        3 => 0.9339820993709525,
        4 => 0.9341469253729105,
        5 => 0.9355143576421766,
        6 => 0.9367689231684877,
        7 => 0.9380246199233998,
        8 => 0.9382774501565182,
        9 => 0.9396552288145633,
        10 => 0.9409152019478633,
        11 => 0.9421816085211267,
        12 => 0.942434865588007,
        13 => 0.9438237077209502,
        14 => 0.9450084998977326,
        15 => 0.946285862506654,
        16 => 0.9466387151988006,
        17 => 0.9480404344121861,
        18 => 0.9492327538251522,
        19 => 0.95051551955283,
        20 => 0.9508613686147231,
        21 => 0.9522756907203746,
        22 => 0.9534797275460731,
        23 => 0.9546865840703788,
        24 => 0.9551165951716415,
        25 => 0.9565436327568848,
        26 => 0.957758204323161,
        27 => 0.9589734456211345,
        28 => 0.9594535730158974,
        29 => 0.9608867344280492,
        30 => 0.9621120484568325,
        31 => 0.9633385064584414,
        32 => 0.9635115184762559,
        33 => 0.9648595848196898,
        34 => 0.9660955347772218,
        35 => 0.9673322479637345,
        36 => 0.9673219727702526,
        37 => 0.9685919359191155,
        38 => 0.9698378354967289,
        39 => 0.9710846891070695,
        40 => 0.9714170594790796,
        41 => 0.9726959223095312,
        42 => 0.9739472760028322,
```

```
43 => 0.9752037515171227,
44 => 0.9755492269766837,
45 => 0.9768396173055278,
46 => 0.9780185289690061,
47 => 0.9792864251065894,
48 => 0.9797249749019787,
49 => 0.9810260642889341,
50 => 0.9822126079644305,
51 => 0.9834859504302272,
52 => 0.9839184826703808,
53 => 0.9852304139412236,
54 => 0.9864281895272759,
55 => 0.9876277064555761,
56 => 0.9881447390114096,
57 => 0.9894692902110069,
58 => 0.9906769448884141,
59 => 0.9918847185511414,
60 => 0.9924402228791357,
61 => 0.9937689464745228,
62 => 0.9949871587910142,
63 => 0.996205950638606,
64 => 0.9964658577029665,
65 => 0.9978049978635253,
66 => 0.9990265861960093,
67 => 1.0002550116783496,
68 => 1.0002512245286794,
69 => 1.001513937689658,
70 => 1.0027464685446398,
71 => 1.0039835863467727,
72 => 1.0040555971801971,
73 => 1.005325032936361,
74 => 1.0065620293795377,
75 => 1.0078103082670074,
76 => 1.0081496984007677,
77 => 1.0094309213624125,
78 => 1.0105984622125846,
79 => 1.0118557871706545,
80 => 1.0122964540808017,
81 => 1.0135901594223622,
82 => 1.0147638127022536,
83 => 1.0160257464795259,
84 => 1.0164571784011593,
85 => 1.0177588246342813,
86 => 1.0189443828094932,
87 => 1.0201359351918826,
88 => 1.0206480161282774,
89 => 1.0219617651579865,
90 => 1.0231579238853046,
91 => 1.0243566202651181,
92 => 1.0249126065984685,
```

```
93 => 1.0262319785377972,
94 => 1.0274378102949708,
95 => 1.02864680746881,
96 => 1.0289053953625891,
97 => 1.0302380381995705,
98 => 1.031376267293398,
99 => 1.0325966106838765,
100 => 1.0326760723468008,
101 => 1.0339327870647023,
102 => 1.0350802089537807,
103 => 1.0363092676419543,
104 => 1.0364539483583242,
105 => 1.0377180158447331,
106 => 1.038868973413971,
107 => 1.0401072466198446,
108 => 1.0402594324753714,
109 => 1.0415343717763726,
110 => 1.0426234941634994,
111 => 1.0438712169721174,
112 => 1.0443679098375208,
113 => 1.0456535957469052,
114 => 1.0467485691210318,
115 => 1.0480011887837504,
116 => 1.0484954199858787,
117 => 1.0497911540400287,
118 => 1.0508970297898504,
119 => 1.0520804372134308,
120 => 1.0526516289075247,
121 => 1.0539589130422671,
122 => 1.0550724281794993,
123 => 1.05626375874356,
124 => 1.0568812395562282,
125 => 1.0581926843265866,
126 => 1.0593160319459548,
127 => 1.0605182820933722,
128 => 1.060863442931976,
129 => 1.0621849999102244,
130 => 1.0633151776059666,
131 => 1.0645215355855339,
132 => 1.0646121565383941,
133 => 1.0658617002144246,
134 => 1.067000736532671,
135 => 1.0682140856411849,
136 => 1.0683725644162426,
137 => 1.06962996320408,
138 => 1.0707744529445549,
139 => 1.0719968485981051,
140 => 1.0721503277759079,
141 => 1.0734182586645921,
142 => 1.0745021366535702,
```

```
143 => 1.0757341845167482,
144 => 1.075975486255241,
145 => 1.0772518859639344,
146 => 1.078340568491569,
147 => 1.079577143718022,
148 => 1.0800600694385784,
149 => 1.0813468602477622,
150 => 1.082445413493106,
151 => 1.0836167398246546,
152 => 1.084182153111391,
153 => 1.0854793388439317,
154 => 1.0865865473748093,
155 => 1.087764898465793,
156 => 1.0883741063849676,
157 => 1.0896766314112573,
158 => 1.090792344312327,
159 => 1.0919799153780513,
160 => 1.0923111916650488,
161 => 1.0936232249195786,
162 => 1.0947478647457294,
163 => 1.095873001410797,
164 => 1.096037633436344,
165 => 1.0972790961406708,
166 => 1.0984121337775466,
167 => 1.0995448507451782,
168 => 1.0997751425286994,
169 => 1.101025132788564,
170 => 1.1021619589375002,
171 => 1.103302413609939,
172 => 1.103532517387344,
173 => 1.1047910663998597,
174 => 1.1058696948103037,
175 => 1.107017909625944,
176 => 1.107323317080391,
177 => 1.1085916538760455,
178 => 1.109676828876273,
179 => 1.110829052075874,
180 => 1.111127186398971,
181 => 1.1124050312671583,
182 => 1.1134982723904018,
183 => 1.1145915357962072,
184 => 1.115202495535452,
185 => 1.11649064638043,
186 => 1.1175918726114063,
187 => 1.1186919136951094,
188 => 1.119359433722918,
189 => 1.1206533433416233,
190 => 1.1217629419967327,
191 => 1.122872762497417,
192 => 1.1232614260909624,
```

```
193 => 1.124566364255586,
194 => 1.125683900548983,
195 => 1.126799920844845,
196 => 1.1269691268044276,
197 => 1.1282060321781397,
198 => 1.1293317341510592,
199 => 1.130454608953153,
200 => 1.1306854018233223,
201 => 1.1319309047572412,
202 => 1.1330601958766427,
203 => 1.1341899497169578,
204 => 1.1344189193765202,
205 => 1.1356729862795563,
206 => 1.1367456832427793,
207 => 1.1378847993243437,
208 => 1.1381890905483081,
209 => 1.1394513823449812,
210 => 1.1405303945448925,
211 => 1.1416738293515332,
212 => 1.1419599059027288,
213 => 1.1432308239535463,
214 => 1.1443179846199658,
215 => 1.145403814449554,
216 => 1.14575918837563,
217 => 1.1470392276249652,
218 => 1.1481328998483229,
219 => 1.1492249624058905,
220 => 1.149874814165585,
221 => 1.1511598660878904,
222 => 1.1522606218196583,
223 => 1.1533618270264697,
224 => 1.1537984132083445,
225 => 1.1550880654104817,
226 => 1.1561968494264178,
227 => 1.1573066446221745,
228 => 1.1574760473916996,
229 => 1.1586938725351493,
230 => 1.1598101259862483,
231 => 1.1609260013288355,
232 => 1.16116755716054,
233 => 1.162393615220309,
234 => 1.163513596815299,
235 => 1.1646373300748365,
236 => 1.1648712198680549,
237 => 1.166105866329324,
238 => 1.1671703109316738,
239 => 1.1683009340075846,
240 => 1.168611994244867,
241 => 1.1698536703885126,
242 => 1.1709250044716103,
```

```
                243 => 1.1720595287042913,
                244 => 1.1723543849659193,
                245 => 1.1736051203171525,
                246 => 1.174683674791986,
                247 => 1.175761681055491,
                248 => 1.1761149761936955,
                249 => 1.1773733180630202,
                250 => 1.178458229245494,
                251 => 1.1795430191574976,
                252 => 1.1799544782818189,
                253 => 1.1812190791906478,
                254 => 1.1823124380448046,
                255 => 1.1834041379314458
);
        type LIP6_SS is array (0 to 255) of real;
        signal freq_lip6_SS : LIP6_SS :=(
                0 => 0.47353787163148445,
                1 => 0.4750798961481912,
                2 => 0.4768305416578326,
                3 => 0.47845576313639486,
                4 => 0.4795212193780018,
                5 => 0.4808333267036447,
                6 => 0.48255190454449844,
                7 => 0.48423397108080786,
                8 => 0.48542209156169176,
                9 => 0.48689056003567094,
                10 => 0.4885839096161608,
                11 => 0.49032214815937483,
                12 => 0.491821352175315,
                13 => 0.4933086319105339,
                14 => 0.4948530515003635,
                15 => 0.4966102131143899,
                16 => 0.49827458497373366,
                17 => 0.4997843375446064,
                18 => 0.5014101291899685,
                19 => 0.5031475795023409,
                20 => 0.5047927272363789,
                21 => 0.50640697660733,
                22 => 0.50788545852076775,
                23 => 0.5093913215244703,
                24 => 0.51157417913163006,
                25 => 0.51311792770676214,
                26 => 0.5146215278257331,
                27 => 0.5161917003287998,
                28 => 0.5183706412641029,
                29 => 0.5199887065628765,
                30 => 0.5215984022642016,
                31 => 0.52316217890890926,
                32 => 0.5243155184656028,
                33 => 0.5256358673447322,
```

```
34 => 0.5273221167265943,
35 => 0.5289111488492946,
36 => 0.52945016901847017,
37 => 0.53055889424871963,
38 => 0.53232898374084157,
39 => 0.53386523152192974,
40 => 0.53541561655824476,
41 => 0.5367355853884937,
42 => 0.5382939823493819,
43 => 0.5400694765335377,
44 => 0.5416467362118495,
45 => 0.5428526415731839,
46 => 0.5445714412795614,
47 => 0.5462208827432398,
48 => 0.5479939341319919,
49 => 0.5494189642254376,
50 => 0.5510803180633459,
51 => 0.5527312582559662,
52 => 0.5545738433900838,
53 => 0.5560167377388471,
54 => 0.55765487624655,
55 => 0.5590242986505578,
56 => 0.561343237794428,
57 => 0.5626927361813105,
58 => 0.5642405292674373,
59 => 0.5658676345562271,
60 => 0.5679745409069881,
61 => 0.5694863013373543,
62 => 0.5710219937302508,
63 => 0.5726409956844348,
64 => 0.5740078199252572,
65 => 0.5754656912456834,
66 => 0.5771356638468138,
67 => 0.5787676840012825,
68 => 0.5792299831903584,
69 => 0.5807054065542051,
70 => 0.5823731293426238,
71 => 0.583962101012873,
72 => 0.5846443726614448,
73 => 0.5859297374899429,
74 => 0.5874501276426165,
75 => 0.5893172363958257,
76 => 0.5907515038305264,
77 => 0.5921084758695554,
78 => 0.5936654976071548,
79 => 0.5952363095865492,
80 => 0.5972839698959626,
81 => 0.5984611156355463,
82 => 0.6000331923289115,
83 => 0.6017462752481478,
```

```
84 => 0.6037124272936218,
85 => 0.6050019657100149,
86 => 0.6066113172651185,
87 => 0.608254326597133,
88 => 0.6101866208893467,
89 => 0.6117760447581635,
90 => 0.6132604607428234,
91 => 0.6149414650521972,
92 => 0.6169056778105348,
93 => 0.6185529940398048,
94 => 0.6202089239819356,
95 => 0.6218605018892102,
96 => 0.6231692043035017,
97 => 0.6246433793248394,
98 => 0.6260863021538861,
99 => 0.6277391916562314,
100 => 0.6287814661789652,
101 => 0.6300380352159535,
102 => 0.631510504721479,
103 => 0.6331706091267807,
104 => 0.6341975435829443,
105 => 0.6356565852331095,
106 => 0.6368901629440649,
107 => 0.6386303846864995,
108 => 0.6397501464001434,
109 => 0.6410536162355187,
110 => 0.6423612125774307,
111 => 0.6441764791009432,
112 => 0.6460555776588376,
113 => 0.6474142521556888,
114 => 0.6487949317999672,
115 => 0.6504395034371071,
116 => 0.6525719532353883,
117 => 0.6539643698430109,
118 => 0.6550069180244018,
119 => 0.6567459161002191,
120 => 0.659137346725542,
121 => 0.6605538019516782,
122 => 0.6617630639931386,
123 => 0.6633831351027545,
124 => 0.6657896757076938,
125 => 0.6672366578013157,
126 => 0.6686227809132576,
127 => 0.670153793206122,
128 => 0.6720401235302867,
129 => 0.6733390502228686,
130 => 0.6746592038624804,
131 => 0.6763526433246473,
132 => 0.6772877621572591,
133 => 0.6786493387941203,
```

```
134 => 0.679866071793677,
135 => 0.6816491609535787,
136 => 0.6825722534397978,
137 => 0.6838451740270474,
138 => 0.6852219027413515,
139 => 0.6869410817665379,
140 => 0.6877248281691186,
141 => 0.6891011713488133,
142 => 0.6904825009157798,
143 => 0.6919765198098598,
144 => 0.6932651689065175,
145 => 0.6947914712479113,
146 => 0.6959615141842906,
147 => 0.6977185075771053,
148 => 0.6997234964677187,
149 => 0.7011396163992823,
150 => 0.7025452327658045,
151 => 0.7039758541905105,
152 => 0.7063838483097038,
153 => 0.7077527782698398,
154 => 0.7090738661435417,
155 => 0.7106521650912834,
156 => 0.7128747093740522,
157 => 0.7143319833749319,
158 => 0.7157432952982041,
159 => 0.7173201608214239,
160 => 0.7191593651802464,
161 => 0.7205613132692655,
162 => 0.7219387283302767,
163 => 0.7233797241041741,
164 => 0.7242563530203192,
165 => 0.7253987570930805,
166 => 0.7268769117586603,
167 => 0.7282515906974636,
168 => 0.7291882559743241,
169 => 0.7304939170988497,
170 => 0.7320351935893528,
171 => 0.7335724036768407,
172 => 0.7347035688246788,
173 => 0.7361426858734515,
174 => 0.7375086553750641,
175 => 0.7392697987219986,
176 => 0.7405076624851545,
177 => 0.7419163430368987,
178 => 0.743233125246393,
179 => 0.7447957755597245,
180 => 0.7461447602014381,
181 => 0.7476380454807353,
182 => 0.7490693275896487,
183 => 0.7503772443568345,
```

```
184 => 0.7527753631931735,
185 => 0.7541317098817691,
186 => 0.7554812295723163,
187 => 0.7570247061247135,
188 => 0.7593012604084753,
189 => 0.7606275233489214,
190 => 0.7619249139527565,
191 => 0.7634305617972767,
192 => 0.7653415231833966,
193 => 0.7668264586062019,
194 => 0.7683303514209092,
195 => 0.769932477362855,
196 => 0.7707268021707914,
197 => 0.7720881799519732,
198 => 0.7736118058921466,
199 => 0.7750141677680416,
200 => 0.7762661572246499,
201 => 0.7774470113301194,
202 => 0.7787626817759277,
203 => 0.7802785555167195,
204 => 0.7815510225425582,
205 => 0.782878486348019,
206 => 0.7840736112457241,
207 => 0.7856473420127618,
208 => 0.7870747916544464,
209 => 0.7885601858224511,
210 => 0.7898237365168896,
211 => 0.7911685155845295,
212 => 0.792643655936216,
213 => 0.7939534155379362,
214 => 0.7953663084606063,
215 => 0.7968465071963443,
216 => 0.7983898347816966,
217 => 0.7998287749181893,
218 => 0.8012771230515684,
219 => 0.8026520225796071,
220 => 0.8050596172876132,
221 => 0.8062145190115328,
222 => 0.8078811009065503,
223 => 0.8093424448940842,
224 => 0.8111460840856409,
225 => 0.8125647692534343,
226 => 0.8138626247757113,
227 => 0.8153625065811672,
228 => 0.8164450035169256,
229 => 0.8173915291450429,
230 => 0.8188494183123528,
231 => 0.8205346026282175,
232 => 0.8216296492443256,
233 => 0.8228693928784833,
```

```
                234 => 0.8245087857049716,
                235 => 0.8260530199874566,
                236 => 0.8272410088673272,
                237 => 0.8285102711539702,
                238 => 0.8298596309478512,
                239 => 0.8314209624208616,
                240 => 0.8328848330049176,
                241 => 0.8341316409887373,
                242 => 0.8356267322730019,
                243 => 0.8372768562719824,
                244 => 0.8386791136317068,
                245 => 0.8396917349432461,
                246 => 0.8411387453735073,
                247 => 0.8425034206267366,
                248 => 0.8444903399946134,
                249 => 0.8457292337631282,
                250 => 0.8470230261369495,
                251 => 0.8483720440975616,
                252 => 0.8498344261593518,
                253 => 0.8513374925291146,
                254 => 0.8524226914672161,
                255 => 0.8543287256351534
);
        type LIP6_FF is array (0 to 255) of real;
        signal freq_lip6_FF : LIP6_FF :=(
                0 => 1.158529288680616,
                1 => 1.1619138154839776,
                2 => 1.165580298413822,
                3 => 1.1688350165364096,
                4 => 1.1715554943100219,
                5 => 1.17476535379232,
                6 => 1.1781864786654544,
                7 => 1.1817045761501765,
                8 => 1.1852834686905904,
                9 => 1.1887271675683846,
                10 => 1.192370385204907,
                11 => 1.1960831770219748,
                12 => 1.1999874412848694,
                13 => 1.2029465466041825,
                14 => 1.2059010483429685,
                15 => 1.2095259918573184,
                16 => 1.2145426488245616,
                17 => 1.2173182544049292,
                18 => 1.220500850842421,
                19 => 1.2244149913553207,
                20 => 1.2287068241940174,
                21 => 1.2320009870603101,
                22 => 1.235570289260339,
                23 => 1.2390111842571993,
                24 => 1.2436831721468773,
```

```
25 => 1.2468733631488767,
26 => 1.2505330673483355,
27 => 1.253897467747421,
28 => 1.2596196212733536,
29 => 1.2629287435914066,
30 => 1.2662070291671405,
31 => 1.2693876228323681,
32 => 1.2724860639718556,
33 => 1.2757796268555734,
34 => 1.2794896044862542,
35 => 1.2829719543345366,
36 => 1.2840728071753368,
37 => 1.2868789175125206,
38 => 1.2901613053859148,
39 => 1.2936371089596581,
40 => 1.2977121518541749,
41 => 1.3001354586434317,
42 => 1.303850695088519,
43 => 1.307463993148757,
44 => 1.312127532557108,
45 => 1.314712117269279,
46 => 1.317719440786665,
47 => 1.321283148608179,
48 => 1.326473554121573,
49 => 1.3292600540338883,
50 => 1.332660507176742,
51 => 1.3368217271560185,
52 => 1.3401909420702891,
53 => 1.3434174933632329,
54 => 1.3470645896732647,
55 => 1.3502951399314013,
56 => 1.3562329713082175,
57 => 1.359050026757597,
58 => 1.3621076168667016,
59 => 1.3655809493524187,
60 => 1.3710899165658858,
61 => 1.3740094797269263,
62 => 1.3776729057324474,
63 => 1.3811180851787722,
64 => 1.384438741655832,
65 => 1.387320346184784,
66 => 1.3912862759045827,
67 => 1.3944023664653583,
68 => 1.3971712895320547,
69 => 1.399534921031269,
70 => 1.4028919001977315,
71 => 1.4066779279400606,
72 => 1.408750194475159,
73 => 1.4119896260008757,
74 => 1.4155406160989327,
```

```
75 => 1.419131570315183,
76 => 1.4232253994523118,
77 => 1.4259395448861134,
78 => 1.428884996089662,
79 => 1.4325475160761673,
80 => 1.4372383201730478,
81 => 1.4404414076431684,
82 => 1.4432208216467025,
83 => 1.446967452849532,
84 => 1.4521141155446832,
85 => 1.4545832005210605,
86 => 1.4576599075221238,
87 => 1.4610315783690238,
88 => 1.4666643724866774,
89 => 1.469582843235469,
90 => 1.4728543605012076,
91 => 1.476233768651031,
92 => 1.4817599893916447,
93 => 1.4846801872934232,
94 => 1.487856871071663,
95 => 1.4913152250742683,
96 => 1.495037108570962,
97 => 1.4983302693038068,
98 => 1.5013464856518562,
99 => 1.5047470346026783,
100 => 1.507090576112938,
101 => 1.5094296755930922,
102 => 1.5128774890096014,
103 => 1.5163742911384816,
104 => 1.519157419936596,
105 => 1.5222235485648031,
106 => 1.5254740609402688,
107 => 1.5290912187930715,
108 => 1.5315489502990239,
109 => 1.5343843013499923,
110 => 1.53722251302808,
111 => 1.540970946874547,
112 => 1.5464723027036724,
113 => 1.54933907458462,
114 => 1.552092608931706,
115 => 1.5558354992479644,
116 => 1.560765740895693,
117 => 1.563535792358255,
118 => 1.5669533426083593,
119 => 1.570301843355766,
120 => 1.5756941392444808,
121 => 1.5789649421244576,
122 => 1.5816951284236927,
123 => 1.5851849545085084,
124 => 1.5910222808854399,
```

```
125 => 1.593850112743153,
126 => 1.5967820351869588,
127 => 1.6000433305724776,
128 => 1.6033404978268707,
129 => 1.6069179814199374,
130 => 1.6097505589855468,
131 => 1.6132573638226004,
132 => 1.6158460353580425,
133 => 1.6184714218781767,
134 => 1.6216013847840803,
135 => 1.625505295561103,
136 => 1.6279854941700768,
137 => 1.631259404273856,
138 => 1.6339443620031025,
139 => 1.6377403135490441,
140 => 1.6402417688848581,
141 => 1.643134401280881,
142 => 1.6457429688351052,
143 => 1.649641293012406,
144 => 1.652682684339663,
145 => 1.6554743415311034,
146 => 1.6586496310012598,
147 => 1.6625815757792394,
148 => 1.6675250274173834,
149 => 1.670523288970108,
150 => 1.6727650505363948,
151 => 1.67617989061416,
152 => 1.6816233691196935,
153 => 1.6854002242114506,
154 => 1.6878959247708433,
155 => 1.69105693134803,
156 => 1.6970512215467355,
157 => 1.700101935150643,
158 => 1.7025640968332038,
159 => 1.705823445041661,
160 => 1.7102160898716593,
161 => 1.7127647104372923,
162 => 1.7165921578383834,
163 => 1.719698998175245,
164 => 1.7219760770541673,
165 => 1.724832266698679,
166 => 1.728542175859351,
167 => 1.7316795417113533,
168 => 1.7341658964877315,
169 => 1.7367840202971954,
170 => 1.7398768561092062,
171 => 1.7431885744107175,
172 => 1.7464495143805695,
173 => 1.748740044633683,
174 => 1.7516373266119874,
```

```
175 => 1.7547416436346374,
176 => 1.7579533946016626,
177 => 1.7607323985991066,
178 => 1.7634683476619577,
179 => 1.7670064284632096,
180 => 1.77052899091156,
181 => 1.773412157038285,
182 => 1.7762871643844433,
183 => 1.7794243837504072,
184 => 1.7849098530913239,
185 => 1.7878875675118027,
186 => 1.7911383441790307,
187 => 1.7939773975419376,
188 => 1.7999207427935522,
189 => 1.8025841857704022,
190 => 1.805433701412458,
191 => 1.8083969986037288,
192 => 1.8129344820105476,
193 => 1.8157744688576055,
194 => 1.8189768557925353,
195 => 1.82212612808812,
196 => 1.8242847414613554,
197 => 1.8274061257552705,
198 => 1.8306826739840887,
199 => 1.8336998386464005,
200 => 1.836631849594167,
201 => 1.8393124718180048,
202 => 1.8421962991164508,
203 => 1.8454086775183556,
204 => 1.8485128358411138,
205 => 1.8514454687199352,
206 => 1.8541875264415157,
207 => 1.8576248635980055,
208 => 1.86163241337171,
209 => 1.8643085639350755,
210 => 1.8676644434374166,
211 => 1.8706868477516222,
212 => 1.8741804467494745,
213 => 1.87735350500781,
214 => 1.8798902458570385,
215 => 1.8826378191186128,
216 => 1.8864361725742764,
217 => 1.8895206819341211,
218 => 1.8927214027594426,
219 => 1.8956650963578055,
220 => 1.9008873331755688,
221 => 1.9039634016751113,
222 => 1.9066605014974766,
223 => 1.910417686650256,
224 => 1.914385225666489,
```

```
                225 => 1.9174316189527519,
                226 => 1.9208317948911102,
                227 => 1.9235660551932635,
                228 => 1.9261799507157757,
                229 => 1.928849566071833,
                230 => 1.9319561288231184,
                231 => 1.9349496447991903,
                232 => 1.9378225293494835,
                233 => 1.9402470092627501,
                234 => 1.943450600983126,
                235 => 1.9465664199462767,
                236 => 1.9500647466481588,
                237 => 1.952825495333771,
                238 => 1.9555199004827342,
                239 => 1.9587012864817889,
                240 => 1.9625178046759255,
                241 => 1.965796624185307,
                242 => 1.9683123836267145,
                243 => 1.9718863421158822,
                244 => 1.9749091207605898,
                245 => 1.9771611797409794,
                246 => 1.9804819521427279,
                247 => 1.9834790810730858,
                248 => 1.986575941363926,
                249 => 1.990278863754896,
                250 => 1.9935744824069893,
                251 => 1.9963807584446278,
                252 => 1.9991913681416159,
                253 => 2.0020573710045884,
                254 => 2.0054993220065284,
                255 => 2.0093235399725347
);

begin
--=======================================================
DECODER_A:
        with A select                   -- turning back to binary...
        DCO_IN(7 downto 5) <= "000" when "0000000",
        "001" when "0000001" ,
        "010" when "0000011" ,
        "011" when "0000111" ,
        "100" when "0001111" ,
        "101" when "0011111",
        "110" when "0111111" ,
        "111" when "1111111" ,
        "---" when others;
--=======================================================
DECODER_B:
        with B select
        DCO_IN(4 downto 2) <= "000" when "0000000" ,
```

```
        "001" when "0000001" ,
        "010" when "0000011" ,
        "011" when "0000111" ,
        "100" when "0001111" ,
        "101" when "0011111" ,
        "110" when "0111111" ,
        "111" when "1111111" ,
        "---" when others;
 --=====================================================
DECODER_C:
        with C select
        DCO_IN(1 downto 0) <= "00" when "000" ,
        "01" when "001" ,
        "10" when "011" ,
        "11" when "111" ,
        "--" when others;
--=====================================================
SWITCHER:
        with TYPEDCO select
        frequency_dco <= freq_lip6_TT(conv_integer(DCO_IN)) WHEN 1,
                freq_lip6_SS(conv_integer(DCO_IN)) WHEN 2,
                freq_lip6_FF(conv_integer(DCO_IN)) WHEN 3,
                freq_cea_TT(conv_integer(DCO_IN)) WHEN 4,
                freq_cea_SS(conv_integer(DCO_IN)) WHEN 5,
                freq_cea_FF(conv_integer(DCO_IN)) WHEN 6,
        1.0 when others;
--=============================================================================
PERIOD_SYNTHESIZER:
        -- calculating the period of oscillation
        period0 <= RESOLUTION * (1000000.0/frequency_dco);
--=============================================================================
PERIOD_CONTROLLED_OSCILLATOR:
        process(smp)
                variable initial: boolean := true;
                variable jitter: time := 0ns;
                variable jitter_prev: time := 0ns;
                variable wander: time := 0ns;
                variable period: time := 1ns;
                variable s1, s2, s3, s4: positive;
                variable x1, x2, x3, x4, randvar1, randvar2: real := 0.0;
        begin
                if initial then
                 period := period0;              --adjust the next period
                 uniform(s1, s2, x1);   --add Gaussian-distributed jitter
                 uniform(s1, s2, x2);
                 randvar1 := sqrt(-2.0*log(x1))*cos(2.0*MATH_PI*x2);
                 jitter := randvar1 * DCO_JRMS;
                 period := period+jitter-jitter_prev;
                 jitter_prev := jitter;
                 uniform(s3, s4, x3);   --add Gaussian-distributed wander
```

```
                uniform(s3, s4, x4);
                randvar2 := sqrt(-2.0*log(x3))*cos(2.0*MATH_PI*x4);
                wander := randvar2 * DCO_WRMS;
                period := period+wander;
                smp <= not smp after period;
                CLK_TMP <= '1','0' after period/2;--temp internal clk 50% duty cycl
               else
                period := period0+DCO_INIT_DELAY;       -- CURRENTLY NOT IN USE
                CLK_TMP <= '0';                                 -- CURRENTLY NOT IN
                smp <= '1';                             -- first transition
                initial := false;                       -- CURRENTLY NOT IN USE
               end if;
        end process;
--=============================================================================
        enable <= '1' after 1ps;                        -- operational delay 4250ns
--=============================================================================
        CLK <= CLK_TMP and enable;                      -- output of high freq. cloc
--=============================================================================
DIVIDER_2_1:
        process(CLK_TMP)
        begin
                if CLK_TMP'event and CLK_TMP='1' then
                        Q1 <= D1;
                end if;
        D1 <= not Q1;
        end process;
--=====================================================
DIVIDER_2_2:
        process(Q1)
        begin
                if Q1'event and Q1='1' then
                        Q2 <= D2;
                end if;
        D2 <= not Q2;
        end process;
--=====================================================
        DIV <= Q2 and enable;   -- output after divider on 4
--=====================================================
DIVIDER_2_3:
        process(Q2)
        begin
                if Q2'event and Q2='1' then
                        Q3 <= D3;
                end if;
        D3 <= not Q3;
        end process;
--=====================================================
        DIV16 <= Q3 and enable; -- output after divider on 8
--=====================================================
end behavioral;
```

Listing A.2: VHDL model of the sign detector

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.MATH_REAL.all;
use IEEE.NUMERIC_STD.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;


entity BB is
        port (
                REF, CLK, RESET: in STD_LOGIC;
                SIGN, nSIGN, TDCIN, TDCCLK: out STD_LOGIC
                );
end BB;

architecture arch_BB of BB is
---- generating of the constants and signals ------------------------------
        constant VCC_CONSTANT : STD_LOGIC := '1';
        signal Ao, Bo, D, Q1, A_S, Q2, B_R, A_first, B_first, nQ3, Q3 : STD_LOGIC;
        signal NET1, NET2, OUT_C, RESET_LOCAL: STD_LOGIC;
        signal rnd_vec: std_logic_vector(1 downto 0);
begin
        D <= VCC_CONSTANT;              -- high level to the D-inputs of the latches
        --====================================================================
        D_FLIP_FLOP_1:                                                  -- input
        Q1 <= '0' when RESET_LOCAL='1' else                             -- latch
        D  when (REF'event and REF='1') else Q1;        -- for REF
        A_S <= not Q1;                                  --
        --====================================================================
        D_FLIP_FLOP_2:                                                  -- input
        Q2 <= '0' when RESET_LOCAL='1' else                             -- latch
        D  when (CLK'event and CLK='1') else Q2;        -- for CLK
        B_R <= not Q2;                                  --
        --====================================================================
        RS_FLIP_FLOP:                                   -- INPUT SR FLIP-FLOP on NOR gates
                process (A_S,B_R)
                -- variables for pseudo-random number generator
                variable seed1, seed2: positive;
                variable rand: real;
                variable int_rand: integer range 0 to 1;
                variable rnd: STD_LOGIC;
        begin
                --====================================================================
                RANDOM_NUMBER_GENERATOR:        -- pseudo-random number generator
                UNIFORM(seed1, seed2, rand);
                int_rand := INTEGER(TRUNC(rand*2.0));
                rnd_vec <= CONV_STD_LOGIC_VECTOR(int_rand,2);
                rnd := rnd_vec(0);
```

```
        --=============================================================
        -- state table of the two NOR elements
        if (A_S = '0') and (B_R = '0') and (Bo = Ao) then
                Ao <= rnd;       -- assignment of the random number
                Bo <= not rnd; -- in case of arriving of two '0'
        elsif (A_S = '0') and (B_R = '0') then
                Ao <= Ao;
                Bo <= Bo;
        elsif (A_S = '1') and (B_R = '1') then
                Ao <= '0';
                Bo <= '0';
        elsif A_S = '1'  then
                Ao <= '1';
                Bo <= '0';
        elsif B_R = '1'  then
                Bo <= '1';
                Ao <= '0';
        end if ;
end process;
--=============================================================
FILTER:  -- Discription of metastability filter
        process (Ao,Bo)
begin
        -- state table of the filter
        if (Ao = '0') and (Bo = '0') then
                A_first <= '1';
                B_first <= '1';
        elsif (Ao = '0') and (Bo = '1') then
                A_first <= '1';
                B_first <= '0';
        elsif (Ao = '1') and (Bo = '0') then
                A_first <= '0';
                B_first <= '1';
        elsif  (Ao = '1') and (Bo = '1') then
                A_first <= '1';
                B_first <= '1';
        end if;
end process;
--=============================================================
SR_FLIP_FLOP: -- store flip-flop
nQ3 <= not(B_first and Q3);
Q3 <= not(A_first and nQ3);
--=============================================================
LOGIC_BLOCK:  -- logic NOR and OR elements
NET1 <= (not(A_first or nQ3)) or (not(B_first or Q3));
--=============================================================
C_ELEMENT:       -- C-element
OUT_C <= '1' when (NET1='1' and Q1='1' and Q2='1') else
'0' when (NET1='0' and Q1='0' and Q2='0') else OUT_C;
--=============================================================
```

```
        RESET_LOGIC:
        RESET_LOCAL <= OUT_C or (not RESET); -- reset OR element
        --=================================================================
        TDCIN <= not(Q1 nor Q2);          -- generating of the measure bit
        TDCCLK <= not(Q1 nand Q2) ;
        nSIGN <= Q3;
        SIGN <= nQ3;

        --=================================================================
        end arch_BB;
```

Listing A.3: VHDL model of the TDC

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.MATH_REAL.all;
use IEEE.NUMERIC_STD.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;


entity TDC_LIP6 is
        port (
                TDCCLK, TDCIN, TDCRES: in STD_LOGIC;
                S : out STD_LOGIC_VECTOR (6 downto 0)
                );
end TDC_LIP6;

architecture behavioral of TDC_LIP6 is
---- generating of the constants and signals ----------------------------
        signal Q_TDC: STD_LOGIC_VECTOR (6 downto 0);    -- latch signal in TDC
        signal tau: time:=20ps;
begin
        --=================================================================

                Q_TDC(0) <= TDCIN after tau;
                Q_TDC(1) <= Q_TDC(0) after tau;
                Q_TDC(2) <= Q_TDC(1) after tau;
                Q_TDC(3) <= Q_TDC(2) after tau;
                Q_TDC(4) <= Q_TDC(3) after tau;
                Q_TDC(5) <= Q_TDC(4) after tau;
                Q_TDC(6) <= Q_TDC(5) after tau;

        process (TDCCLK, TDCRES)
        begin
                if TDCRES='0' then
                        S <= "0000000";
                elsif (TDCCLK'event and TDCCLK='1') then
                        S(6 downto 0) <= Q_TDC(6 downto 0);
                end if;
        end process;
```

```
end behavioral;
```

Listing A.4: VHDL model of the digital processing block

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.MATH_REAL.all;
use IEEE.STD_LOGIC_ARITH.all;

entity FILTER_V2 is
   port (
         CLKFLT, RESFLT : in STD_LOGIC;
         ERRORIN1, ERRORIN2, ERRORIN3, ERRORIN4 : in std_logic_vector (3 downto 0);
         COEFS : in std_logic_vector (19 downto 0);
         A : out std_logic_vector (6 downto 0);
         B : out std_logic_vector (6 downto 0);
         C : out std_logic_vector (2 downto 0)
         );
end FILTER_V2;

architecture behavioral_8bits_v2_4 of FILTER_V2 is
        -- Signals for block of coefficients programing
         signal kintI: integer range 0 to 2**8-1;-- Kint and Kprop are never "0"
         signal ERROR1, ERROR2, ERROR3, ERROR4: integer range -2**3 to 2**3-1;
         signal adder_in_1, adder_in_3: integer range -2**5 to 2**5-1;   --6 bits
         signal adder_in_2, adder_in_4 : integer range -2**5 to 2**5-1;  --5 bits
         signal TOTAL_ERROR8: integer range -2**7 to 2**7-1;             --8 bits
         signal TOTAL_ERROR8_vector: std_logic_vector(7 downto 0);       --8 bits
         signal AVG_ERROR6: integer range -2**5 to 2**5-1;               --6 bits
         signal AVG_ERROR6_DELAYED: integer range -2**5 to 2**5-1;
         signal AVG_ERROR6_vector: std_logic_vector(5 downto 0);         --6 bits
         signal AVG_ERROR8: integer range -2**7 to 2**7-1;               --8 bits
         signal AVG_ERROR8_vector: std_logic_vector(7 downto 0);         --8 bits
         signal v_xi_K_prop: integer range -2**5 to 2**5-1;              --6 bits
         signal v_xi_K_int: integer range -2**13 to 2**13-1;             --14 bits
        -- integral path gives a 8-bit result, the devider is 2^12,
        -- thus 20 bits for integrator
         signal s_v_xi, s_v_xi_m1: integer range -2**19 to 2**19-1:=0;
         signal numKW, kp : std_logic_vector (1 downto 0);

         signal PIOUTI : integer range 0 to 2**8-1;
         signal PIOUT_BIN : std_logic_vector (7 downto 0);

        -- Signals for DCO decoder
         signal COL1 : STD_LOGIC;
         signal COL2 : STD_LOGIC;
         signal COL3 : STD_LOGIC;
         signal ROW1 : STD_LOGIC;
         signal ROW2 : STD_LOGIC;
         signal ROW3 : STD_LOGIC;
         signal ROW4 : STD_LOGIC;
```

```
        signal ROW5 : STD_LOGIC;
        signal ROW6 : STD_LOGIC;
        signal ROW7 : STD_LOGIC;
        signal A_NONDELAYED : std_logic_vector (6 downto 0);
        signal B_NONDELAYED : std_logic_vector (6 downto 0);
        signal C_NONDELAYED : std_logic_vector (2 downto 0);
begin
--Converting the std_logic input error to integer number
        ERROR1 <= conv_integer(signed(ERRORIN1));
        ERROR2 <= conv_integer(signed(ERRORIN2));
        ERROR3 <= conv_integer(signed(ERRORIN3));
        ERROR4 <= conv_integer(signed(ERRORIN4));
        kintI <= conv_integer(unsigned(COEFS(15 downto 8)));
--============================== ARITHMETIC BLOCK ============================
ARITH: -- input weightening and addition with averaging
   adder_in_1 <= 0 when (COEFS(7)='0' and COEFS(6)='0') else
                 ERROR1 when (COEFS(7)='0' and COEFS(6)='1') else
                 ERROR1*2 when (COEFS(7)='1' and COEFS(6)='0') else
                 ERROR1*4 when (COEFS(7)='1' and COEFS(6)='1') else
                 0;
   adder_in_2 <= 0 when (COEFS(5)='0' and COEFS(4)='0') else
                 ERROR2 when (COEFS(5)='0' and COEFS(4)='1') else
                 ERROR2*2 when (COEFS(5)='1' and COEFS(4)='0') else
                 ERROR2*4 when (COEFS(5)='1' and COEFS(4)='1') else
                 0;
   adder_in_3 <= 0 when (COEFS(3)='0' and COEFS(2)='0') else
                 ERROR3 when (COEFS(3)='0' and COEFS(2)='1') else
                 ERROR3*2 when (COEFS(3)='1' and COEFS(2)='0') else
                 ERROR3*4 when (COEFS(3)='1' and COEFS(2)='1') else
                 0;
   adder_in_4 <= 0 when (COEFS(1)='0' and COEFS(0)='0') else
                 ERROR4 when (COEFS(1)='0' and COEFS(0)='1') else
                 ERROR4*2 when (COEFS(1)='1' and COEFS(0)='0') else
                 ERROR4*4 when (COEFS(1)='1' and COEFS(0)='1') else
                 0;
--================== FOUR INPUT ADDER AND DIVIDER ======================
-- Adding errors
        TOTAL_ERROR8 <=(adder_in_1+adder_in_2+adder_in_3+adder_in_4);
        TOTAL_ERROR8_vector <= conv_std_logic_vector(TOTAL_ERROR8,8);
        numKW <= COEFS(19 downto 18);
        process(numKW, TOTAL_ERROR8)
        begin
        case numKW is
                when "00" => AVG_ERROR8 <= 0;
                when "01" => AVG_ERROR8 <= TOTAL_ERROR8; -- one input
                when "10" => AVG_ERROR8 <= TOTAL_ERROR8/2; -- two inputs
            when others => AVG_ERROR8 <= TOTAL_ERROR8/4; -- three or four inputs
        end case;
        end process;
        AVG_ERROR8_vector(7 downto 0) <= conv_std_logic_vector(AVG_ERROR8,8);
```

```
        AVG_ERROR6 <= conv_integer(signed(AVG_ERROR8_vector(5 downto 0)));
        -- AVG_ERROR8 is never larger than 6 bits indeed
        process(RESFLT,CLKFLT)
        begin
                if RESFLT='0' then
                        AVG_ERROR6_DELAYED <= 0;
                elsif (CLKFLT'event and CLKFLT='1') then
                        AVG_ERROR6_DELAYED <= AVG_ERROR6;
                end if;
        end process;
--================== PROPORTIONAL-INTEGRAL FILTER ======================
FILTER:-- filtering :
-- proportional path:
        kp <= COEFS(17 downto 16);
        process(kp, AVG_ERROR6)
        begin
        case kp is
                when "00" => v_xi_K_prop <= AVG_ERROR6;
                when "01" => v_xi_K_prop <= AVG_ERROR6/2;
                when "10" => v_xi_K_prop <= AVG_ERROR6/4;
            when others => v_xi_K_prop <= 0;
        end case;
        end process;

-- integral path:
-- adding and multiplying:
        v_xi_K_int <= AVG_ERROR6_DELAYED * kintI;
        s_v_xi <= v_xi_K_int + s_v_xi_m1;
-- first delay rising edge
-- latches:
        process(RESFLT,CLKFLT)
        begin
                if RESFLT='0' then
                        s_v_xi_m1 <= 0;
                elsif (CLKFLT'event and CLKFLT='1') then
                        s_v_xi_m1 <= s_v_xi;
                end if;
        end process;


        PIOUTI <=s_v_xi/2**12 + v_xi_K_prop + 128;
        PIOUT_BIN <= conv_std_logic_vector(PIOUTI,8);

--============================== DCO A-, B-, C- channel ENCODERS ========
----  Component instantiations  ----
        C_NONDELAYED(0) <= PIOUT_BIN(1) or PIOUT_BIN(0);
        C_NONDELAYED(1) <= PIOUT_BIN(1);
        C_NONDELAYED(2) <= PIOUT_BIN(1) and PIOUT_BIN(0);
-- ------------------------------------------------------------------------

        B_NONDELAYED(0) <= PIOUT_BIN(4) or PIOUT_BIN(3) or PIOUT_BIN(2);
```

```
        B_NONDELAYED(1) <= PIOUT_BIN(4) or PIOUT_BIN(3);
        B_NONDELAYED(2) <= PIOUT_BIN(4) or (PIOUT_BIN(3) and PIOUT_BIN(2));
        B_NONDELAYED(3) <= PIOUT_BIN(4);
        B_NONDELAYED(4) <= PIOUT_BIN(4) and (PIOUT_BIN(3) or PIOUT_BIN(2));
        B_NONDELAYED(5) <= PIOUT_BIN(4) and PIOUT_BIN(3);
        B_NONDELAYED(6) <= PIOUT_BIN(4) and PIOUT_BIN(3) and PIOUT_BIN(2);
-------------------------------------------------------------------------------
        A_NONDELAYED(0) <= PIOUT_BIN(7) or PIOUT_BIN(6) or PIOUT_BIN(5);
        A_NONDELAYED(1) <= PIOUT_BIN(7) or PIOUT_BIN(6);
        A_NONDELAYED(2) <= PIOUT_BIN(7) or (PIOUT_BIN(6) and PIOUT_BIN(5));
        A_NONDELAYED(3) <= PIOUT_BIN(7);
        A_NONDELAYED(4) <= PIOUT_BIN(7) and (PIOUT_BIN(6) or PIOUT_BIN(5));
        A_NONDELAYED(5) <= PIOUT_BIN(7) and PIOUT_BIN(6);
        A_NONDELAYED(6) <= PIOUT_BIN(7) and PIOUT_BIN(6) and PIOUT_BIN(5);

        process(RESFLT,CLKFLT)
        begin
                if RESFLT='0' then
                        A <= "0001111";
                        B <= "0000000";
                        C <= "000";
                elsif (CLKFLT'event and CLKFLT='1') then
                        A <= A_NONDELAYED;
                        B <= B_NONDELAYED;
                        C <= C_NONDELAYED;
                end if;
        end process;

end behavioral_8bits_v2_4;
```

Appendix B

VHDL models for built-in test circuit

This appendix presents the RTL-level VHDL model of the built-in test circuit block: ProcessUnit and the functional model of clock signals generator which generates two clock signals to be measured. Known static phase error and Gaussian distributed dynamic errors can be injected between two clocks. The distribution of added static and dynamic phase errors is compared with measurement results of simulations to evaluate performance of the methodology presented in Chapter 4.

Listing B.1: Generation of two clock signals with jitters and skew injection

```
--
-- Questa ADMS model for test generator_clks adms_vhdl
--
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.MATH_REAL.all;
use IEEE.NUMERIC_STD.all;
use std.textio.all;
use ieee.std_logic_textio.all;

ENTITY generator_clks IS
    PORT (
        clk1, clk2, enable: out std_logic
    );
END;

ARCHITECTURE arch_generator_clks OF generator_clks IS
    signal smp: bit := '0';
    signal period0: time := 4ns;
        signal skew_pos: time := 0ps;
        signal skew_neg: time := 0ps;
        signal clk1_tmp, clk2_tmp: std_logic;

file file_out: text open write_mode is "./generator_clks_jitter_BIT_TOP.txt";
```

```
BEGIN

process(smp)
        variable initial: boolean := true;
        variable jitter: time := 0ns;
        variable jitter_prev: time := 0ns;
        variable wander: time := 0ns;
        variable period: time := 4ns;
        variable s1, s2, s3, s4: positive;
        variable x1, x2, x3, x4, randvar1, randvar2: real := 0.0;
                variable JRMS: time :=  32ps;
                variable BUF:LINE;
                variable err_phase: time := 0ns;
        begin
        if initial then
            period := period0;              --adjust the next period
            uniform(s1, s2, x1);            --add Gaussian-distributed jitter
            uniform(s1, s2, x2);
            randvar1 := sqrt(-2.0*log(x1))*cos(2.0*MATH_PI*x2);
            jitter := randvar1 * JRMS;

                        err_phase := jitter+skew_pos-skew_neg;
                        write(BUF,err_phase);
                    writeline(file_out,BUF);

            period := period+2*jitter;

            smp <= not smp after period;
            clk1_tmp <= '0','1' after period0/2;    --temp internal clk 50% duty cycle
                        clk2_tmp <= '0','1' after period/2;
         else
            smp <= '1';                 -- first transition
            initial := false;            -- CURRENTLY NOT IN USE
        end if;
    end process;
--================================================================================
    enable <= '0', '1' after 30ns;              -- operational delay 30ns
--================================================================================
    clk1 <= clk1_tmp after skew_neg;
        clk2 <= clk2_tmp after skew_pos;            -- output of high freq. clock


END;
```

Listing B.2: VHDL model of the digital process unit in test circuit

```
--
-- Questa ADMS model for test ProcessUnit adms_vhdl
--
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.MATH_REAL.all;
use IEEE.NUMERIC_STD.all;


ENTITY ProcessUnit IS
    PORT (
                enable: in std_logic;
        clk2, clk2_b: in std_logic:='0';
        dataIN: in std_logic:='0';
        cntOUT: out std_logic_vector(7 downto 0)
    );
END;

ARCHITECTURE arch_ProcessUnit OF ProcessUnit IS
signal dataINby2 : std_logic:='0';
signal Q2,Q3,Q4,Q5 : std_logic:='0';
signal clk2Bar : std_logic;
signal s_nxor : std_logic;
signal overflow, read : std_logic:='0';
signal cnt1, cnt2 : integer range 0 to 2**8-1 := 0;                    --8 bits counters

BEGIN
-- divide input signal frequency by 2
process(dataIN)
begin
        if(rising_edge(dataIN)) then
                dataINby2 <= not dataINby2;
        end if;
end process;

-- R2 and R4
process(clk2, enable)
begin
        if(enable='1') then
                if(rising_edge(clk2)) then
                        Q2 <= dataINby2;
                        Q4 <= Q3;
                end if;
        end if;
end process;

--clk2Bar <= not clk2;
--R3 and R5
process(clk2_b, enable)
begin
        if(enable='1') then
                if(rising_edge(clk2_b)) then
                        Q3 <= Q2;
```

```
                        Q5 <= Q4;
                end if;
        end if;
end process;

s_nxor <= not(Q5 xor Q3);


--counter 1
process(clk2, enable, overflow)
begin
        if(enable='1') then
                if (overflow='1') then
                        cnt1 <= 0;
                elsif(rising_edge(clk2)) then
                        if(s_nxor = '1') then
                                cnt1 <= cnt1 + 1;
                        else
                                cnt1 <= cnt1;
                        end if;
                end if;
        else
                cnt1 <= 0;
        end if;
end process;


--counter 2
process(clk2, enable)
begin
        if(enable='1') then
                if(rising_edge(clk2)) then
                        if(cnt2 = 2**8-2) then
                                read <= '1';
                                cnt2 <= cnt2 + 1;
                        elsif(cnt2 = 2**8-1) then
                                read <= '0';
                                overflow <= '1';
                                cnt2 <= 0;
                        else
                                overflow <= '0';
                                cnt2 <= cnt2 + 1;
                        end if;
                end if;
        else
                cnt2 <= 0;
        end if;
end process;

--Rs sampling for output
```

```
process(read,enable)
begin
        if(enable='1') then
                if(rising_edge(read)) then
                        cntOUT <= conv_std_logic_vector(cnt1,cntOUT'length);
                end if;
        end if;

end process;

END;
```

Appendix C

Matlab scripts

Matlab phase/gain margins plot

The Matlab script here presented is used during the step of defining specification of ADPLL. It allows observing the phase/gain margins of ADPLL and helping us estimate the optimal combination of PFD and DCO parameters.

Listing C.1: Script plots phase/gain margin of an ADPLL

```
clear all;
% This script allows to plot the linear frequency transmission characteristics
% of digital ADPLL modeled as discret time linear system.

% The sampling frequency of the filter. The filter is supposed to be cadanced
% with a fixed frequency. This is not exactly the case, since the filter clock
% signal is generated by the local DCO. However, in the locked mode, the
% behaviour of the DCO can be considered regular and the generated signal
% periodic. We consider that the nomianl output DCO frequency is 1.04 GHz. The
% filter is cadenced by this frequency divided by 4.
fs=260e+6
%the samplig period
Ts=1/fs;
%angular frequency
omega_s=fs*2*pi;

%The TDC resolution
tau_tdc=20e-12;
% The value of the gain of the phase-frequency detector, expressed in
% integer units/radian. This value is valid for
% the linear zone of the PFD characteristic, provided by the TDC. The PFD is
% supposed to increase its output value by 1 when the phase error is increased
% by 2 pi fs
Kpfd=1/(tau_tdc*2*pi*fs);

% The DCO gain is expressed in radian/(second*input integer units). Since the
% implemented DCO has 2*pi*1 MHz/integer units, the divided signal have a four
% time smaller gain :
```

```
Kdco=0.253e6*2*pi;  %DCO 3nd version

% The values chosen for the integral and proportional gains of the filter

Ki= 50/2^14;
Kp=31/2^5;

%number of additional delays
nret=2;

Hdco = tf([Ts*Kdco Ts*Kdco],[2 -2],Ts);
Hfilter = tf([Ts*Ki+2*Kp -Ts*Ki-2*Kp], [2 -2],Ts);
Hdelay = tf([1],[1 0 0],Ts);
Hopen = Hdco*Hfilter*Hdelay*Kpfd;
figure(4)
margin(Hopen)
```

Matlab control sequence generators

Here you can find the Matlab scripts used during the step of verification of the ADPLL and clock network. Each script returns data files, which have been included during the VHDL and Eldo simulations of the circuits.

Listing C.2: Script generates a programming sequence for the clock network

```
clc
clear

Ki=0.005;
Kp=1;

KW_coef =1;

if Kp==1     % 0->1; 1->0.5; 2->0.25
        Kprop=0
elseif Kp==0.5
        Kprop=1
elseif Kp==0.25
        Kprop=2
end

Kint=round(Ki*(2^12))

KW1(1)=1; KW2(1)=1; KW3(1)=0; KW4(1)=1; Kint_num(1)=Kint;
Kprop_num(1)=Kprop; numKW(1)=3;
KW1(2)=1; KW2(2)=1; KW3(2)=0; KW4(2)=1; Kint_num(2)=Kint;
Kprop_num(2)=Kprop; numKW(2)=3;
KW1(3)=1; KW2(3)=1; KW3(3)=0; KW4(3)=1; Kint_num(3)=Kint;
```

```
Kprop_num(3)=Kprop; numKW(3)=3;
KW1(4)=1; KW2(4)=1; KW3(4)=0; KW4(4)=1; Kint_num(4)=Kint;
Kprop_num(4)=Kprop; numKW(4)=3;
KW1(5)=1; KW2(5)=1; KW3(5)=0; KW4(5)=1; Kint_num(5)=Kint;
Kprop_num(5)=Kprop; numKW(5)=3;
KW1(6)=1; KW2(6)=1; KW3(6)=0; KW4(6)=1; Kint_num(6)=Kint;
Kprop_num(6)=Kprop; numKW(6)=3;
KW1(7)=1; KW2(7)=1; KW3(7)=0; KW4(7)=1; Kint_num(7)=Kint;
Kprop_num(7)=Kprop; numKW(7)=3;
KW1(8)=1; KW2(8)=1; KW3(8)=0; KW4(8)=1; Kint_num(8)=Kint;
Kprop_num(8)=Kprop; numKW(8)=3;
KW1(9)=1; KW2(9)=1; KW3(9)=0; KW4(9)=1; Kint_num(9)=Kint;
Kprop_num(9)=Kprop; numKW(9)=3;
KW1(10)=1; KW2(10)=0; KW3(10)=0; KW4(10)=1; Kint_num(10)=Kint;
Kprop_num(10)=Kprop; numKW(10)=2;

for l=1:8
        KW1(10*l+1)=0; KW2(10*l+1)=1; KW3(10*l+1)=1; KW4(10*l+1)=1;
        Kint_num(10*l+1)=Kint; Kprop_num(10*l+1)=Kprop; numKW(10*l+1)=3;
        KW1(10*l+2)=1; KW2(10*l+2)=1; KW3(10*l+2)=1; KW4(10*l+2)=1;
        Kint_num(10*l+2)=Kint; Kprop_num(10*l+2)=Kprop; numKW(10*l+2)=4;
        KW1(10*l+3)=1; KW2(10*l+3)=1; KW3(10*l+3)=1; KW4(10*l+3)=1;
        Kint_num(10*l+3)=Kint; Kprop_num(10*l+3)=Kprop; numKW(10*l+3)=4;
        KW1(10*l+4)=1; KW2(10*l+4)=1; KW3(10*l+4)=1; KW4(10*l+4)=1;
        Kint_num(10*l+4)=Kint; Kprop_num(10*l+4)=Kprop; numKW(10*l+4)=4;
        KW1(10*l+5)=1; KW2(10*l+5)=1; KW3(10*l+5)=1; KW4(10*l+5)=1;
        Kint_num(10*l+5)=Kint; Kprop_num(10*l+5)=Kprop; numKW(10*l+5)=4;
        KW1(10*l+6)=1; KW2(10*l+6)=1; KW3(10*l+6)=1; KW4(10*l+6)=1;
        Kint_num(10*l+6)=Kint; Kprop_num(10*l+6)=Kprop; numKW(10*l+6)=4;
        KW1(10*l+7)=1; KW2(10*l+7)=1; KW3(10*l+7)=1; KW4(10*l+7)=1;
        Kint_num(10*l+7)=Kint; Kprop_num(10*l+7)=Kprop; numKW(10*l+7)=4;
        KW1(10*l+8)=1; KW2(10*l+8)=1; KW3(10*l+8)=1; KW4(10*l+8)=1;
        Kint_num(10*l+8)=Kint; Kprop_num(10*l+8)=Kprop; numKW(10*l+8)=4;
        KW1(10*l+9)=1; KW2(10*l+9)=1; KW3(10*l+9)=1; KW4(10*l+9)=1;
        Kint_num(10*l+9)=Kint; Kprop_num(10*l+9)=Kprop; numKW(10*l+9)=4;
        KW1(10*l+10)=1; KW2(10*l+10)=0; KW3(10*l+10)=1; KW4(10*l+10)=1;
        Kint_num(10*l+10)=Kint; Kprop_num(10*l+10)=Kprop; numKW(10*l+10)=3;
end

KW1(91)=0; KW2(91)=1; KW3(91)=1; KW4(91)=0; Kint_num(91)=Kint;
Kprop_num(91)=Kprop; numKW(91)=2;
KW1(92)=1; KW2(92)=1; KW3(92)=1; KW4(92)=0; Kint_num(92)=Kint;
Kprop_num(92)=Kprop; numKW(92)=3;
KW1(93)=1; KW2(93)=1; KW3(93)=1; KW4(93)=0; Kint_num(93)=Kint;
Kprop_num(93)=Kprop; numKW(93)=3;
KW1(94)=1; KW2(94)=1; KW3(94)=1; KW4(94)=0; Kint_num(94)=Kint;
Kprop_num(94)=Kprop; numKW(94)=3;
KW1(95)=1; KW2(95)=1; KW3(95)=1; KW4(95)=0; Kint_num(95)=Kint;
Kprop_num(95)=Kprop; numKW(95)=3;
KW1(96)=1; KW2(96)=1; KW3(96)=1; KW4(96)=0; Kint_num(96)=Kint;
```

```
Kprop_num(96)=Kprop; numKW(96)=3;
KW1(97)=1; KW2(97)=1; KW3(97)=1; KW4(97)=0; Kint_num(97)=Kint;
Kprop_num(97)=Kprop; numKW(97)=3;
KW1(98)=1; KW2(98)=1; KW3(98)=1; KW4(98)=0; Kint_num(98)=Kint;
Kprop_num(98)=Kprop; numKW(98)=3;
KW1(99)=1; KW2(99)=1; KW3(99)=1; KW4(99)=1; Kint_num(99)=Kint;
Kprop_num(99)=Kprop; numKW(99)=3;
KW1(100)=1; KW2(100)=0; KW3(100)=1; KW4(100)=0; Kint_num(100)=Kint;
Kprop_num(100)=Kprop; numKW(100)=2;

fid_vhdl=fopen('./param100.dat', 'w'); %% file to write binary data

order_of_loading=[91 92 93 94 95 96 97 98 99 100 90 89 88 87 86 85
84 83 82 81 71 72 73 74 75 76 77 78 79 80 70 69 68 67 66 65 64 63
62 61 51 52 53 54 55 56 57 58 59 60 50 49 48 47 46 45 44 43 42 41
31 32 33 34 35 36 37 38 39 40 30 29 28 27 26 25 24 23 22 21 11 12
13 14 15 16 17 18 19 20 10 9 8 7 6 5 4 3 2 1];
M = length(order_of_loading);
for count=1:M
i=order_of_loading(count)
        send_to_file(KW4(i), 2, 0, fid_vhdl);
        send_to_file(KW3(i), 2, 0, fid_vhdl);
        send_to_file(KW2(i), 2, 0, fid_vhdl);
        send_to_file(KW1(i), 2, 0, fid_vhdl);
        send_to_file(Kint_num(i), 8, 0, fid_vhdl);
        send_to_file(Kprop_num(i), 2, 0, fid_vhdl);
        send_to_file(numKW(i), 2, 0, fid_vhdl);
end
fprintf(1, 'Ding␣VHDL!\n');

Z=importdata('./param100.dat');    % loading data from file
fid_eldo=fopen('./param100_eldo', 'w'); %% file to write timing data
N = length(Z);

Fsck = 200;            % data send rate, MHz
tsda = 1000/Fsck;          % data send period, ns
tr = 0.03;
vdd = 1.1;               % vdd voltage, V
bit0 = Z(1);
fprintf(fid_eldo, 'V110␣sda␣0␣PWL␣(␣0␣%1.2g␣%2.4gn␣%1.2g\n',
bit0*vdd, tsda-tr, bit0*vdd);

for p = 2:N
   bit = Z(p);
   fprintf(fid_eldo, '+%4.6gn␣%1.2g␣%4.6gn␣%1.2g\n', (p-1)*tsda,
    bit*vdd, p*tsda-tr, bit*vdd);
end

fprintf(fid_eldo, ')\n');
fprintf(fid_eldo, 'V111␣reset␣0␣PWL␣(␣0␣1.1␣1n␣1.1␣1.03n␣0␣3n
```

```
␣0␣3.03n␣1.1␣%4.6gn␣1.1␣%4.6gn␣0␣%4.6gn␣0␣%4.6gn␣1.1)\n', 2+p*tsda,
 2+p*tsda+tr, 2+p*tsda+tr+1, 2+p*tsda+tr+1+tr);
fprintf(fid_eldo, 'V112␣ntwrk_en␣0␣PWL␣(␣0␣0␣%4.6gn␣0␣%4.6gn
␣0␣%4.6gn␣0␣%4.6gn␣1.1)\n', p*tsda, p*tsda+tr, p*tsda+tr+1, p*tsda+tr+1+tr);

fprintf(fid_eldo, 'V113␣sck␣0␣PULSE␣(␣%1.2g␣0␣0␣%2.4gn␣%2.4gn␣%3.4gn␣%3.4gn␣)\n',
vdd, tr, tr, tsda/2, tsda);

fprintf(fid_eldo, 'V114␣enable_BIT1␣0␣PWL␣(␣0␣0␣10n␣0␣10.03n␣1.1␣)\n');
fprintf(fid_eldo, 'V115␣enable_BIT2␣0␣PWL␣(␣0␣0␣10n␣0␣10.03n␣1.1␣)\n');
fprintf(fid_eldo, 'V116␣enable_BIT3␣0␣PWL␣(␣0␣0␣10n␣0␣10.03n␣1.1␣)\n');
fprintf(fid_eldo, 'V117␣enable_BIT4␣0␣PWL␣(␣0␣0␣10n␣0␣10.03n␣1.1␣)\n');
fprintf(1, 'Ding␣ELDO!\n');
```

Listing C.3: Integer to binary subscript

```
function y=integer_to_binary(x, n, s)
% x: la valeur, n: nombre de bit, s: signe ou non

x_fi=fi(x,s,n,0);
for i=1:n
        y(i)=int16(bitget(x_fi,n-i+1));
end
end
```

Listing C.4: Script writes the sequence to file

```
function y=send_to_file(value, n, s, fid_vhdl)
value_bin=integer_to_binary(value,n,s);
for i=1:n
fprintf(fid_vhdl, '%i\n', value_bin(n-i+1));
end
y=1;
end
```

Matlab 3-D video creator for observing ADPLL network transient process

The Matlab script presented here is used to generate a 3-D video of phase error variation in an ADPLL network during a transient process.

Listing C.5: Script generates a 3-D video of phase error variation in an ADPLL network

```
clc
clear
%=========================================================================
%=== ANIMATION PARAMETERS ================================================
%=========================================================================
code_min = -2048;       % minimal highlighted code
```

```
code_max = 2047;      % maximal highlighted code
fps = 24;             % FPS in animation
N = 1;                % number of runs for animation
%=========================================================================
%=== LOADING INPUT WAVEFORMS ============================================
%=========================================================================
tic
fprintf(1, 'Loading␣waveforms...');
% loading data from file
Z=importdata('./Matlab/Visualization/list100.lst');
fprintf(1, 'DONE!\n');
toc
%=========================================================================
%=== EXTRACTING PARAMETERS OF NETWORK ====================================
%=========================================================================
M = length(Z);              % number of the samples
K = size(Z,2)-2;            % number of the nodes in network
N = sqrt(K);                    % size of the network N*N
X = nan(1,K);                     % initialize samples temporary array
Y = nan(N,N,K);                     % initialize data storage array
%=========================================================================
%=== CALCULATING NETWORK STATE MATRICE ===================================
%=========================================================================
fprintf(1, 'Matrice␣reshaping␣and␣rotating....');
tic
for p = 1:1:M
    for l = 3:K+2;
        X(1,l-2) = Z(p,l);
    end
    Y(:,:,p) = fliplr(rot90(reshape(X,N,N),3));
end
fprintf(1, 'DONE!\n');
toc
%=========================================================================
%=== SAVING STATES =======================================================
%=========================================================================
fprintf(1, 'Saving␣matrices....');
save('WAVES', 'Y', 'M');              % storing states in file
fprintf(1, 'DONE!\n');
%=========================================================================
%=== LOADING INPUT WAVEFORMS ============================================
%=========================================================================
fprintf(1, 'Loading␣matrices....')
load('WAVES', 'Y', 'M');
fprintf(1, 'DONE!\n')
%=========================================================================
%=== CREATING ANIMATION ==================================================
%=========================================================================
fprintf(1, 'Creating␣animation....')
tic
```

```
writerObj = VideoWriter('network100_with_border_without_perturbation_50us.avi');
open(writerObj);

clims = [code_min code_max];          % color range settings
caxis([code_min code_max]);
colorbar;
set(gca,'nextplot','replacechildren');
set(gcf,'Renderer','zbuffer');

for j = 1:1000:M
    set(gca,'CameraViewAngleMode','manual')
    colormap hsv
    axis([0 10 0 10 code_min code_max])
    surf(Y(:,:,j));
    F(j) = getframe;                  % storing frame
    writeVideo(writerObj,F(j));
end
close(writerObj);
fprintf(1, 'DONE!\n')
toc

fprintf(1, 'Saving␣frames....')
save('FRAMES', 'F');
fprintf(1, 'DONE!\n')
%movie(F,N,fps)                       % playing the animation
```

Appendix D

FPGA prototyping of the clocking network

Environment and measurement set

The Mentor Graphics AdvaceMS mixed simulator is used to perform behavioral mixed simulations of the developed code before porting it to Altera Quartus II environment, which has been extensively used in the design of the clocking network prototype.

The clocking network has been implemented onto EP2C70F672C6 chip. The clocking network core has been designed keeping in mind the spatial location criterion. This means that nodes of the network were spatially distributed over the FPGA area so to form an equivalent to the specified 2D mesh network.

The reference clock has been generated by external FLUKE PM 5136 Synthesized Function Generator. The output signals were captured and processed by LeCroy Waverunner oscilloscope with Mixed Signal Oscilloscope Option.

The FPGA chip is installed on *Cyclone II DSP Development Board*. Fig. D.1 depicts the functional diagram of this platform and Fig. D.2 top view of the board with description.

The measurement set is depicted in a Fig. D.3.

VHDL models

This section introduces the complete synthesizable VHDL models of the proposed DCO and TDC for FPGA implementation.

Listing D.1: VHDL model of the synthesizable DCO

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.MATH_REAL.all;
use IEEE.NUMERIC_STD.all;
```

```
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY nco_fpga IS
        PORT (
                CLK_fpga: in STD_LOGIC;
                FLT_OUT: in STD_LOGIC_VECTOR(9 downto 0);
                CLK_OUT: out std_logic
        );
END;

ARCHITECTURE rtl OF nco_fpga IS

---- generating of the constants and signals ----------------------------

        constant N: INTEGER:=1789;      -- max value of counter
        signal M: INTEGER:=1155;        -- medium value of counter
        signal sum:INTEGER range 0 to N:=512 ;
        signal FLT_OUT_integer : integer range 0 to 1023;
BEGIN

 FLT_OUT_integer <= conv_integer(FLT_OUT);
-- The period is equal to (N-FLI_OUT+1)*T_CLK_fpga
process(CLK_fpga,FLT_OUT) is
  begin
        if (CLK_fpga'event and CLK_fpga='1') then

                        if (sum < M) then
                                sum <= sum + 1;
                                CLK_OUT<='1';
                        elsif (sum < N) then
                                sum <= sum + 1;
                                CLK_OUT<='0';
                        else
                                sum <= FLT_OUT_integer;
                                M <= (FLT_OUT_integer+N)/2;
                                CLK_OUT<='1';
                        end if;
        end if;
 end process;
END rtl;
```

Listing D.2: VHDL model of the synthesizable TDC

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.MATH_REAL.all;       use IEEE.NUMERIC_STD.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.numeric_std.all;
```

```
entity tdc_fpga is
  port (
        ERR,SIGNE,CLK_fpga,RESET: in STD_LOGIC;
        ERREUR, NERREUR: out std_logic_vector (4 downto 0)
      );
end tdc_fpga;

architecture rtl of tdc_fpga is
---- generating of the constants and signals ----------------------------
  signal cpt_int: INTEGER range 0 to 15;
  signal CODER_OUT: integer range -16 to 15 ;
  signal count_reset: integer range 0 to 7;
begin
process(RESET,CLK_fpga,ERR,cpt_int) is
 begin
  if (RESET ='0') then
        cpt_int<=1;
        count_reset<=0;
  else
        if(CLK_fpga'event and CLK_fpga='1') then
              if(ERR = '0') then
                  if (count_reset>2) then
                        cpt_int<=1;
                        count_reset<=0;
                  else
                        cpt_int <=cpt_int;
                        count_reset<=count_reset+1;
                  end if;
              else
                  count_reset<=0;
                  if(cpt_int < 15) then
                        cpt_int<=cpt_int+1;
                  else cpt_int<=cpt_int;
                  end if;
              end if;
        end if;
  end if;
   end process;

process (ERR, RESET, SIGNE)
  begin
  if (RESET='0') then
        CODER_OUT<=1;
  elsif (ERR='0' and ERR'event) then
      if (SIGNE='0') then
        CODER_OUT <= -1*cpt_int;
      else
        CODER_OUT<=cpt_int;
      end if;
  end if;
```

```
end process;

--=============================================================
ERREUR<=std_logic_vector(to_signed(CODER_OUT, ERREUR'length));
NERREUR<=std_logic_vector(to_signed(-1*CODER_OUT, NERREUR'length));
end rtl;
```

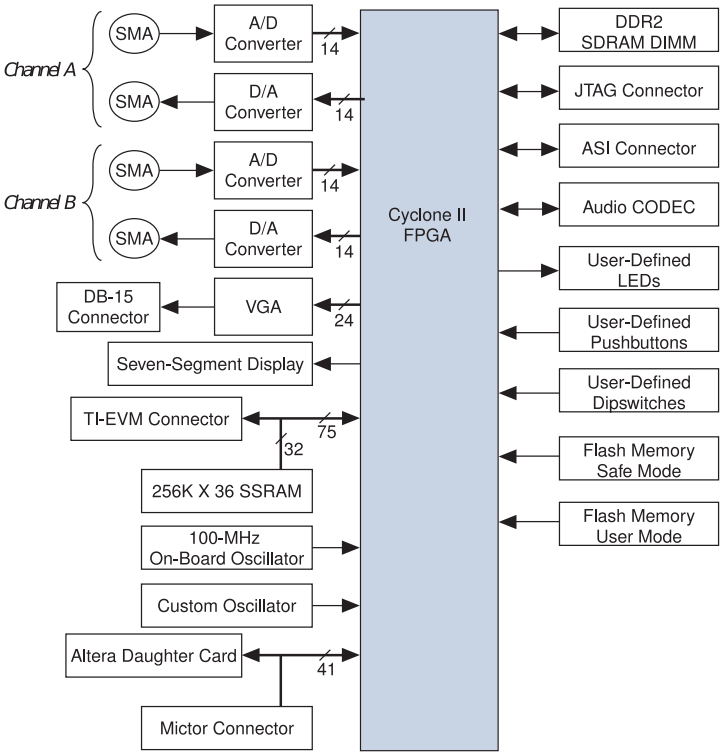


Figure D.1: **Functional diagram of Cyclone II DSP Development Board**

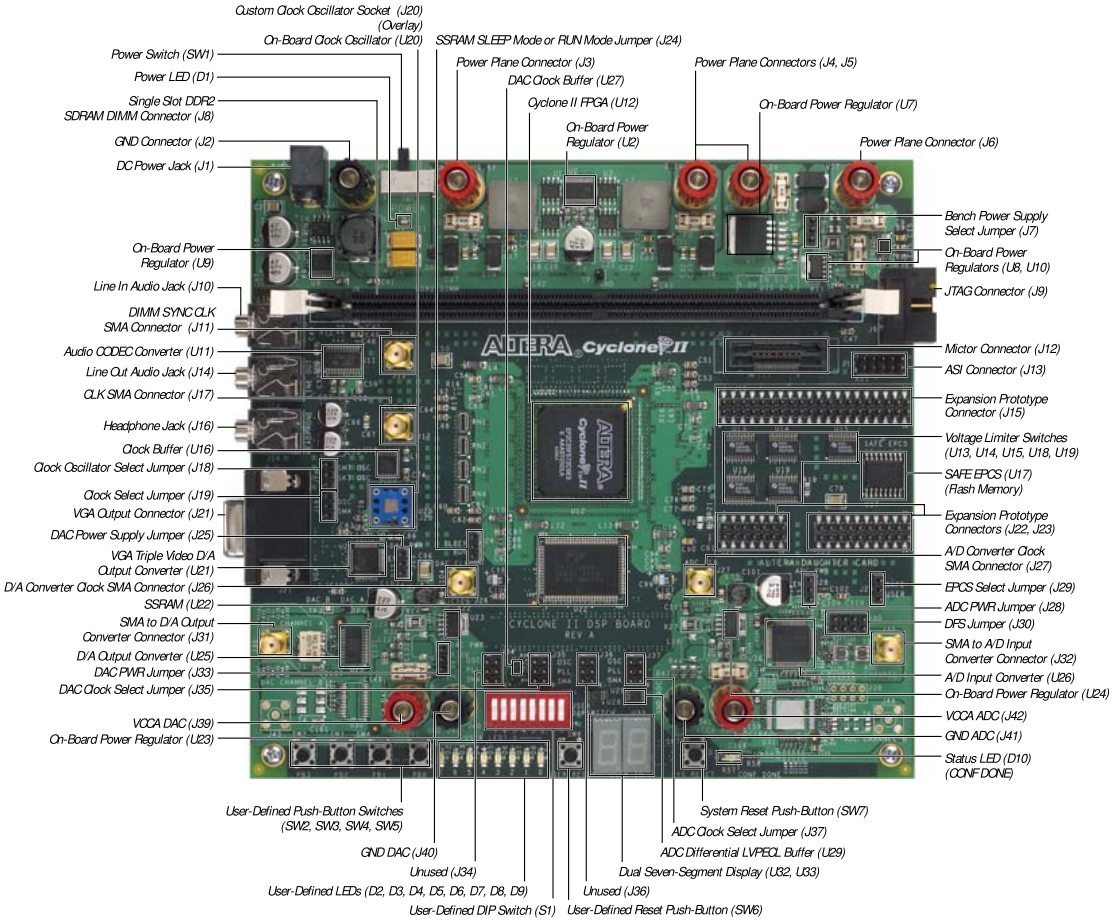


Figure D.2: **Top view of Cyclone II DSP Development Board**

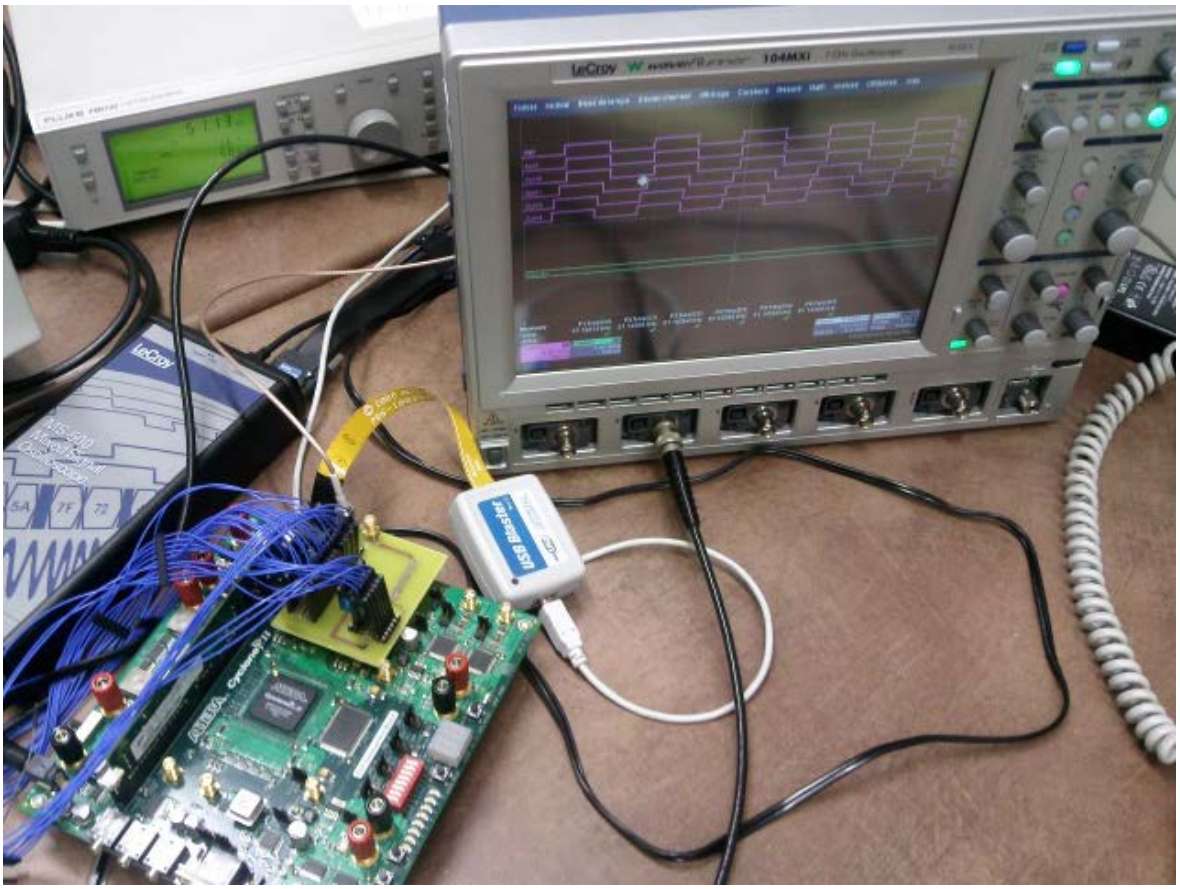


Figure D.3: **FPGA prototyping platform**: Cyclone II EP2C70 DSP Development Board and LeCroy Waverunner Digital Sampling Oscilloscope

Appendix E

Tcl script for automatic floorplan of network

Automatic floorplan of network

The Tcl script here presented is used for the automatic placement of the chip at the top level. The parametrable script allows a parallel design of the blocks and the global chip floorplanning: if the layout of a cell is modified, the chip floorplan can be immediately regenerated without extra work.

Listing E.1: Script for automatic floorplan of network

```
#placement of blocks in an ADPLL network
proc max {x y} {expr {$x>$y? $x: $y}}

set pitch 5.2
set pad_height 112
set pad_width 40

set pad_numberY 50
set pad_numberX 50

# distance between the core and the die boundary
set  delta_core_die_left [expr 39*$pitch+$pad_height]
set  delta_core_die_bottom [expr 39*$pitch+$pad_height]
set  delta_core_die_right [expr 40*$pitch+$pad_height]
set  delta_core_die_top   [expr 40*$pitch+$pad_height]

set NODE_width 153.000
set NODE_height 148.200

set NODEI_width 153.000
set NODEI_height 148.200

set BIT_width 170.000
set BIT_height 23.400
```

```
set PFD_width 51.000
set PFD_height 28.600

set BIT_core_gapX [expr 4*$pitch]
set BIT_core_gapY [expr 6*$pitch]

set NODE_instance_name  {{INODE_10_1 INODE_10_2 INODE_10_3 INODE_10_4 INODE_10_5\
                          INODE_10_6 INODE_10_7 INODE_10_8 INODE_10_9 INODE_10_10} \
                        {INODE_9_1 INODE_9_2 INODE_9_3 INODE_9_4 INODE_9_5 INODE_9_6\
                        INODE_9_7 INODE_9_8 INODE_9_9 INODE_9_10} \
                        {INODE_8_1 INODE_8_2 INODE_8_3 INODE_8_4 INODE_8_5 INODE_8_6\
                        INODE_8_7 INODE_8_8 INODE_8_9 INODE_8_10} \
                        {INODE_7_1 INODE_7_2 INODE_7_3 INODE_7_4 INODE_7_5 INODE_7_6\
                        INODE_7_7 INODE_7_8 INODE_7_9 INODE_7_10} \
                        {INODE_6_1 INODE_6_2 INODE_6_3 INODE_6_4 INODE_6_5 INODE_6_6\
                        INODE_6_7 INODE_6_8 INODE_6_9 INODE_6_10} \
                        {INODE_5_1 INODE_5_2 INODE_5_3 INODE_5_4 INODE_5_5 INODE_5_6\
                        INODE_5_7 INODE_5_8 INODE_5_9 INODE_5_10} \
                        {INODE_4_1 INODE_4_2 INODE_4_3 INODE_4_4 INODE_4_5 INODE_4_6\
                        INODE_4_7 INODE_4_8 INODE_4_9 INODE_4_10} \
                        {INODE_3_1 INODE_3_2 INODE_3_3 INODE_3_4 INODE_3_5 INODE_3_6\
                        INODE_3_7 INODE_3_8 INODE_3_9 INODE_3_10} \
                        {INODE_2_1 INODE_2_2 INODE_2_3 INODE_2_4 INODE_2_5 INODE_2_6\
                        INODE_2_7 INODE_2_8 INODE_2_9 INODE_2_10} \
                        {INODE_1_1 INODE_1_2 INODE_1_3 INODE_1_4 INODE_1_5 INODE_1_6\
                        INODE_1_7 INODE_1_8 INODE_1_9 INODE_1_10}}


set core_width_min [expr $NODE_width*10+$BIT_height+$BIT_core_gapX]
set core_height_min [expr $NODE_height*10+$BIT_height+$BIT_core_gapX+\
                    [max $BIT_height+$BIT_core_gapX $PFD_height]]

set core_width_pad_defined [expr $pad_width*$pad_numberX\
              -$delta_core_die_left-$delta_core_die_right+2*$pad_height]
set core_height_pad_defined [expr $pad_width*$pad_numberY\
              -$delta_core_die_top-$delta_core_die_bottom+2*$pad_height]

if {($core_width_pad_defined>$core_width_min) \
   && ($core_height_pad_defined>$core_height_min)} {
  puts "␣#␣Core␣dimensions␣are␣defindeb␣by␣pads"
  puts "␣#␣Core␣dimensions:␣$core_width_pad_defined␣$core_height_pad_defined"

floorPlan -s $core_width_pad_defined $core_height_pad_defined \
         [expr $delta_core_die_left-$pad_height] \
         [expr $delta_core_die_bottom-$pad_height] \
         [expr $delta_core_die_right-$pad_height] \
         [expr $delta_core_die_top-$pad_height]
set delta_coreX [expr ($core_width_pad_defined-$core_width_min)/2]
set delta_coreY [expr ($core_height_pad_defined-$core_height_min)/2]
```

```
puts "#  delta_coreX  =  $delta_coreX"
puts "#  delta_coreY  =  $delta_coreY"

} else {
puts "  #  Core  dimensions  are  defined  by  the  circuit  size"
puts "  #  Core  dimensions:  $core_width_min  $core_height_min"
set delta_coreX 0
set delta_coreY 0
floorPlan -s $core_width_min $core_height_min \
        [expr $delta_core_die_left-$pad_height] \
        [expr $delta_core_die_bottom-$pad_height] \
        [expr $delta_core_die_right-$pad_height] \
        [expr $delta_core_die_top-$pad_height]
}

specifyBlackBox -cell NODE_V2 -size $NODE_width $NODE_height
specifyBlackBox -cell NODE_V2_I -size $NODEI_width $NODEI_height
specifyBlackBox -cell PFD_V2 -size $PFD_width $PFD_height
specifyBlackBox -cell BIT_TOP -size $BIT_width $BIT_height

#placement of the NODEs
for {set in 0} {$in < 10} {incr in} {
    for {set jn 0} {$jn < 10} {incr jn} {
            set yc [expr $delta_core_die_bottom+$delta_coreY\
                    +$BIT_height+$BIT_core_gapY+$NODE_height*$in]
        set xc [expr $delta_core_die_left+$delta_coreX\
                +$BIT_height+$BIT_core_gapX+$NODE_width*$jn]
        set NODE_coordX($in,$jn) $xc
        set NODE_coordY($in,$jn) $yc
        set current_instance_name [lindex $NODE_instance_name $in $jn ]
        setObjFPlanBox Instance $current_instance_name $NODE_coordX($in,$jn) \
        $NODE_coordY($in,$jn) [expr $NODE_coordX($in,$jn)+$NODE_width ] \
        [expr $NODE_coordY($in,$jn)+$NODE_height ]
         }
}

#placement of the BIT_TOP: IBIT_1
#(between the first node (top left) and the reference)
set yc [expr $delta_core_die_bottom+$delta_coreY+$BIT_height+$BIT_core_gapY\
        +$NODE_height*10+$BIT_core_gapY]
set xc [expr $delta_core_die_left+$delta_coreX+$BIT_height+$BIT_core_gapX]
set BIT_TOP1_coordX $xc
set BIT_TOP1_coordY $yc
set current_instance_name IBIT_1
setObjFPlanBox Instance $current_instance_name $BIT_TOP1_coordX $BIT_TOP1_coordY \
          [expr $BIT_TOP1_coordX+$BIT_width] [expr $BIT_TOP1_coordY+$BIT_height]

#placement of the BIT_TOP: IBIT_2
#(between the reference and the node (5,5))
```

```
set yc [expr $delta_core_die_bottom+$delta_coreY+$BIT_height+$BIT_core_gapY\
      +$NODE_height*5-$BIT_width/2]
set xc [expr $delta_core_die_left+$delta_coreX]
set BIT_TOP2_coordX $xc
set BIT_TOP2_coordY $yc
set current_instance_name IBIT_2
setObjFPlanBox Instance $current_instance_name $BIT_TOP2_coordX $BIT_TOP2_coordY \
        [expr $BIT_TOP2_coordX+$BIT_width] [expr $BIT_TOP2_coordY+$BIT_height]
placeinstance $current_instance_name $BIT_TOP2_coordX $BIT_TOP2_coordY R90

#placement of the BIT_TOP: IBIT_3
#(between the nodes (10,5) and (10,6))
set yc [expr $delta_core_die_bottom+$delta_coreY]
set xc [expr $delta_core_die_left+$delta_coreX+$BIT_height+$BIT_core_gapX\
       +$NODE_width*5-$BIT_width/2]
set BIT_TOP3_coordX $xc
set BIT_TOP3_coordY $yc
set current_instance_name IBIT_3
setObjFPlanBox Instance $current_instance_name $BIT_TOP3_coordX $BIT_TOP3_coordY \
        [expr $BIT_TOP3_coordX+$BIT_width] [expr $BIT_TOP3_coordY+$BIT_height]
placeinstance $current_instance_name $BIT_TOP2_coordX $BIT_TOP2_coordY R180

#placement of the BIT_TOP: IBIT_4
#(between the reference and the node (10,10))
set yc [expr $delta_core_die_bottom+$delta_coreY+$BIT_height+$BIT_core_gapY\
      +$NODE_height*10+$BIT_core_gapY]
set xc [expr $delta_core_die_left+$delta_coreX+$BIT_height+$BIT_core_gapX+$NODE_width*9]
set BIT_TOP4_coordX $xc
set BIT_TOP4_coordY $yc
set current_instance_name IBIT_4
setObjFPlanBox Instance $current_instance_name $BIT_TOP4_coordX $BIT_TOP4_coordY \
        [expr $BIT_TOP4_coordX+$BIT_width] [expr $BIT_TOP4_coordY+$BIT_height]
```

List of Abbreviations and Symbols

Abbreviation	Description	Definition
SoC	System-on-Chip	page 1
SCA	Synchronous Clocking Area	page 3
GALS	Globally Asynchronous Locally Synchronous	page 3
GSLS	Globally Synchronous Locally Synchronous	page 3
PVT	Process, supply Voltage and Temperature	page 94
VCO	Voltage Controlled Oscillator	page 7
PLL	Phase-Locked Loop	page 5
ADPLL	All-digital phase-locked loop	page 7
DPC	Digital Phase Comparator	page 8
DLF	Digital Loop Filter	page 8
DAC	Digital-to-Analog Converter	page 8
DCO	Digitally-Controlled Oscillator	page 8
FO	Filter/oscillator	page 9
PI	Proportional-Integral filter	page 9
DFT	Design-For-Test	page 139
CTI	Coarse-Tuning Inverter	page 18
CTIA	Coarse-Tuning Inverter Additional	page 18
FTI	Fine-Tuning Inverter	page 19
DK	Design Kit	page 21
AMS	Analog Mixed Signal	page 22
LTI	Linear Time-Invariant	page 22
TSV	Through-Silicon Via	page 187
B2T	Binary-to-Thermometer	page 16
LUT	Look-Up Table	page 166
BB	Bang-Bang	page 11
PWM	Pulse Width Modulation	page 9
FCW	Frequency Control Word	page 70
VCD	Voltage-Controlled Delay	page 90
VCDE	Voltage-Controlled Delay Element	page 90
PDF	Probability Density Function	page 79
IP	Intellectual Property	page 130
TDC	Time-to-Digital Converter	page 11
PFD	Phase-Frequency Detector	page 9

Abbreviation	Description	Definition
CLA	Carry Look-Ahead adder	page 57
SPI	Serial Programming Interface	page 68
DDFS	Direct Digital Frequency Synthesis	page 110
ER	Error Rate	page 97
RFI	Reference Frequency Indicator	page 166

Symbol	Description	Definition
ϕ	Phase of the periodic signal	page 6
$\Delta\phi$,$e_{a,b}$	Phase error	page 6
F_0	Minimal oscillation frequency of DCO	page 63
Δf_{DCO}	Oscillator tuning step	page 39
f_s	frequency of digital filter	page 38
F_{ref}	Reference frequency	page 9
α	Proportional coefficient of PI filter	page 17
β	Integral coefficient of PI filter	page 17
K_p	Integer proportional parameter of PI filter	page 17
K_i	Integer integral parameter of PI filter	page 17
τ_{TDC}	TDC resolution	page 15
$e_{ri}[n]$	Quantified phase error	page 11
ΔF_{fine}	Fine frequency tuning step	page 63
ΔF_{coarse}	Coarse frequency tuning step	page 63
τ_{TDC}	TDC resolution	page 15
ΔT	Timing error	page 50
K_w	Filter input weighting coefficients	page 57
$numKw$	divisor programmed in the filter	page 58
$T_{DCO,fpga}$	DCO clock period in FPGA	page 111
$F_{n,vlsi}$	Nominal divided frequency of ASIC DCO	page 112
$F_{n,fpga}$	Nominal divided frequency of FPGA DCO	page 112
$\Delta F_{n,fpga}$	Frequency tuning step of FPGA DCO at nominal frequency	page 112
τ_{fpga}	TDC resolution in FPGA	page 114

Bibliography

[1] A. Abdelhadi, R. Ginosar, A. Kolodny, and E.G. Friedman. Timing-driven variation-aware nonuniform clock mesh synthesis. In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, pages 15–20, 2010. [cited at p. xxiv, 2]

[2] J.M. Akre, J. Juillard, D. Galayko, and E. Colinet. Synchronization Analysis of Networks of Self-Sampled All-Digital Phase-Locked Loops. 59(4):708–720, 2012. [cited at p. 8, 17, 23]

[3] J.M. Akre, J. Juillard, M. Javidan, E. Zianbetov, D. Galayko, A. Korniienko, and E. Colinet. A Design Approach for Networks of Self-Sampled All-Digital Phase-Locked Loops. In *Circuit Theory and Design (ECCTD), 2011 20th European Conference on*, pages 725–728, 2011. [cited at p. 17, 23]

[4] F. Anceau. Une technique de réduction de la puissance dissipée par l'horlogerie des circuits complexes rapides Zones isochrones. *Evolution*. [cited at p. xxiv, 3]

[5] C.J. Anderson, J.G. Petrovick, J.M. Keaty, J. Warnock, G. Nussbaum, J.M. Tendier, C. Carter, S. Chu, J. Clabes, and J. DiLullo. Physical design of a fourth-generation POWER GHz microprocessor. In *Solid-State Circuits Conference, 2001. Digest of Technical Papers. ISSCC. 2001 IEEE International*, volume 13, pages 232–233. IEEE, 2001. [cited at p. xxiv, 2]

[6] Venkatesh Arunachalam and Wayne Burleson. Low-power clock distribution in a multilayer core 3d microprocessor. In *Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pages 429–434. ACM, 2008. [cited at p. xlii, 187]

[7] Keith A Bowman, Carlos Tokunaga, Tanay Karnik, Vivek K De, and James W Tschanz. A 22 nm all-digital dynamically adaptive clock distribution for supply voltage droop tolerance. 2013. [cited at p. 2]

[8] M. Cabanas-Holmen, E. Cannon, A. Kleinosowski, J. Ballast, J. Killens, and J. Socha. Clock and Reset Transients in a 90 nm RHBD Single-Core Tilera Processor. *IEEE Transactions on Nuclear Science*, 56(6):3505–3510, December 2009. [cited at p. xxiv, 2]

[9] A. Chakraborty, K. Duraisami, A. Sathanur, P. Sithambaram, L. Benini, A. Macii, E. Macii, and M. Poncino. Dynamic Thermal Clock Skew Compensation Using Tunable Delay Buffers, 2008. [cited at p. 2]

[10] Antonio H. Chan and Gordon W. Roberts. A jitter characterization system using a component-invariant vernier delay line. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(1):79–95, 2004. [cited at p. 80]

[11] Piotr Dudek, Stanislaw Szczepanski, and John V Hatfield. A high-resolution cmos time-to-digital converter utilizing a vernier delay line. *Solid-State Circuits, IEEE Journal of*, 35(2):240–247, 2000. [cited at p. 51, 80]

[12] S.E. Esmaeili, A.J. Al-Khalili, and G.E.R. Cowan. A novel approach for skew compensation in energy recovery clock distribution networks, 2008. [cited at p. 2]

[13] R Franch, P Restle, N James, W Huott, J Friedrich, R Dixon, S Weitzel, K Van Goor, and G Salem. On-chip timing uncertainty measurements on ibm microprocessors. In *Test Conference, 2007. ITC 2007. IEEE International*, pages 1–7. IEEE, 2007. [cited at p. xvii, xxvi, xxviii, 28, 80, 81]

[14] Floyd M Gardner. *Phaselock techniques*. John Wiley & Sons, 2005. [cited at p. 9, 41]

[15] V. Gutnik and A.P. Chandrakasan. Active GHz clock network using distributed PLLs. *IEEE Journal of Solid-State Circuit*, 35:1553–1560, 2000. [cited at p. xxvii, 7, 108]

[16] Vadim Gutnik and Anantha Chandrakasan. On-chip picosecond time measurement. In *VLSI Circuits, 2000. Digest of Technical Papers. 2000 Symposium on*, pages 52–53. IEEE, 2000. [cited at p. 80]

[17] A. Hajimiri, S. Limotyrakis, and T.H. Lee. Jitter and phase noise in ring oscillators. *IEEE Journal of Solid-State Circuits*, 34:790–804, 1999. [cited at p. 18]

[18] Jens U Horstmann, Hans W Eichel, and Robert L Coates. Metastability behavior of cmos asic flip-flops in theory and test. *Solid-State Circuits, IEEE Journal of*, 24(1):146–157, 1989. [cited at p. xxxvii]

[19] J.U. Horstmann, H.W. Eichel, and R.L. Coates. Metastability behavior of CMOS ASIC flip-flops in theory and test. *IEEE Journal of Solid State Circuits*, 24(1):146–157, 1989. [cited at p. 85]

[20] H.Y. Hsieh, W. Liu, M. Clements, and P. Franzon. Self-calibrating clock distribution with scheduled skews, 1998. [cited at p. 2]

[21] M. Javidan, E. Zianbetov, F. Anceau, D. Galayko, A. Korniienko, E. Colinet, G. Scorletti, J.M. Akre, and J. Juillard. All-digital PLL array provides reliable distributed clock for SOCs. In *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pages 2589–2592. IEEE, 2011. [cited at p. xxiv, 3, 158]

[22] Mohammad Javidan, E Zianbetov, F Anceau, Dimitri Galayko, Anton Korniienko, Eric Colinet, Gérard Scorletti, Jean-Michel Akre, and Jérome Juillard. All-digital pll array provides reliable distributed clock for socs. In *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pages 2589–2592. IEEE, 2011. [cited at p. 174, 175]

[23] B. Jeon, Y. Moon, and T. Ahn. A study on 11 MHz $\sim$ 1537 MHz DCO using tri-state inverter for DAB application. *TENCON 2009 - 2009 IEEE Region 10 Conference*, pages 1–5, November 2009. [cited at p. 18]

[24] Deog-Kyoon Jeong, Gaetano Borriello, David A Hodges, and Randy H Katz. Design of pll-based clock generation circuits. *Solid-State Circuits, IEEE Journal of*, 22(2):255–261, 1987. [cited at p. xvii, 91]

[25] R. Ji, L. Chen, G. Luo, X.X Zeng, J. Zhang, and Y. Feng. A Novel Low-Power Clock Skew Compensation Circuit, 2008. [cited at p. 2]

[26] C. Johnson, D.H. Allen, J. Brown, S. Vanderwiel, R. Hoover, H. Achilles, C.Y. Cher, G.A. May, H. Franke, and J. Xenedis. A wire-speed powerTM processor: 2.3 GHz 45nm SOI with 16 cores and 64 threads. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, volume 44, pages 104–105. IEEE, 2010. [cited at p. xxiv, 3]

[27] A. Korniienko, E. Colinet, G. Scorletti, and E. Blanco. HâĹđ loop shaping control for distributed PLL network. In *2009 PhD Research in Microelectronics and Electronics*, pages 336–339, 2009. [cited at p. 17, 23]

[28] A. Korniienko, E. Colinet, G. Scorletti, E. Blanco, D. Galayko, and J. Juillard. A clock network of distributed ADPLLs using an asymmetric comparison strategy, 2010. [cited at p. 17, 23]

[29] A. Korniienko, G. Scorletti, E. Colinet, E. Blanco, J. Juillard, and D. Galayko. Control law synthesis for distributed multi-agent systems: Application to active clock distribution networks. In *Proceedings of the 2011 American Control Conference*, pages 4691–4696. Laboratoire Ampere, UMR CNRS 5005, Ecole Centrale de Lyon, 36 Av. Guy de Collongue, 69134 Ecully cedex, France, IEEE, 2011. [cited at p. 23, 57]

[30] Volodymyr Kratyuk, Pavan Kumar Hanumolu, Kartikeya Mayaram, and Un-Ku Moon. A 0.6 ghz to 2ghz digital pll with wide tracking range. In *Custom Integrated Circuits Conference, 2007. CICC'07. IEEE*, pages 305–308. IEEE, 2007. [cited at p. xix, 163, 164]

[31] Manoj Kumar, Sandeep K Arya, and Sujata Pandey. Low power digitally controlled oscillator designs with a novel 3-transistor xnor gate. *Journal of Semiconductors*, 33(3):035001, 2012. [cited at p. 17]

[32] Parag K Lala. *Digital circuit testing and testability*. Academic press, 1997. [cited at p. 139]

[33] P.M. Levine and G.W. Roberts. A high-resolution flash time-to-digital converter and calibration scheme. In *Test Conference, 2004. Proceedings. ITC 2004. International*, pages 1148–1157, 2004. [cited at p. 15]

[34] Jacob Minz, Xin Zhao, and Sung Kyu Lim. Buffered clock tree synthesis for 3d ics under thermal variations. In *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, pages 504–509. IEEE Computer Society Press, 2008. [cited at p. xlii, 187]

[35] Mosin Mondal, Andrew J Ricketts, Sami Kirolos, Tamer Ragheb, Greg Link, Narayanan Vijaykrishnan, and Yehia Massoud. Thermally robust clocking schemes for 3d integrated circuits. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07*, pages 1–6. IEEE, 2007. [cited at p. xlii, 187]

[36] T. Olsson and P. Nilsson. A digitally controlled PLL for SoC applications. *IEEE Journal of Solid-State Circuits*, 39:751–760, 2004. [cited at p. 18]

[37] I Miro Panades, Alain Greiner, and Abbas Sheibanyrad. A low cost network-on-chip with guaranteed service well suited to the gals approach. *Proc. NANONET*, 2006. [cited at p. 3]

[38] Vasilis F Pavlidis, Ioannis Savidis, and Eby G Friedman. Clock distribution networks for 3-d ictegrated circuits. In *Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE*, pages 651–654. IEEE, 2008. [cited at p. xix, xlii, xliii, 187, 188]

[39] G.A. Pratt and J. Nguyen. Distributed synchronous clocking. *IEEE Transactions on Parallel and Distributed Systems*, 6:314–328, 1995. [cited at p. xxvi, 5, 6, 24, 26, 36]

[40] Greg Rose. A stream cipher based on linear feedback over gf (2 8). In *Information Security and Privacy*, pages 135–146. Springer, 1998. [cited at p. 168]

[41] S. Rusu, S. Tam, H. Muljono, D. Ayers, J. Chang, B. Cherkauer, J. Stinson, J. Benoit, R. Varada, and J. Leung. A 65-nm dual-core multithreaded XeonÂő processor with 16-MB L3 cache. *Solid-State Circuits, IEEE Journal of*, 42(1):17–25, 2007. [cited at p. xxiv, 3]

[42] S. Rusu, S. Tam, H. Muljono, J. Stinson, D. Ayers, J. Chang, R. Varada, M. Ratta, S. Kottapalli, and S. Vora. A 45 nm 8-core enterprise Xeon processor. *Solid-State Circuits, IEEE Journal of*, 45(1):7–14, 2010. [cited at p. xxiv, 3]

[43] Stefan Rusu, Harry Muljono, David Ayers, Simon Tam, Wei Chen, Aaron Martin, Shenggao Li, Sujal Vora, Raj Varada, and Eddie Wang. 5.4 ivytown: A 22nm 15-core enterprise xeon® processor family. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 102–103. IEEE, 2014. [cited at p. 7]

[44] M. Saint-Laurent and M. Swaminathan. A multi-PLL clock distribution architecture for gigascale integration. *Proceedings IEEE Computer Society Workshop on VLSI 2001. Emerging Technologies for VLSI Systems*, pages 30–35, 2001. [cited at p. 26]

[45] M. Saint-Laurent, P. Zarkesh-Ha, M. Swaminathan, and J.D. Meindl. Optimal clock distribution with an array of phase-locked loops for multiprocessor chips. *Proceedings of the 44th IEEE 2001 Midwest Symposium on Circuits and Systems. MWSCAS 2001 (Cat. No.01CH37257)*, 1:454–457, 2001. [cited at p. 26]

[46] J.P. Schoellkopf. Circuit indicating the phase relation between several signals having the same frequency, 1996. [cited at p. 2]

[47] G. Scorletti and G. Duc. An LMI approach to dencentralized HâĹđ control. *International Journal of Control*, 74(3):211–224, 2001. [cited at p. xv, 24]

[48] R. Senthinathan, S. Fischer, H. Rangchi, and H. Yazdanmehr. A 600 MHz IA-32 microprocessor with enhanced data streaming for graphics and video, 1999. [cited at p. 2]

[49] C. Shan, E. Zianbetov, M. Javidan, F. Anceau, M. Terosiet, S. Feruglio, D. Galayko, O. Romain, E. Colinet, and J. Juillard. FPGA implementation of reconfigurable ADPLL network for distributed clock generation. In *2011 International Conference on Field-Programmable Technology*, pages 1–4. Ieee, December 2011. [cited at p. 128]

[50] Chuan Shan, Francois Anceau, Dimitri Galayko, and Eldar Zianbetov. âĂIJswimming poolâĂİ-like distributed architecture for clock generation in large many-core soc. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pages 2768–2771. IEEE, 2014. [cited at p. 3]

[51] Chuan Shan, Dimitri Galayko, and François Anceau. Design and modeling of adpll with sliding-window for wide range frequency tracking. In *New Circuits and Systems Conference (NEWCAS), 2012 IEEE 10th International*, pages 269–272. IEEE, 2012. [cited at p. 182]

[52] Chuan Shan, E Zianbetov, Mohammad Javidan, F Anceau, M Terosiet, S Feruglio, Dimitri Galayko, Olivier Romain, Eric Colinet, and Jérôme Juillard. Fpga implementation of reconfigurable adpll network for distributed clock generation. In *Field-Programmable Technology (FPT), 2011 International Conference on*, pages 1–4. IEEE, 2011. [cited at p. 77, 108]

[53] Chuan Shan, Eldar Zianbetov, Weiqiang Yu, Francois Anceau, Olivier Billoint, and Dimitri Galayko. Fpga prototyping of large reconfigurable adpll network for distributed clock generation. In *Reconfigurable Computing and FPGAs (ReConFig), 2013 International Conference on*, pages 1–6. IEEE, 2013. [cited at p. 128]

[54] Youngmin Shin, Ken Shin, Prashant Kenkare, Rajesh Kashyap, Hoi-Jin Lee, Dongjoo Seo, Brian Millar, Yohan Kwon, Ravi Iyengar, Min-Su Kim, et al. 28nm high-metal-gate heterogeneous quad-core cpus for high-performance and energy-efficient mobile application processor. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, pages 154–155. IEEE, 2013. [cited at p. 2]

[55] R.B. Staszewski, J.L. Wallberg, S. Rezeq, C.M. Hung, O. Eliezer, S. Vemulapalli, C. Fernando, K. Maggio, R. Staszewski, N. Barton, M.C. Lee, P. Cruise, M. Entezari, K. Muhammad, and D. Leipold. All-Digital PLL and Transmitter for Mobile Phones. *Technology*, 40(12):2469–2482, 2005. [cited at p. 7]

[56] Robert Bogdan Staszewski, Dirk Leipold, Khurram Muhammad, and Poras T Balsara. Digitally controlled oscillator (dco)-based architecture for rf frequency synthesis in a deep-submicrometer cmos process. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 50(11):815–828, 2003. [cited at p. 17]

[57] Simon Tam, Justin Leung, Rahul Limaye, Sam Choy, Sujal Vora, and Mitsuhiro Adachi. Clock generation and distribution of a dual-core xeon processor with 16mb l3 cache. In *Solid-State Circuits Conference, 2006. ISSCC 2006. Digest of Technical Papers. IEEE International*, pages 1512–1521. IEEE, 2006. [cited at p. 77]

[58] Paul Teehan, Mark Greenstreet, and Guy Lemieux. A survey and taxonomy of gals design styles. *Design & Test of Computers, IEEE*, 24(5):418–428, 2007. [cited at p. 3]

[59] J.A. Tierno, A.V. Rylyakov, and D.J. Friedman. A wide power supply range, wide tuning range, all static CMOS all digital PLL in 65 nm SOI. *IEEE Journal of Solid-State Circuits*, 43:42–51, 2008. [cited at p. xv, 12, 13, 18]

[60] José A Tierno, Alexander V Rylyakov, and Daniel J Friedman. A wide power supply range, wide tuning range, all static cmos all digital pll in 65 nm soi. *Solid-State Circuits, IEEE Journal of*, 43(1):42–51, 2008. [cited at p. xix, 164, 165]

[61] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing power in high-performance microprocessors. In *Proceedings of the 35th annual Design Automation Conference*, pages 732–737. ACM, 1998. [cited at p. 2]

[62] Y. Wang, J. Yu, Y. Surya, and C. Huang. A compact delay-recycled clock skew-compensation and/or duty-cycle-correction circuit, 2011. [cited at p. 2]

[63] R.B. Watson and R.B. Iknaian. Clock-buffer-chip with multiple-target automatic skew compensation, 1995. [cited at p. 2]

[64] John V Wehausen and Edmund V Laitone. *Surface waves*. Springer, 1960. [cited at p. 153]

[65] Jan Wilstrup. A method of serial data jitter analysis using one-shot time interval measurements. In *Test Conference, 1998. Proceedings., International*, pages 819–823. IEEE, 1998. [cited at p. xxvi, 80]

[66] L. Xiu. *VLSI circuit design methodology demystified: a conceptual taxonomy*. Wiley-IEEE Press, 2008. [cited at p. 139]

[67] Liming Xiu, Wen Li, Jason Meiners, and Rajitha Padakanti. A novel all-digital pll with software adaptive filter. *Solid-State Circuits, IEEE Journal of*, 39(3):476–483, 2004. [cited at p. 173]

[68] Takahiro J Yamaguchi, Mani Soma, Masahiro Ishida, Toshifumi Watanabe, and Tadahiro Ohmi. Extraction of peak-to-peak and rms sinusoidal jitter using an analytic signal method. In *VLSI Test Symposium, 2000. Proceedings. 18th IEEE*, pages 395–402. IEEE, 2000. [cited at p. xxvi, 80]

[69] T. Yamashita, T. Fujimoto, and K. Ishibashi. A dynamic clock skew compensation circuit technique for low power clock distribution, 2005. [cited at p. 2]

[70] Alfred Yeung, Hamid Partovi, Qawi Harvard, Luca Ravezzi, John Ngai, Russ Homer, Matthew Ashcraft, and Greg Favor. 5.8 a 3ghz 64b arm v8 processor in 40nm bulk cmos technology. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 110–111. IEEE, 2014. [cited at p. 2]

[71] M. Yuffe, M. Mehalel, E. Knoll, J. Shor, T. Kurts, E. Altshuler, E. Fayneh, K. Luria, and M. Zelikson. A Fully Integrated Multi-CPU, Processor Graphics, and Memory Controller 32-nm Processor. *Solid-State Circuits, IEEE Journal of*, (99):1–1, 2011. [cited at p. xxiv, 3]

[72] Jun Zhao and Yong-Bin Kim. A 12-bit digitally controlled oscillator with low power consumption. In *Circuits and Systems, 2008. MWSCAS 2008. 51st Midwest Symposium on*, pages 370–373. IEEE, 2008. [cited at p. 17]

[73] E Zianbetov, F Anceau, Mohammad Javidan, Dimitri Galayko, Eric Colinet, Jérôme Juillard, et al. Design and vhdl modeling of all-digital plls. In *Proceedings of the 8th IEEE International NEWCAS Conference (NEWCAS'10)*, pages 293–296, 2010. [cited at p. 163, 174]

[74] Eldar Zianbetov. *Distributed clocking for synchronous SoC*. PhD thesis, UPMC, 2013. [cited at p. 53, 71]