MSc THESIS

Adaptive Clock Scheduling for Pipelined Structures

Ben Kuiper

Abstract

CE-MS-2009-24

With the advance of fabrication technology into the deep sub-micron era process parameter variations, temperature, and voltage fluctuations start to induce large variations into the delay of integrated circuits. The smaller the feature size the larger the variations become, and thus also their influence on the critical path delay of a logic stage in a pipelined structure. Traditionally, the critical path delay variations are dealt with by augmenting the critical path delay with a safety margin (over design) that is large enough to cover the worst case scenario. Due to the increasing variability the performance gap between this worst-case clocking method and the potential performance is increasing resulting into a massive underutilization of the technology. This thesis introduces a design for variability for pipelined structures called Adaptive Inverter Chain Based Pipeline (AICBP). It can observe and compensate for delay variations in pipelined structures. The main idea is to expose the data and the clock to the same variations such that the clocking of the registers is adapted to the delay of the logic. This results in a substantial reduction of the over design thus on a better utilization of the technology potential performance. Besides the hardware which can compensate for delay variations, the AICBP design also includes a number of smart mechanisms which can resolve timing errors and improve performance. The area overhead of the AICBP design is relatively small in

comparison to other state-of-the-art proposals, since it does not require the augmentation of circuitry at the flip-flop level as is the case with some designs for variability. Some high level simulations have been done and these indicate an up to 46% performance improvement. However, this performance improvement depends heavily on the specific delay variations and the properties of the combinatorial logic. Two extensions to the basic AICBP design are presented in this thesis as well. One of these includes an advanced recovery mechanism and the other focuses on data dependent delay compensation techniques. These extensions allow the performance to be improved, but the cost in terms of area and also power is quite high.

Adaptive Clock Scheduling for Pipelined Structures

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Ben Kuiper
born in Leiderdorp, The Netherlands

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Adaptive Clock Scheduling for Pipelined Structures

by Ben Kuiper

Abstract

With the advance of fabrication technology into the deep sub-micron era process parameter variations, temperature, and voltage fluctuations start to induce large variations into the delay of integrated circuits. The smaller the feature size the larger the variations become, and thus also their influence on the critical path delay of a logic stage in a pipelined structure. Traditionally, the critical path delay variations are dealt with by augmenting the critical path delay with a safety margin (over design) that is large enough to cover the worst case scenario. Due to the increasing variability the performance gap between this worst-case clocking method and the potential performance is increasing resulting into a massive underutilization of the technology. This thesis introduces a design for variability for pipelined structures called Adaptive Inverter Chain Based Pipeline (AICBP). It can observe and compensate for delay variations in pipelined structures. The main idea is to expose the data and the clock to the same variations such that the clocking of the registers is adapted to the delay of the logic. This results in a substantial reduction of the over design thus on a better utilization of the technology potential performance. Besides the hardware which can compensate for delay variations, the AICBP design also includes a number of smart mechanisms which can resolve timing errors and improve performance. The area overhead of the AICBP design is relatively small in comparison to other state-of-the-art proposals, since it does not require the augmentation of circuitry at the flip-flop level as is the case with some designs for variability. Some high level simulations have been done and these indicate an up to 46% performance improvement. However, this performance improvement depends heavily on the specific delay variations and the properties of the combinatorial logic. Two extensions to the basic AICBP design are presented in this thesis as well. One of these includes an advanced recovery mechanism and the other focuses on data dependent delay compensation techniques. These extensions allow the performance to be improved, but the cost in terms of area and also power is quite high.

Laboratory : Computer Engineering
Codenummer : CE-MS-2009-24

Committee Members :

Advisor: Sorin Cotofana, CE, TU Delft

Chairperson: Koen Bertels, CE, TU Delft

Member: Stephan Wong, CE, TU Delft

Member: René van Leuken, CAS, TU Delft

Contents

List of Figures	xi
List of Tables	xiii
Acknowledgements	xv
1 Introduction	1
1.1 Problem sketch	1
1.2 Proposed systematic approaches	1
1.3 Casta DIVA	2
1.4 Adaptive Inverter Chain Based Pipeline	2
1.5 Overview of the thesis	3
2 Logic delay characteristics and basics of the AICBP design	5
2.1 Logic delay characteristics	5
2.1.1 Delay types	5
2.1.2 Time domain of delays	6
2.1.3 Variance of the logic delays	7
2.1.4 Spatial correlation of delays	7
2.2 General idea of AICBP	7
2.3 Inverter chain details	9
3 AICBP components	11
3.1 Importance of contamination delay	11
3.2 Contamination detection system	13
3.2.1 Ways of dealing with contaminations	13
3.2.2 Principles of the contamination detection system	15
3.2.3 Implementation of the contamination detection system	15
3.2.4 Meta-stability detection	19
3.2.5 Inverter chain for the contamination detection system	20
3.3 Performance improvement with a hold system	21
3.4 Phase system	24
3.4.1 Phase properties	24
3.4.2 Keeping the high phase equal	25
3.4.3 High phase system	26
3.4.4 Specific implementation of the high phase system	29
3.4.5 Constraints of the high phase system	31
3.5 Cost and overhead of the added systems	32

4	Fault handling	35
4.1	Actions to be taken when a fault occurs	35
4.2	Flushing the pipeline	36
4.3	Details on flushing	37
4.3.1	Bookkeeping to identify sets of data	37
4.3.2	Resetting in order to kill all high phases	37
4.3.3	Specific usage of the FIFO	38
4.4	RTL design of fault treatment logic	38
4.4.1	FIFO	39
4.4.2	Faults and reinsertion of data	40
4.4.3	Constraint	41
4.4.4	Glitch removal circuitry	43
4.5	Resets given to all the stages	44
4.6	Control logic	46
4.7	Conclusions regarding the fault handling	48
5	Interconnection and clock frequency	49
5.1	Interconnection with surrounding logic	49
5.2	The determination of the clock frequency	50
6	Simulation and performance results	53
6.1	Pipelined array multiplier	53
6.2	VHDL implementation of the pipelined array multiplier	55
6.2.1	Properties of the gate and inverter chain delays	56
6.2.2	Delay mechanism in VHDL simulation	57
6.2.3	Specific delay values	58
6.3	Simulation results	59
6.3.1	Simulation with only dynamic delays	59
6.3.2	Simulation with static and dynamic delays	61
6.3.3	Simulation with reduced contamination delay	63
6.4	Simulation evaluation	63
6.4.1	General remarks	63
6.4.2	Relation between faults and performance	64
6.4.3	Reducing delay variance	65
6.4.4	static delay variance	65
6.4.5	Reducing contamination delay	66
6.4.6	Relation between delay variance, contamination delay, hold system position, and average output period	66
6.5	Concluding remarks regarding the simulation	67
7	Out of order execution	69
7.1	Out of order execution	69
7.2	Out of order execution for AICBP	70
7.3	Out of order contamination detection system	71
7.3.1	General working	71

7.3.2	Constraints	72
7.4	Fault handling	75
7.4.1	Basics of the fault treatment logic	75
7.4.2	Constraints of the fault treatment logic	76
7.5	Simulation results	77
7.6	Conclusions regarding out-of-order execution	77
8	Data dependent delay compensation	83
8.1	Basic idea	83
8.2	Basic components	83
8.2.1	Time giving and time taking inverter chain	84
8.2.2	Input transition detection flip-flop	85
8.3	Implementation of data dependent delay compensation	86
8.4	Constraints for the data dependent delay compensation system	90
8.5	Simulation	92
8.5.1	Input data	92
8.5.2	Transparency window length	93
8.5.3	Contamination detection system	93
8.5.4	Simulation setup	93
8.6	Evaluation of the simulation results	94
8.7	Conclusions	94
9	Design steps	97
10	Recommendations and conclusions	99
10.1	Recommendations for future work	99
10.2	Conclusions	100
	Bibliography	104
A	Simulation results	105
A.1	simulation with only dynamic delay variation	105
A.2	Simulation with static and dynamic delays	107
A.2.1	No static delay variations	107
A.2.2	Increasing static delays	108
A.2.3	Decreasing static delays	109
A.2.4	One stage with a long static delay	110
A.3	Simulation with reduced contamination delay	111
A.4	Out of order execution	112
A.5	Data dependent delay compensation	113
A.5.1	Transparency period of 0%	113
A.5.2	Transparency period of 10% and uncorrelated input data	114
A.5.3	Transparency period of 10% and correlated input data	115
A.5.4	Transparency period of 20% and uncorrelated input data	116
A.5.5	Transparency period of 20% and correlated input data	117

List of Figures

2.1	Clock signal distribution in the proposed pipeline structure	8
2.2	Clock signal distribution in the proposed pipeline structure	8
3.1	Snapshot of the values in all the inverter chains.	12
3.2	Clarification of the contamination and propagation delay definition. . . .	12
3.3	Example of contamination and propagation delay.	14
3.4	Example of contamination and propagation delay.	14
3.5	Principles of a contamination detection system.	16
3.6	Design of the contamination detection system.	16
3.7	NAND-based SR-latch.	17
3.8	Waveform of a situation with no contamination fault.	18
3.9	A meta-stability detection circuit.	19
3.10	Example of how to use two separate contamination detection systems. . .	21
3.11	Circuitry for holding a rising edge.	22
3.12	The hold system.	22
3.13	Holding of the rising edge reduces the high phase.	25
3.14	A simple layout of the phase system.	26
3.15	Snapshot of a situation with smaller high phases.	29
3.16	The design of the high phase system.	30
4.1	Overview of the circuitry needed for fault treatment.	36
4.2	The RTL design of the fault treatment logic.	39
4.3	Timeline with all the events during a flush.	41
4.4	Clarification of the added latency constraint.	42
4.5	Low pass RC filter.	44
4.6	Glitch remover circuit from [2].	44
4.7	Reset circuitry.	44
4.8	Advanced reset circuitry.	45
4.9	Finite state machine of the control logic.	47
6.1	Example of an array multiplier from [18].	54
6.2	A stage of the pipelined array multiplier.	54
6.3	Implementation of the multiply-add block.	55
6.4	Performance of a 4 stage AICBP (100,000 clock cycles)	61
6.5	Performance of an 8 stage AICBP (100,000 clock cycles)	61
6.6	Performance of a 16 stage AICBP (100,000 clock cycles)	62
7.1	Simple overview of the logic needed for out-of-order re-execution.	70
7.2	The contamination detection system for out-of-order re-execution. . . .	71
7.3	The RTL design of the out-of-order fault treatment logic.	79
7.4	<i>Pointer difference</i> unit from the RTL design in Figure 7.3.	80
7.5	The finite state machine of the RTL design in Figure 7.3	81
7.6	Performance of a 4 stage AICBP with out-of-order re-execution	82

7.7	Performance of an 8 stage AICBP with out-of-order re-execution	82
8.1	Time giving chain.	84
8.2	Time taking chain.	84
8.3	An ep-SFF from [20] with circuitry wich signals input changes.	86
8.4	Circuitry which can generate a pulse on an input change.	86
8.5	Combining the <u>Late Arrival</u> pulse of every flip-flip into one signal.	87
8.6	Implementation for data dependent delay compensation.	88
8.7	The toggle system.	89
A.1	Performance of a 4 stage AICBP with on the left side the average output period and on the right side the number of faults.	105
A.2	Performance of a 8 stage AICBP with on the left side the average output period and on the right side the number of faults.	106
A.3	Performance of a 16 stage AICBP with on the left side the average output period and on the right side the number of faults.	106
A.4	Performance of a 4 stage AICBP. Left: average output. Right: number of faults.	107
A.5	Performance of an 8 stage AICBP. Left: average output. Right: number of faults.	107
A.6	Performance of a 4 stage AICBP. Left: average output. Right: number of faults.	108
A.7	Performance of an 8 stage AICBP. Left: average output. Right: number of faults.	108
A.8	Performance of a 4 stage AICBP. Left: average output. Right: number of faults.	109
A.9	Performance of an 8 stage AICBP. Left: average output. Right: number of faults.	109
A.10	Performance of a 4 stage AICBP. Left: average output. Right: number of faults.	110
A.11	Performance of an 8 stage AICBP. Left: average output. Right: number of faults.	110
A.12	Performance of a 4 stage AICBP. Left: average output. Right: number of faults.	111
A.13	Performance of an 8 stage AICBP. Left: average output. Right: number of faults. The performance results for spacing=0.7 are not shown, since they do not improve the performance.	111
A.14	Performance of a 4 stage AICBP with out of order re-execution. Left: average output. Right: number of faults.	112
A.15	Performance of an 8 stage AICBP with out of order re-execution. Left: average output. Right: number of faults.	112
A.16	Performance of a 4 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.	113
A.17	Performance of an 8 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.	113

- A.18 Performance of a 4 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.114
- A.19 Performance of an 8 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.114
- A.20 Performance of a 4 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.115
- A.21 Performance of an 8 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.115
- A.22 Performance of a 4 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.116
- A.23 Performance of an 8 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.116
- A.24 Performance of a 4 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.117
- A.25 Performance of an 8 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.117

List of Tables

2.1	Order-of-magnitude variability time scale and estimated delay impact. (from [14])	6
3.1	Hardware cost of the systems added to the inverter chain.	33
4.1	Hardware cost of the fault treatment system surrounding the AICBP. . .	48
6.1	Relation between delay variance, contamination delay, hold system position, and average output period	67

Acknowledgements

At the end it all comes down to doing some hard work if you want to achieve something. Especially when things don't seem to work the way you want, the only option is just to keep trying and to work a bit harder. This certainly applies as well for this thesis. Nevertheless, doing some hard work must be made possible by the people you are surrounded with, and in view of that I want to thank my parents. I want to thank them not only for their financial support, but also for their moral support. Thanks to them I have been able to fully concentrate on my study and also on my thesis.

Last but not least I would like to thank Sorin Cotofana for his guidance. The discussions we had kept me thinking and they have ultimately guided me towards a final and good solution. Most of all I would like to thank him for the work and advice on the paper.

Ben Kuiper
Delft, The Netherlands
September 29, 2009

Introduction

1.1 Problem sketch

Due to the ongoing downscaling of CMOS technology the delay variation of CMOS circuitry grows. The main cause is an increasing parameter variation determined by manufacturing imperfections [15]. Together with other sources of variance, such as temperature and voltage variations, this parameter variation causes the variance of logic delay in CMOS to become bigger as CMOS technology feature size becomes smaller.

The clock frequency of synchronized circuitry is determined by the worst-case delay of the critical path in the combinational logic plus augmented with the register overhead and a certain safety margin. Due to the larger delay variations, the gap between the critical path worst-case delay and the average delay is becoming bigger. This implies that in practical designs the clock frequency is not scaling with the mean critical path delay made possible by the advances in technology. As suggested in [5] this gap is increasing, and consequently a generation of performance improvement can be lost only due to process parameter variations.

Attempts to reduce the variations at the technology level may be very expensive or they may be eventually not possible any longer. It is thus of crucial importance to deal with these variations not only at the physical design level, but also at the circuit and system levels. Up to date the designers were able to apply some ad hoc solutions on these levels to account for the variability and to improve performance. However, structural and systematic techniques are of interest as they can be applied in various situations without redesigning the circuit. This allows the designers to fulfill the short time to market constrain imposed by the current market conditions. Currently some systematic approaches have been proposed which can make the clock period less than the worst-case critical path and these are explained in the upcoming section.

1.2 Proposed systematic approaches

Some approaches have been proposed until now and among these are Razor [10] and the techniques proposed by Bowman et al. [6]. Both require additional circuitry at the flip-flop level which can detect timing errors. As a consequence the clock period or supply voltage can be dynamically controlled in order to face delay variations. The main disadvantage of such an approach is the substantial circuitry overhead present in each and every flip-flop in the pipeline.

Another approach is the one of TEAtime [21]. This approach does not have a circuit overhead per flip-flop, but some logic is added to the whole stage instead. Logic is added which resembles the critical path of the stage, and with this logic it is dynamically determined whether the clock period is larger than the critical path delay. With this

information the clock frequency can be dynamically controlled. Although this is most likely a cheaper solution in terms of area than Razor, it still has some disadvantages. The critical path delay in one stage determines the clock period although other stages may have a smaller critical path delay. Thus there is no compensation possible if one stage has a very long critical path delay while the others do not have a long critical path delay. Also when a long critical path delay occurs only once it has a negative impact on the clock frequency. If a critical path delay is only once almost longer than the clock period then the clock frequency is scaled down. After that it may take some time before the clock frequency is scaled up to its original frequency. The key thing is that a one time event reduces the frequency whereas this is not really needed.

Another design for variability is the one from Andrade et al. [3]. The idea here is to add a chain with inverters next to the combinational logic in each stage. All the inverter chains are connected to each other and the first inverter chain is fed with the clock signal. The output of each inverter chain in each stage is used for clocking the stage register. The overhead of this scheme is relatively small, but the disadvantage is that this scheme only focuses on latency. As will become clear later on, this scheme has some similarities with the techniques proposed in this thesis. The techniques in this thesis are focused on improving the throughput though instead of the latency.

1.3 Casta DIVA

Another systematic proposal for design for variability is Casta DIVA [9], which proposes an architectural template for a System On Chip (SOC) that allows circuit and system designers to seamlessly incorporate design for variability in their designs. The proposed architecture is organized into three layers and the lowest layer (local agents) deals with variations in combinational logic. However, while the behavior of this layer is clearly specified, no specific implementation for the lowest layer is proposed in [9]. The design for variability presented in this thesis can be used as a local agent implementation. The specific interaction of the design in this thesis with the Casta DIVA platform is not specified yet, but interaction with the Casta DIVA platform is certainly possible. By using the design presented in this thesis in the Casta DIVA platform, a complete design for variability platform is retrieved.

It must be noted that the design presented in this thesis is more general and can be used in conjunction with any pipelined structure. However, if integrated in the Casta DIVA platform, the design in this thesis is augmented with other higher level services, which makes it more effective in compensating delay variations. The special interconnections between the IP blocks are an example of such a service.

1.4 Adaptive Inverter Chain Based Pipeline

This thesis introduces a design for variability called Adaptive Inverter Chain Based Pipeline (AICBP), which can observe and compensate delay variations in pipelined structures. The main idea behind the design is to mold into the clock propagation the data propagation profile such that data and clock are exposed to the same variations. As a

result the register data sampling process is not disturbed by variations. This can be partially achieved by the utilization of an inverter chain like in [3], which lets the clock signal propagate along with the data. This results in a substantial reduction of the over design, and the consequence is a better utilization of the technology potential performance. Besides the inverter chain some other smart mechanisms need to be embedded as well. As is explained later on, the synchronization between different sets of data can not always be maintained due to different logic delays. Due to that some smart mechanisms are embedded next to the inverter chain in an AICBP which can also detect timing errors. As explained later in the thesis, when increasing the clock frequency certain timing errors can occur and these need to be detected. Once detected, a recovery procedure needs to be executed in order to let the pipeline function in a correct way.

1.5 Overview of the thesis

In the next chapter (Chapter 2) the logic delay characteristics are described, and also the principles of the AICBP design are made clear. In the following chapters 3 and 4 all the mechanisms which are there to prevent and handle the timing errors are explained. Following that the interconnection of the AICBP with the surrounding logic is discussed in Chapter 5.

To get an impression of the performance improvements possible, some simulations have been done as well and the results are there in Chapter 6.

The AICBP design in this thesis also has some extensions. One of these extensions includes a faster fault handling by allowing out of order execution. The details are discussed in Chapter 7 where the resulting performance improvements are shown as well.

It must be said that the AICBP design discussed thus far is actually a design which compensates for critical path variations and not for data dependent delay variations. An extension is shown in Chapter 8 which makes use of data dependent delay variations and can therefore improve performance.

Before making some final remarks and conclusions, a short design methodology is explained in Chapter 9 which designers can follow in order to use AICBP. At the end of the thesis in Chapter 10 some final remarks are made and conclusions are drawn. Some recommendations for future work are made here as well.

Logic delay characteristics and basics of the AICBP design

2

Before discussing the details on AICBP, more is said about the characteristics of the logic delays. These are very important for the design of an AICBP and therefore discussed first. After that the principles of an AICBP are discussed.

2.1 Logic delay characteristics

Since AICBP is a design for variability, it is important to know what the variability characteristics are of logic delays. Delays for logic gates in the Deep Sub-Micron (DSM) regime have a statistical nature due to factors like process parameter variations. The parameter variations are caused by manufacturing imperfections and are one of the problems inherent to very small (nano scale) devices. Not only process parameter variations cause the delay of logic gates to vary, but also voltage and temperature fluctuations cause the delay to vary. This section makes clear the impact these factors have on the delays and also in what kind of timescale the effect is.

2.1.1 Delay types

To put things into perspective, there are different kinds of delay variations and those are the following:

- W2W (wafer-to-wafer): The variation of circuit delays between different wafers.
- D2D (die-to-die): The variation of circuit delays between different dies from one wafer.
- WID (within-die): The variation of circuit delays between different circuits on the same die.

WID variations make up the biggest part of the total variation. In modern integrated circuits the correlation of the power supply fluctuations is low [14], meaning that different regions on the die can get a different voltage and also the variation of the voltage in each region is different. Also the temperature within the die can differ quite a lot due to hot spots [4], and this also contributes to the delay variation within the die. According to [3] for medium and large size circuits around 60% of the environmental factors which influence delay are made up by independent nature fluctuations. This means that around 60% of the total variations are made up by WID variations.

The influence of W2W and D2D variations is still significant though and it has a negative effect on the clock frequency for traditional pipelined logic. This clock frequency is not only determined by the slowest critical path delay in the slowest stage on the die,

but also by the slowest die on the slowest wafer [5]. Thus the speed of the faster dies is limited by the delay of a logic stage in the slowest die. This reduces the potential performance of a lot of dies especially when the difference between an average die and worst case die becomes larger.

2.1.2 Time domain of delays

There are a lot of mechanisms involved which determine the delay variability. The timescale of the variations caused by all these mechanisms are very different. For example, the voltage can fluctuate within a time domain of one clock cycle, but the temperature can certainly not do so. The temperature can change quite slowly and has its effect on the delay, but delays determined by for example the transistor widths can never change, because these are determined at the fabrication.

Table 2.1 is given to give an impression of the different time scales for the mechanisms which determine the delay variance. It is a table from [14] and it illustrates some of the mechanisms involved. As can be seen, 32% of the total delay variance is caused by mechanisms which influence delay in the time domain of a clock period. It can be seen as well that 30% of the influence on the delay variance is not going to change ever. There is thus actually a static and dynamic part of the delay variation. Especially for short computations this is the case, because variations in the range of $1e-04$ till $1e-01$ are fixed during that computation as well, and one can say that there is truly a static and dynamic part. These characteristics are quite important for the simulation of the AICBP design and have a major impact on performance (Chapter 6).

Time Domain (Secs)	Mechanism	Delay Impact (3 sigma)
1e12	Lithography Node	20%
1e09	Electromigration	5%
1e08	Hot Electron Effect	5%
1e06	Neg Bias Temperature Instability	15%
1e04	Chip Electrical Mean Variation	15%
1e-01	Across Chip L_{poly} Variation	15%
1e-04	Self Heating / Temperature	12%
1e-08	SOI History Effect	10%
1e-10	Supply Voltage	17%
1e-10	Line-to-Line Coupling	10%
1e-11	Residual Source/Drain Charge	5%

Table 2.1: Order-of-magnitude variability time scale and estimated delay impact. (from [14])

2.1.3 Variance of the logic delays

For design and simulation purposes it is very important to know how much the variance of a logic stage is. This is rather hard to tell though, because it depends on multiple factors like technology and the specific design and layout of the circuit. Even if all these factors are known, the specific delay variance and also the specific distribution of the delays are still difficult to tell, because these depend on a lot of factors and are therefore very complex.

Nevertheless, an assumption needs to be made regarding the variance, because this needs to be known for simulation. In [14] the delay variance is given for different kind of adder circuits and this is below $\frac{\sigma}{\mu} = 0.1$ (σ = variance and μ = mean delay). The delays follow a normal/Gaussian distribution and the delay variance is the total delay variance, so WID, D2D, and W2W variations are included. This number does certainly not apply to all other circuits and technologies, but it just serves here to get an idea of the logic delay variance. The delay variances of circuits in [17] are also below $\frac{\sigma}{\mu} = 0.1$, and therefore the $\frac{\sigma}{\mu} = 0.1$ is assumed as a valid delay variance. This assumption may be in a lot of cases too severe, but it accounts then for bigger variances for future technologies.

2.1.4 Spatial correlation of delays

The delays of gates which are located close to each other are correlated. One reason for this is that often these gates use the same power supply, and are thus subjected to the same power supply variations. As a consequence, the delays of the gates located close to each other are all influenced in the same way by the power supply fluctuation. The temperature, which also has its influence on the delay, is the same for all the gates which are located close to each other, and therefore these gates are all influenced in the same way as well. The process parameter variability also has a degree of spatial correlation as is stated in [16]. It means that gates located next to each other suffer into a certain extent from the same process parameter variations.

All these factors cause the delay variability to have a high spatial correlation as is shown in [14] and [16]. In other words, the delays of the gates which are located next to each other are very much alike, and if one of the delays changes with a certain amount, the rest of the delays are likely going to be changed in the same way. This spatial correlation of the delays is actually the basis of the AICBP as will turn out in the next section.

2.2 General idea of AICBP

In a normal pipeline registers are clocked by a clock signal. This clock signal is distributed to all the registers and also to all the flip-flops within the registers. The clock signal in AICBP is not connected to all the registers, but an inverter chain is used to distribute the clock signal as is displayed in figure 2.1. The purpose of the inverter chain is to have a delay that is just as long as the delay of the combinational logic plus the setup time of the flip-flops. This could be achieved by choosing the right even number of inverters and to correctly size them. By making the inverter chain delay just as long as the logic delay,

it is made sure that the rising clock edge (assuming that this is the triggering edge) will arrive at every register when the data are ready to be sampled as well.

Figure 2.1: Clock signal distribution in the proposed pipeline structure

Of course the inverter chain will burn some extra power and use some extra area, but such an inverter chain has got a property that makes it worth using. If the inverter chain is routed through the combinational logic (shown in figure 2.2), it has a strong correlation with both the dynamic and static delay variations of the logic. If the inverter chain is laid out next to the critical path, the correlation with the critical path delay is very high because of the spatial delay correlation explained in Section 2.1.4. This is exactly what is wanted, because the clock delay is actually made equal with the delay of the critical path regardless the circumstances. Such an approach guarantees that the register data sampling process is adapted to the delay of the logic. For example, if the delay of the logic takes longer due to a factor like for example temperature, the inverter chain is delayed as well. Consequently the saving action of the registers happens later, because the rising edge arrives later.

Figure 2.2: Clock signal distribution in the proposed pipeline structure

Due to the use of the concatenated inverter chains, AICBP is able to let the different stages compensate for each others delays. This is one of the reasons why the slowest stage will not determine the clock frequency, and this is the big difference with the proposed techniques of [10], [6], and [21] explained in section 1.2. These techniques exploit the W2W and D2D variations, but, unlike AICBP, they do not exploit the WID variations.

AICBP is operating with a certain clock frequency and at every rising clock edge (assuming that this is the saving edge) a new set of input data is sampled by the first register in the pipeline. The results corresponding to these input data become available at the AICBP output after the latency of all the chains added together. The clock period of an AICBP can be reduced because the mean delay of the chain is less than the clock period determined by the worst-case delay. Moreover a clock period which is equal to the mean delay can be used, because then all the registers sample the data at more or less the same time moment. At least, that is the case when all the stages in a specific

pipeline have a delay which equals the mean delay. Of course this is certainly not always the case, and therefore an AICBP has to embed, apart of the inverter chains, specific circuitry to prevent, detect, and resolve time faults that may occur due to the utilization of a clock frequency higher than the one determined by the worst case delay. These subsystems, presented in chapter 3, are the main difference between AICBP and the proposed technique in [3] and enable the AICBP performance improvement.

2.3 Inverter chain details

Before discussing the circuits or systems added to the inverter chains, something more can be said about the inverter chains.

The length of the inverter chain can be expressed as follows:

$$\text{Length inverter chain} = \text{critical path length} + \text{safety margin} + \text{setup delay flip-flop} \quad (2.1)$$

The *safety margin* is there to compensate if there is not a 100% correlation between the critical path and the inverter chain. For example, if the delay of the critical path under some circumstances is slightly longer than the delay of the chain, the clocking of the registers may occur too early. A safety margin is added to prevent such a situation. In practice there will never be a 100% correlation, so some safety margin always needs to be added. How much safety margin is needed depends on the technology, layout, and routing, so nothing concrete can be said here.

A choice can be made with the placing and routing to place the inverter chain very close to the critical path or to place it further away. When placing it further away, there is less delay correlation with the critical path, and consequently the safety margin needs to be larger. A larger safety margin will reduce performance, thus placing the inverter chain next to the critical path is preferred. However, it could be the case that the logic in one stage contains multiple critical paths. In such a case the inverter chain cannot be laid out close to all critical paths, so therefore the safety margin needs to be larger. The enlargement is needed to account for the lesser correlation of some of the critical paths with the delay of the inverter chain.

An advantage of using an inverter chain, which has not been mentioned thus far, is that no clock network is needed. The clocking of the registers is done by the outcome of the inverter chain, so designers do not have to design a clock network and worry about clock skew. On the other hand this means that all the registers will not clock the data at the same time due to delay differences between the inverter chains. This phenomenon may be hazardous as will be explained in the next chapter.

AICBP components

In the last section it is noted that each register at the end of a pipeline stage will clock data at a different time. In other words, the clocking of the registers is asynchronous. In this chapter it is explained why this is problematic, and how this problem can be solved. Also other components are described which are there to improve performance. Actually the following components, or systems, are described in this chapter:

- Contamination detection system
- Hold system
- High phase system

All these systems are applied to each inverter chain in each stage. For example, if there is a four stage pipeline, four contamination detection systems are needed. What these systems do and how they work is explained in the next sections. First of all, the contamination detection system is discussed, since it relates to the asynchronous register clocking.

3.1 Importance of contamination delay

When using an AICBP, the data and the rising edge propagate through the pipeline with a speed that is determined by the logic it goes through. The delay of the logic is most likely different for each stage, and also it varies in time. Consequence of this variation in time is that the delay of the same inverter chain and logic may be larger or smaller for one data set than for another. The result of all these delay variations is that the arrival of the rising edges, and thus also data arrival, will not occur at the same time at all the registers.

Figure 3.1 tries to sketch a situation where the rising edge will not arrive at the same time at all the registers. The rectangular boxes above the logic are the inverter chains, and the other boxes between the logic are of course the registers. The waveform above tries to make clear the value of the outputs of all the even inverters. It actually describes the state of the inverter chain at a certain position in the chain. The figure is a snapshot at a certain time instance, so the horizontal axis of the waveform defines the position in the chain.

As can be seen in the figure, the rising edges will not arrive at the same time due to delay variations explained earlier in 2.1. The second rising edge from the left is travelling faster than the previous rising edge, which is the edge in front or the third rising edge from the left. Because of the faster travelling there are in this situation two rising edges in one stage and that may be problematic. Two rising edges in one stage also means

Figure 3.1: Snapshot of the values in all the inverter chains.

there are two sets of data travelling in one stage. It may happen that the last inserted data may influence the data that is in front which belongs to the rising edge in front. Before the in front data is clocked, it may already be influenced by the upcoming data, and that causes incorrect functioning of the pipeline. To understand this phenomenon the following definitions are introduced:

- *Contamination delay* - The time it takes for a change at the input of a combinational circuit to affect the output.
- *Propagation delay* - The time it takes for a change at the input of a combinational circuit to change the output and keep it stable. When an input change occurs, it takes some time before the output changes and is finally stabilized. The time that this takes is the propagation delay.

Figure 3.2 makes clear the two definitions. A waveform for a bit signal at both the input and output of some combinational logic is shown.

Figure 3.2: Clarification of the contamination and propagation delay definition.

Noteworthy is that both the contamination and propagation delay are data dependent. For example, if in Figure 3.2 the input signal does not change its value, the output signal might not change its value either depending on other inputs. If the output does not change its value, the propagation delay for the output is zero and the contamination delay is infinitely long.

When returning back to the situation in Figure 3.1, one can see that especially the contamination delay is quite important. If in the second stage both sets of data must not interfere, then the contamination delay of the logic must be large enough. If it would be too short, then the upcoming data, which belongs to the edge on the left of the second stage, would influence the data that is almost at the register. That is because with a small contamination delay the effects of a different set of data are noticed quite fast at the input of the stage register. If the previous data is not sampled yet, then it is affected or contaminated.

Throughout the rest of the thesis the term contamination will be used if a set of data is affected by upcoming data. So when data is contaminated, it means that it is affected by another set of data that is following in the pipeline. Consequently the data are invalid.

3.2 Contamination detection system

3.2.1 Ways of dealing with contaminations

Contaminations are a serious threat, because they can let the pipeline malfunction. In this paragraph a way for preventing contaminations is explained, and also a technique is proposed for AICBP which can deal with contaminations.

To prevent a contamination from happening one could choose to make the clock frequency low enough. When the clock frequency is low, the sets of data in the pipeline are situated far apart from each other and will not have the chance to contaminate each other. As is explained in Section 2.1 the variance of the logic delay is quite big. That also applies for the contamination delay. If one has to account for the whole range of contamination delays, the spacing between two sets of data must be quite big. Contamination delays may be in rare cases very small, and that means spacing must be enough at all times.

Another problem is that one set of data can propagate faster through the pipeline than other sets of data. So the spacing between the sets of data at the beginning of the pipeline may be different from what it will be for the same sets at the end of the pipeline. To have enough spacing at the end of the pipeline the spacing at the beginning must be larger.

To assure a correct working of the pipeline at all times, the spacing must be large enough to account for the shortest contamination delays. Also spacing at the beginning must be larger in order to have enough spacing at the end of the pipeline. These two constraints can be met by making the clock period low enough. However, this is not a desirable thing to do, because throughput will not be that good with a lower clock period.

Instead of avoiding contamination, one can also allow contaminations to happen. Contaminations need to be detected however then, because if a contamination occurs, the appropriate measures can be taken in order to always have a correct result. In such a way the clock frequency will not have to be decreased as much as with the avoiding of contaminations. The drawback is that fixing the contamination faults will also cost time, but altogether the throughput will still be faster than with the avoiding

of contaminations. To maximize throughput, it has been chosen to allow contaminations for AICBP. In the next paragraph it is explained how the contamination detection can be done.

To better understand the relation between contamination, propagation delay, and critical path delay, the figures 3.3 and 3.4 are given as an example. They both show some combinational logic which has some inputs and outputs. In figure 3.3 it can be seen that the delay of the AND-gate determines the smallest contamination delay possible, because its output transition will be seen first. In figure 3.4 the smallest contamination delay possible is determined by the AND- and NOR-gate, because they form the shortest path to one of the outputs. The delay of the gates that will determine the smallest contamination delay possible is heavily related to the delay of the other gates. This is due to the spatial correlation of the delay of logic gates (Section 2.1.4).

Figure 3.3: Example of contamination and propagation delay.

Figure 3.4: Example of contamination and propagation delay.

The longest propagation delay is in both figures three gate delays long, and since these three gates are the longest path they make up the critical path. Because of the heavy correlation between all the gates, one can say that the smallest contamination delay possible is a fraction of the critical path. The critical path delay in figures 3.3 and 3.4 is both three gate delays, and the smallest contamination delay possible is for 3.3 one gate delay and for 3.4 two gate delays. Consequently the smallest contamination delay possible is respectively $\frac{1}{3}$ and $\frac{2}{3}$ of the critical path delay.

The property that the smallest contamination delay possible is a fraction of the critical path delay is used in the system to detect contaminations. Just as with a chain, which one can use to measure the delay of the critical path, one could use a chain to detect contamination delays as well. The chain just needs to be a fraction of the main chain that signals the critical path delay. The fraction is determined by the logic like is just explained. This chain for contamination detection does not have to be a separate chain, because the main inverter chain can be used. However it may be the case that a separate chain needs to be used though, because the correlation with the logic is not good enough. More on this separate chain is explained later in section 3.2.5.

It must be noted that by using the just explained fraction in the chain for contamination detection untrue contaminations may be detected, because contamination delays may be larger than the smallest contamination delay possible. For example, the contamination delay may be formed by two gates instead of one in Figure 3.3 depending on the input data pattern. The contamination detection system assumes the smallest contamination delay possible, because it does not check the data which are in the logic stage. Therefore the contamination detection system may signal a contamination while there may be no contamination at all.

3.2.2 Principles of the contamination detection system

A general circuit implementation for the contamination detection is shown in figure 3.5. The two AND-gates both give a short pulse when a rising edge comes along. The first AND-gate lets the SR-latch get a high value when the rising edge, and thus also the data, travels by. If the same rising edge gets to the end of the stage, the other AND-gate will let the SR-latch reset to a zero value. This is the normal operation when no contaminations occur. When a possible contamination does occur however, the SR-latch will not have the chance to reset to a zero value, because the next rising edge will arrive at the flip-flop clocking point before the current rising edge arrives at the last AND-gate. Consequently the contamination fault signal is set, and the arrival of the rising edge at the last AND-gate is not going to change that.

It is of utmost importance to place the flip-flop clocking point at a fraction of the chain length that equals the fraction of the earlier explained contamination and critical path delay. If for example the smallest contamination delay is a third of the critical path, the clocking point must be at a third of the total chain length as well. If in this example the rising edge is at a third of the chain length, it means that signals may already have arrived at the input of the register. When the previous data is not clocked yet, a contamination may have occurred and the SR-latch, which signals a contamination fault, is set. When the contamination fault signal is high, the control logic will notice this, and consequently a reset will be given. The exact details on this are explained later in Chapter 4, but first there are some details and constraints to be explained regarding this contamination detection system. This is done in the next paragraph.

3.2.3 Implementation of the contamination detection system

Figure 3.6 shows a design with slightly more details. The logic gates are replaced with gates that are cheaper in complementary CMOS logic, and the SR-latch is replaced with

Figure 3.5: Principles of a contamination detection system.

a NAND-based SR-latch. Such a NAND-based SR-latch is shown in figure 3.7 and is most of the time a preferred latch in complementary CMOS logic. The most striking thing in 3.6 however is the meta-stability detector. The purpose and use of this will be explained in the next paragraph, because first some important constraints will be enlightened.

Figure 3.6: Design of the contamination detection system.

The first important thing to note is that the pulse generated by the NAND gates must be longer than the hold time of the NAND-based SR-latch. The duration of the pulse is determined by the number of inverters between the two connection points. As can be seen, three inverters are used and that might not be enough. If it is not enough, five inverters, or any other odd number, could be used instead.

The next constraint is there to assure the hold time of the flip-flop is not violated.

Figure 3.7: NAND-based SR-latch.

The flip-flop must have sampled its input well before the (NAND-based) SR-latch gets the chance to change its output. The constraint can be expressed as follows:

$$\text{hold time ff} < \text{distance clk-NAND} + \text{delay NAND} + \text{delay SR-latch} \quad (3.1)$$

distance clk-NAND is the time it takes for a rising edge to travel from the clock input point of the flip-flop to the NAND connection point. It is marked as well in figure 3.6. In practice, two inverters will be more than enough for this, because the hold time will not most likely be larger than two inverter delays and two NAND-gate delays. Another important thing to note regarding this constraint is that there is no accounting for the wire delays in this constraint, because that depends very much on the routing.

One may wonder if it is possible to make both the set and reset of the SR-latch low, because this combination is generally considered a forbidden combination. That is indeed possible, but it will not lead to faulty behaviour of the contamination detection system. When both the set and reset are low, the flip-flop has already registered a contamination fault, so it will not matter if the SR-latch will malfunction because of this. The control logic will notice the contamination fault and the next step of the control logic is to give a reset. If the reset is long enough, the SR-latch will function as it did before. There is more to be said about the length of the reset, but this is explained more in detail in Section 4.5.

To make sure the set and reset signals are not both low when there is no contamination fault, the following constraint must be applied:

$$\text{duration NAND-pulse} - \text{delay SR-latch} + \text{hold time ff} < \text{distance clk-NAND} + \text{delay NAND} \quad (3.2)$$

This constraint is derived from the situation where no contamination occurs. In fact it is the situation at which the arrival of the rising edge at the end of the chain cannot occur later, because else a contamination would be detected. This situation is shown in waveform in figure 3.8 and is used to derive the constraint. The specific situation is a worst-case scenario, because the low reset cannot occur any later without a contamination fault occurring. Consequently the low reset can only occur earlier, which means the low set and low reset are more spaced apart from each other in time. In the situation of figure 3.8 the hold time of the flip-flop is violated. Likely this will cause a meta-stable output, which is detected by the meta-stability detector. Consequently a contamination fault will be detected. However, there is a chance that the flip-flop, depending on the

specific implementation, will give a zero output value and no contamination fault is detected. This is a worst-case scenario, so it is assumed for constraint 3.2.

Figure 3.8: Waveform of a situation with no contamination fault.

If constraint 3.2 is not completely satisfied it may not be a disaster after all, because the set and reset can be both low without malfunctioning ([12]). The problem occurs when both signals go to high again more or less at the same time and consequently the SR-latch will have an internal race condition. To be on the safe side though, it is chosen not to let the set and reset be low at the same time.

The following constraint defines where the clock input point of the flip-flop must be in the chain. More specific it defines after how many inverters it must be placed.

$$\begin{aligned}
 \text{Position clock point} = & \frac{\text{Smallest contamination delay}}{\text{Critical path delay}} \times \text{number of chain inverters} \\
 & + \frac{2 \times \text{delay NAND} + \text{delay wire} + \text{inverter delay} + \text{delay SR-latch} - \text{setup time ff}}{\text{inverter delay}} \\
 & - \frac{\text{hold time register}}{\text{inverter delay}} \quad (3.3)
 \end{aligned}$$

In this case the delay of the wire is not neglected, because the wiring to the SR-latch can be quite long. Two other important remarks must also be made regarding this constraint. The first is that the input point of the flip-flop must be behind an even number of inverters. If the constraint gives an odd number of inverters, the input point must be one inverter earlier in order to have an even number. The other remark concerns the delay of the gates in the constraint. In order to always have a correct working, one must assume the fastest delays possible for all the delays in the numerator. On the other hand the worst-case value for the delay of an inverter must be assumed.

The *hold time register* is subtracted, because during the hold time of the register the inputs of the register may not change. That means that signal changes caused by the smallest contamination delay are not allowed either during this period. By subtracting

hold time register it is guaranteed that if there are signal changes during the hold time, a contamination fault will be signaled.

The gate delays are also added to the constraint, because they take up a considerable amount of time, and if one would ignore them contamination faults would unnecessarily occur more often. Actually the flip-flop does a comparison between two points in the inverter chain. The first point is the arrival of the rising edge in the next stage and the other point is the point in the chain at which contamination is going to occur (*Position clock point*). The information of the next stage arrival is not available directly, because it takes some time to propagate to the flip-flop. Consequently the comparison must take place at a later time instance. To have the same comparison, *Position clock point* is delayed as well, and this can be done by simply placing it an even number of inverters later in the inverter chain. It is not problematic if this is not done, but it will decrease performance, since contamination faults are going to occur more often then.

One may wonder why a flip-flop is used for the contamination detection in figure 3.6 while a simple SR-latch could have been used as well. The reason is that with a flip-flop the violation of the hold-time will not lead to race conditions. This is very important, because the hold time will most likely be violated since the input of the flip-flop is asynchronous. However, another phenomena can occur when violating the setup or hold time, and that is a meta-stability state of the flip-flop. More on this is explained in the next paragraph.

3.2.4 Meta-stability detection

The input of the flip-flop in figure 3.6 is asynchronous which means that an input transition can occur during the setup- and hold-time of the flip-flop. Such a transition can cause the flip-flop to malfunction and not give a proper high or low output value. It means the output has a voltage which is in between the voltage of the high and low value. In such a situation it is said that the flip-flop is in a meta-stable state. Detecting such a meta-stable state is important, because the upcoming gates may interpret the in-between voltage level as a different logic state. The possibility also exists that upcoming gates also give an in-between voltage as a consequence.

Figure 3.9: A meta-stability detection circuit.

One possible circuit for detecting meta-stability is shown in figure 3.9. It is also used in [10] and is more or less the same as the patent of [11]. The trick in this circuit is to use different sizing for the transistors in the inverters connected to the input. Different sizing will lead to different resistance of the transistors, and as a consequence the inverters will switch at a different voltage level. When there is an in-between voltage level, the sizing can be made in such a way that the upper left inverter gives a low output value, and the lower inverter gives a high output value. As a consequence, the AND-gate will give a high output value, and the meta-stable state is detected.

There is however a very small chance that the output of the meta-stability detection circuit is meta-stable itself. This can occur when the output voltage of the flip-flop is at the border of the meta-stable voltage range. Consequently the detection circuit cannot give a definite high or low output value, and the output is meta-stable. Such an output is not problematic when the output of the flip-flop is high, because the NOR-gate will have a low output value regardless the voltage of the contamination detection circuit. If the output of the flip-flop is low however, there is a small chance that the output of the NOR-gate is meta-stable as well. It is questionable if the following SR-latch will react on this meta-stable voltage, and also how big the chance of this is.

The issue of meta-stability, and also reducing the chance of failure because of meta-stability, to an acceptable level is part of low-level circuit design. Actually this is a subject on its own, and much can be told about this, but it is out of the scope of this thesis.

3.2.5 Inverter chain for the contamination detection system

In section 3.2.1 it is noted that the chain which is connected to the contamination detection system may not have a good delay correlation with the gates that determine the smallest contamination delay. This is problematic, because if the delay of the gates which determine the contamination delay is smaller than the chain delay, contaminations may be undetected.

The correlation may not be that good due to two factors:

- The first part of the main inverter chain which is connected to the contamination detection system is not laid out on the die close to the gates which determine the smallest contamination delay.
- The number and type of gates of the path through to the logic which determines the smallest contamination delay may differ a lot from the inverter chain. For example, the contamination delay path may be laid out next to the inverter chain, but this path may contain a lot of wiring and also perhaps gates which are much larger than an inverter. Therefore the delay distribution of the chain may be different from the distribution of the path determining the contamination delay even though the mean delays of both paths may be the same.

To account for this bad correlation a safety margin can be applied. This comes down to making the fraction $\frac{\text{Smallest contamination delay}}{\text{Critical path delay}}$ in equation 3.3 smaller. Disadvantage is that the performance will be reduced, because contamination faults are signalled more often then. Another option, which will not degrade performance, is to use a separate inverter chain for the contamination detection system. This inverter chain can be laid out next to the path which determines the smallest contamination delay. Also the inverters and wiring in this chain can be shaped in a way that it resembles the logic which determines the smallest contamination delay. The obvious disadvantage of this extra chain is the extra power and area usage.

What also may occur is that multiple paths through the logic may determine the smallest contamination delay. The delay of these paths may be quite un-correlated, so using one chain for these might not be a good idea since the fraction $\frac{\text{Smallest contamination delay}}{\text{Critical path delay}}$

needs to be made smaller to account for the non-correlations. A solution for this could be to use multiple inverter chains. Each inverter chain is for each path which can determine the smallest contamination delay, and each chain has its own contamination detection system. Figure 3.10 gives an example of how two separate contamination detection systems can be used. Using separate chains and contamination detection systems is of course more expensive in terms of area and power.

Figure 3.10: Example of how to use two separate contamination detection systems.

3.3 Performance improvement with a hold system

Contamination faults are quite costly, because when such a fault occurs it needs to be fixed. Fixing means re-executing the same calculation, and possibly also re-executing the other data in the pipeline as well. How much the fixing costs depends on the re-execution strategy and the pipeline length, but more is told on this in Chapter 4. Regardless the re-execution strategy that is used, a contamination fault is still quite costly and will have an impact on performance in a negative way. It is therefore that next to contamination detection contamination prevention is very important.

Preventing contaminations can be done by making sure the rising edges are spaced apart enough from each other. Of course the distance between the rising edges can be made big enough such that contaminations are not going to occur anyway, but as told in 3.2.1 this will have a negative impact on throughput. Another option is to make sure the distance between the rising edges is big enough to make sure that the chance of a contamination is small. In such a way contamination faults will not occur that very often

and will not have a big impact on the throughput of the pipeline. On the other hand the rising edges can still be close enough to each other to still have a good throughput.

Making sure that the distance between the rising edges is enough in such a way that contamination faults are not likely can be done by delaying the rising edge. Delaying the rising edge in its turn can be done with circuitry shown in figure 3.11. If the $\overline{\text{Hold}}$ signal is high, the circuitry will act as a normal inverter. On the other hand if the $\overline{\text{Hold}}$ signal is low, the output is high regardless the input. One can use this property to delay a rising edge for a certain amount of time, because when the rising edge arrives at the input of the circuitry, it cannot continue to propagate. It can only continue propagating when the $\overline{\text{Hold}}$ signal is high again.

Figure 3.11: Circuitry for holding a rising edge.

It can be said that the circuitry of figure 3.11 can be used to hold the rising edge for a certain amount of time at its place. Holding the edge at its place for a while will make the space between two rising edges bigger, because the rising edge in front of the edge which is hold can still propagate. Holding a rising edge can be done with the hold system shown in figure 3.12, which uses the circuitry of figure 3.11 as a key element.

Figure 3.12: The hold system.

The two NAND-gates connected to the chain in figure 3.12 give a negative pulse when the rising edge comes by. The most left NAND-gate makes sure the SR-latch is

reset. The output of the latch will be in such a way then that the next rising edge will be held at its place. This next rising edge can only start propagating if the current rising edge is at the second to the chain connected NAND. Then the SR-latch is set and the hold circuitry of figure 3.11 is given a high value, so that the next rising edge can start propagating. The NAND-gate connected to the input and output of the hold circuitry of figure 3.11 is there to make sure that the high phase behind the rising edge is not killed. If it was not there, the input of the chain would be made low just after the rising edge started propagating. The result is a short high phase, which is not desirable as will be told in 3.4.1.

$$\begin{aligned} \text{Time between rising edges} > \text{delay inverter} + \text{delay NAND} + \text{delay SR-latch} + \\ \text{delay NAND} + \text{switching delay hold logic} \end{aligned} \quad (3.4)$$

The constraint makes sure that if a rising edge is allowed to start traveling, the logic has enough time to set the hold logic at a blocking state before the next rising edge arrives. It must be noted that the wire delays are important as well in this constraint, but they are not mentioned since they are unknown. To reduce the wire delays it might be a good idea to place the SR-latch close to the hold circuit.

The time between two rising edges can be guaranteed by the hold system of the previous stage. That system must make sure there is enough spacing such that the time between the rising edges at the beginning of the current stage is large enough to apply to constraint 3.4. Such a thing is not very difficult to guarantee, and it also strongly depends on the length of the high phase in the inverter chain. A more difficult thing to guarantee though is the following constraint:

$$\begin{aligned} \text{Time between falling edge and next rising edge} > \text{delay NAND} + \\ \text{switching delay hold logic} \end{aligned} \quad (3.5)$$

It is there to make sure that once the high phase is gone, the hold circuitry can be set to a blocking state in time before the next rising edge arrives. This cannot be guaranteed by this hold system, but the phase system in the next paragraph 3.4.3 can make sure constraint 3.5 always applies.

The question which remains is where the release point in figure 3.12 must be placed in the inverter chain. Of course that depends on the amount of spacing one wants between two rising edges. If the wanted spacing is expressed as a fraction of the inverter chain length, the position in the chain where to put the release point is expressed as follows:

$$\begin{aligned} \text{Release point} = \text{wanted spacing} \times \text{number of chain inverters} - \\ \frac{\text{delay NAND} + \text{delay AND} + \text{delay SR-latch} + \text{wire delay} + \text{delay NAND}}{\text{inverter delay}} \\ - \frac{\text{switching delay hold circuit}}{\text{inverter delay}} \end{aligned} \quad (3.6)$$

The *Release point* is expressed in an amount of inverters that needs to be placed in front of the release point. It must be placed after an even amount of inverters. In case of

an odd number it will not matter for a correct working if the release point is placed one inverter earlier or later. The release point is actually placed earlier in the chain, because it takes a while before the rising edge at the release point reaches the hold circuit. To have a good placing of the release point, the average delays must be taken for all the gates in the fraction. For a correct working it will not matter if one or more gates will deviate from this delay.

The most interesting question is still what the amount of spacing between two rising edges (*wanted spacing* in constraint 3.6) should be to have the best throughput possible. There is no simple answer for this question, since it depends heavily on the variation of the delays in the pipeline and the chosen input clock frequency. This will become clear in Chapter 6 where simulations are shown and also the effects of certain spacings caused by the hold system are shown.

3.4 Phase system

3.4.1 Phase properties

The clock signal which is inserted in the AICBP has a high phase and a low phase. These phases travel through the inverter chains just as a rising and a falling edge does. The high phase is located in the chain where the output of all the even inverters is high. The other way around goes for the low phase as well, because all the even inverters have a low output then. A snapshot of such a situation is shown in figure 3.1.

The length of the high and low phase can differ as is the case in figure 3.1. The cause is for example a different propagation speed for a rising and falling edge which belong to one high phase. As a result, the length of a high and low phase at the end of the pipeline can differ very much from the same phases a while ago at the beginning of the pipeline. The reason is that the propagation speed of the edges of the same phase differs continuously throughout their traveling through all the chains.

In practice it could be possible for a long pipeline that a phase is completely going to disappear. It may happen for example with a high phase when the belonging rising edge travels slower than the falling edge. If that happens in every stage, the high phase is going to disappear. However, it must be noted that when the falling edge gets close to the rising edge they will most likely propagate with the same speed, due to exposure of the same factors which influence delay. At least, that is the case with inverters which have a balanced delay. Still there is a danger of the disappearance of a phase, and appropriate measures need to be taken to prevent such a situation.

Disappearing of a high phase may also occur when a rising edge is hold. With holding it is meant the holding of the rising edge as described in section 3.3. When a rising edge is hold, the falling edge, which is behind somewhere in the previous chain, is still travelling. As a result if one holds the rising edge long enough, the falling edge reaches the hold circuit and the high phase is gone. Figure 3.13 shows two moments in time at which the rising edge is hold. One can see that the falling edge is still travelling and it shortens the high phase.

Besides the problem of the disappearance of a phase, a problem may be encountered with glitches in the inverter chain. When a falling edge is very close to a rising edge

Figure 3.13: Holding of the rising edge reduces the high phase.

in front, the former high phase is actually nothing more than a glitch. Such a glitch may cause the flip-flops in the registers not to function correct anymore. Not only the flip-flops will not function anymore, but also the contamination fault detection circuitry of figure 3.6 will not function correct anymore, because the set and reset pulses to the latches may be too short.

Clearly there is some circuitry needed to prevent the just described situations from occurring. The next paragraph describes a system, which can prevent these situations.

3.4.2 Keeping the high phase equal

As stated in the previous paragraph, a system is needed which guarantees that no low or high phase is going to disappear. This system must also prevent the occurrence of glitches. To achieve these goals, a system can be made which keeps the length of the high phase as long as a portion of the chain length in one stage. If for example the length of the high phase should be the half of the chain length, circuitry can be made which makes the high phase take up half of the chain. This circuitry must be placed in every pipeline stage, because if a phase, which is made to take up half of the chain, travels to a next stage its length may change. This changed length will need to be corrected again if one wants to keep the phase at a fixed portion.

When keeping the high phase at a fixed portion, the length of the low phase is going to vary. This must be the case, since the distance between two rising edges can vary. Consequently the length of the low phase needs to vary if the length of the high phase is fixed. Letting the low phase vary still has the potential risk of a disappearance of the low phase. Instead of trying to prevent such a situation from occurring, it is chosen to allow such a situation, because it is quite rare in most pipeline setups. This rare disappearance must be detected though, because it will cause the whole pipeline to malfunction.

3.4.3 High phase system

A system which keeps the length of the high phase equal in every stage and detects the disappearance of a low phase is actually described in the last paragraph. The basic layout of such a system is shown in figure 3.14. One can see a piece of logic in figure 3.14 that can insert a zero or a one into the chain. Circuitry which can do this was already shown in figure 3.11. The logic can also signal a *collision fault* which means a low phase is gone. In case of such a *collision fault* the control logic of the whole pipeline will take the appropriate actions.

Figure 3.14: A simple layout of the phase system.

The placement of the insertion logic, which is the logic that inserts a '0' or a '1', determines the length of the high phase that is made. If it is placed for example at the half of the chain length, then the resulting high phase will also have the length of half of the chain.

The working of the high phase system is as follows in chronological order:

- When a rising edge arrives, this will be detected by the logic, because the AND-gate in front of the insertion logic will give a pulse then. The next action of the logic will be to insert a '1' into the chain. This is actually equal to the propagation of the rising edge, because that is actually also an insertion of a '1'. If this is equal, one could wonder what the purpose is of this, but the purpose is to lengthen high phases that are too short. After an incoming rising edge the logic will keep inserting a '1', also if the falling edge arrives shortly thereafter. Consequently the high phase will be lengthened then, because a portion of the original low phase is made high now.
- When the rising edge is at the next stage and is also allowed to continue travelling, the logic receives a pulse from the most right AND-gate. As a consequence the logic will stop inserting a '1', and a '0' is inserted instead. This action will make a high phase shorter if it is too long, because a portion of the high phase is made low then. It is important to note that the AND-gate will only give a pulse once the rising edge is allowed to travel further in the next stage. Situations like in

figure 3.13 are not known to occur then, because the high phase is kept at the same length during the holding period.

- When the next rising edge comes in, the logic will stop inserting a '0' and starts inserting a '1' again. In this way the rising edge is allowed to travel further. Actually the last two steps repeat themselves again and summarize the working of the high phase system when there is no disappearance of the low phase.

If there is a disappearance of a low phase, the logic cannot detect this disappearance, but it can predict however that it is going to happen. This can be done by detecting if the rising edge is very close to the falling edge in front. Since the position of the falling edge is actually defined by the high phase system, the high phase system can detect as well if the next rising edge is nearby. It can be done by simply checking if the insertion is still '1' when the next rising edge comes along. If the insertion is '1', it means that the falling edge is not out of the high phase system yet.

A falling edge must travel faster than a rising edge in order to have a disappearance of a low phase. To prevent such a situation, it must be made sure that those two edges are separated far enough from each other. That is the reason why the most left AND-gate is used as an input for the logic. If this AND-gate gives a pulse and the insertion is still '1', then the two edges are too close to each other. In addition, the *collision fault* signal is used to signal the control logic of the pipeline that a low phase might disappear in the future. The control logic will take the appropriate actions as is told in Chapter 4.

A collision fault is thus an existence of the possibility of two edges colliding with each other, and consequently a disappearance of a low phase. Noteworthy is that with a collision fault there might not be a disappearance of a low phase after all, because the rising edge may travel faster than the falling edge, or perhaps with equal speed. Thus a number of collision faults may be false alerts, because the low phase may not disappear after all. One might wonder if these false alerts are not going to influence the performance in a negative way. The trick is however to make sure that these collision faults are not going to occur that often anyway. In that way false alerts will not have a big impact on performance.

Making sure collision faults are not going to occur that often can be done in two ways: Reducing the distance between the left AND-gate and insertion logic, and reducing the length of the high phase. Both techniques are discussed in the next two sub-paragraphs.

3.4.3.1 Reducing distance between AND-gate and insertion logic

Making sure the inputs of the left AND-gate in figure 3.14 are not too far away from the insertion logic will reduce the number of collision faults. If the inputs are close to the insertion logic, only rising edges that are very close to the falling edge will be marked as a collision fault. However, the inputs may not be too close, because else a possible disappearance of the low phase is not detected. The distance must thus be big enough to detect a possible disappearance and small enough to prevent a lot of unnecessary collision faults.

It must be noted that not only a possible disappearance of a low phase must be detected, but also a possible too small low phase must be detected. If a low phase is too

small, it can cause the pulses generated by for example the NAND-gates in figures 3.6 and 3.12 to be too small. Thus not only the disappearance of the low phase must be prevented, but also the low phase must be prevented from being too small.

One might say that the distance needed between the AND-gate and insertion logic should be very big in order to detect all possible disappearances and too small low phases. This will not have to be the case, because when a falling edge is close to a rising edge, both edges will be exposed to the same factors which influence delay. Consequently the two edges will have the same speed and will not collide with each other. That is at least the case when all the inverters have an equal rise and fall time. Of course this is never precisely the case, so that is why there needs to be some reasonable separation between the two edges. It is thus of utmost importance to have an equal rise and fall time for all the inverters in the chain, so that a minimal distance between the inputs of the AND-gate and the insertion logic can be achieved.

Another reason why the distance between the AND-gate and insertion logic will not have to be that big, is that the guarantee must only be given for one stage. In other words, it must only be guaranteed that the edges will not be too close to each other for one stage length, because in the next stage there is the same piece of logic which will detect again if a collision fault must be signaled.

An interesting option to reduce the distance as well is to place multiple detection circuits in the chain which can detect a possible collision fault. The guarantee of the low phase not being too small must now be given over a smaller length, because both edges will reach the next detection circuit sooner. As a consequence, the edges are allowed to be closer to each other, because the distance the falling edge has to get closer to the rising edge is smaller now. If the edges are allowed to be closer to each other, it means that the distance between the inputs of the AND-gate and the insertion logic can be less. Consequently less contamination faults will be detected, and performance will be improved.

It is actually up to the designer to decide what to do, because the needed distance between the AND-gate and insertion logic, or the use of extra detection circuits, strongly depends on technology and the specific situation.

3.4.3.2 Reducing length of the high phase

Instead of making the high phase just as long as the low phase, the high phase can be made shorter. This can be done by simply placing the high phase system more towards the end of the inverter chain of the stage. The high phase, which is made by the high phase system, is as long as the distance from the system to the end of the inverter chain. Reducing this distance, by placing it more towards the end, will make the high phase smaller.

A situation with smaller high phases is shown in figure 3.15. It can be easily seen that the chance of a rising edge colliding with a falling edge is smaller now. The low phase is made bigger, and that is why it is more unlikely to disappear. Consequently an edge collision is not going to occur that soon. The freedom of where the rising edges are is also bigger now, because they can get closer to each other without a disappearance of a low phase.

Figure 3.15: Snapshot of a situation with smaller high phases.

Thus it is preferred to have a small high phase, but it cannot be too small on the other hand. If it would be too small, there could be a chance that the high phase is going to disappear. The high phase should thus be big enough to never disappear, because a disappearance would let the pipeline malfunction.

The high phase must not only be big enough to prevent a disappearance, but it must also be big enough to let certain gates generate pulses that are long enough. An example of such gates are the two NAND-gates connected to the chain in figure 3.6, and also the two NAND-gates connected to the chain in figure 3.12. If pulses are not long enough, constraints 3.2 and 3.9 will for example also not apply. Thus the high phase system must make high phases that are long enough under all circumstances.

3.4.4 Specific implementation of the high phase system

In figure 3.16 a specific implementation for the high phase system can be seen. The AND-gates of figure 3.14 have been replaced with respectively a flip-flop, NOR-gate, and NAND-gate. The specific implementation for the insertion logic is shown as well.

The detection of a possible disappearance of the low phase is done by the flip-flop. When a rising edge comes in, the flip-flop samples its input which is connected to the chain behind the insertion logic. If still a '1' is inserted in the chain, the rising edge gets too close to the falling edge, and consequently the flip-flop will have a high output which signals a collision fault. The point in the chain where the clock input of the flip-flop is connected to thus determines the allowed length of the low phase. Placing it more in front of the insertion logic makes sure only bigger low phases are allowed.

When a collision fault occurs, the logic in figure 3.16 will make sure the low phase before and high phase after the incoming rising edge are erased completely. This is done by making one input of the NOR-gate high, so that the arrival of the rising edge will not be seen. There are two reasons for this. The first reason is to make sure no weird low glitch remains after a collision fault. A collision fault means that there is a rising edge nearby a falling edge which can be seen as a low glitch. Such a glitch may trigger some flip-flops in the register and some not. Also it may or may not trigger the flip-flop of the contamination system or the flip-flop of the high phase system in the next stage.

The other reason is to make sure that both the SR-latches connected to the chain will not get a set and a reset signal simultaneously. The rising edge is not allowed to propagate any further, so it will not respectively reset and set the SR-latches. The other set and reset, coming from the NAND-gate in the next stage, can do their job without being interfered then. As told in section 3.2.3 and [12], if an SR-latch gets both a reset

Figure 3.16: The design of the high phase system.

and set at the same time, it can lead to an internal racing. Such a situation is prevented by not letting through the rising edge if there is a collision fault.

The just described glitch and internal racing may not be a disaster after all, because the control logic of the whole pipeline will see the collision fault and will send a reset (see Section 4.3.2). The reset will make sure the glitch, the possible consequences of the glitch, and the internal racing will be removed. It is up to the designer to decide if a glitch and internal racing is allowed or not. If it is allowed, removing the connection between the flip-flop and NOR-gate in figure 3.16 is everything it takes to allow this. However, this will only save two transistors. On the other side, preventing an internal racing will reduce power consumption, so that is why the connection is there in figure 3.16.

Just as with the flip-flop within the contamination system, the flip-flop in figure 3.16 has an asynchronous input. As told in section 3.2.4 this may lead to a meta-stable state of the flip-flop. Everything that has been said in section 3.2.4 regarding meta-stability also applies here. There is one extra situation here regarding meta-stability which deserves attention though:

A meta-stable output of the flip-flop may be problematic for example if both NOR-gates connected to the output read this as a different value. If the NOR-gate connected to the SR-latch reads this as a high value, and the other gate will read this as a low value, then there will be no collision fault signalled, while actually the whole incoming rising

edge is discarded. Consequently the whole pipeline will malfunction. A meta-stable detection circuit can prevent such a situation however, because it makes sure a collision fault is signalled in such a case.

3.4.5 Constraints of the high phase system

Just as with the previous systems, some important constraints apply to this system as well. The first constraint makes sure the output of the flip-flop reaches the input of the NOR-gate in time.

$$\text{Distance } clk-NOR > clk-Q \text{ delay } ff \quad (3.7)$$

Distance clk-NOR is the time it takes for a rising edge to travel from the clock input point of the flip-flop to the first point in the chain where the NOR-gate is connected to. It is marked in figure 3.16 as well.

The next constraint makes sure that once a rising edge comes in at the high phase system, it can propagate and will not be hold for a while by the circuit which inserts a '0' (the first transistor circuitry).

$$\begin{aligned} \text{Distance } NOR-(\text{'0' insertion}) &> \text{delay } NOR + \text{delay } SR\text{-latch} \\ &+ \text{switching delay insertion logic} \end{aligned} \quad (3.8)$$

Distance NOR-(\text{'0' insertion}) is also shown in figure 3.16, and it is also the time it takes for a rising edge to travel from the beginning point to the endpoint of the marked length in figure 3.16.

The following constraint is a more complicated one, because it concerns the length of the pulse given by the NAND-gate in the next stage.

$$\begin{aligned} \text{Received pulse width next stage} &> \text{delay } SR\text{-latch} + \text{switching delay insertion logic} \\ &+ \text{setup time } SR\text{-latch} \end{aligned} \quad (3.9)$$

When there is a pulse from the next stage, this pulse will let the first SR-latch be set, and as a consequence the next SR-latch will get a high input value at the $\bar{S}$ input. The pulse from the next stage must also reset this next SR-latch, so that a '1' will not be inserted anymore into the chain. In order to get a reset, the $\bar{S}$ input needs to be high. The constraint is there to guarantee that there is a period at which the set is high and the reset low. It thus might occur that both the set and reset are low. This will not lead to a malfunctioning of the latch ([12]), because both the set and reset will not return to a high value at the same time, because of constraint 3.9. The *setup time SR-latch* is there to guarantee there is enough time between the rising of the $\bar{S}$ and falling of the $\bar{R}$ input in order for the latch to have a high value on output $\bar{Q}$.

In order to allow the pulse width in constraint 3.9 to be as small as possible, a NOR-based SR-latch is used instead of a NAND-based latch. It takes only one NOR-gate delay for the set signal to affect the $\bar{Q}$ output. As a consequence, *delay SR-latch* in constraint 3.9 will also be one NOR-gate delay then.

A far more important reason to use the NOR-based SR-latch is its property that $\bar{Q}$ is low when input S is high regardless the value of input R. It will mean that on a reset output $\bar{Q}$ is low and always a '0' is inserted into chain. It does not matter if a rising edge comes in, because still a '0' will be inserted in the chain. Such behaviour will turn out to be quite important in Section 4.3.2.

The final constraint defines where the insertion logic must be placed into the chain in order to get a certain high phase length:

$$\begin{aligned} \text{'0' insertion point} = \text{wanted high phase} \times \text{number of chain inverters} \\ + \frac{2 \times \text{delay NAND} + \text{delay SR-latch} + \text{switching delay insertion circuit} + \text{wire delay}}{\text{inverter delay}} \end{aligned} \quad (3.10)$$

wanted high phase is expressed as a fraction of the total chain length, and *'0'-insertion point* is the point in the chain at which the first transistor logic (is the same as the hold circuit in Figure 3.11) must be placed. It must be placed after an even amount of inverters.

It takes a while before the signal from the NAND in the next stage gets to the insertion logic. Meanwhile the rising edge is already propagating in the next stage. To account for this the fraction is added in constraint 3.10. The lowest possible delays for the gates and wire delays in the numerator must be applied, and the inverter delay in the denominator needs to get the highest value. In such a way the high phase will never be smaller than *wanted high phase*.

3.5 Cost and overhead of the added systems

In this chapter three systems are discussed which are to be added to every pipeline stage along with the inverter chain. It is interesting to see what the overhead is of the systems and the inverter chain in terms of area and power. This is rather hard to say since this depends on the specific pipeline to which the systems and the chain are going to be added. The precise area and power usage also depend on the technology used for implementation, and therefore nothing concrete can be said yet.

However, an indication of the area usage can be retrieved by examining the number of transistors and flip-flops used for the three systems per stage. This does not account for the area used for wiring, but it still does give an indication on the area usage. Table 3.1 gives an overview of the number of transistors used in each system. Assumed is that the gates are implemented in complementary CMOS and that gates like the NAND and NOR require 4 transistors if a two input gate is used. The transistor count for a flip-flop is unknown since it depends on the specific implementation, and therefore flip-flops are mentioned in Table 3.1. Noteworthy is that the cost for the extra contamination detection system does not include the extra inverter chain needed.

It is difficult to get an idea of the area and power usage of an inverter chain, since it depends mostly on the length of the logic stage. An indication can be retrieved however from [3], because an inverter chain is used in this paper as well. In this paper the logic stages are composed by small arithmetic logic units which add up to a total of 250 gates

Contamination detection system	flip-flop + 36 transistors
Hold system	28 transistors
High phase system	flip-flop + 50 transistors
Gates used for all systems	12 transistors
Total cost per pipeline stage	2 flip-flops + 126 transistors
Cost of extra contamination detection system	flip-flop + 32 transistors

Table 3.1: Hardware cost of the systems added to the inverter chain.

for each stage. The inverter chain is made of 50 inverters. With 90 nm CMOS technology, $V_{dd} = 1$ Volt, and a used clock frequency of 25 MHz the area overhead is lower than 20% of the entire circuit area, and the increase in power consumption is around 10%. The overhead is considerable, but it must be noted that the logic stages are quite small.

The area usage of the systems is most likely comparable to the area usage of the inverter chain, because in [3] the inverter chain uses 100 transistors. Of course this latter amount can vary depending on the logic in the pipeline stage. It is quite difficult to say something about the power usage of the systems, but given the nature of the switching in these systems the power consumption will be much less.

The overhead in terms of area and power is still relatively small when comparing it to the already existing techniques mentioned in Section 1.2. The overhead in terms of area, and also most likely power, is smaller than with Razor [10] since the overhead is not at the flip-flop level, but at the pipeline stage level. When comparing the overhead to TEAtime [21] it will be more or less equal, because TEAtime uses more or less also the principle of adding an inverter chain.

Fault handling

Both the contamination and high phase system discussed in the last chapter can signal a fault. When such a thing happens, appropriate actions need to be taken in order to let the AICBP output valid computation results. In this chapter it is described how contamination and collision faults are solved for an AICBP.

4.1 Actions to be taken when a fault occurs

Both the contamination and phase system in each stage can signal respectively a contamination and collision fault. If one of these faults happens, appropriate actions must be taken. This section describes the logic which makes sure the pipeline always gives correct results despite the contamination and collision faults.

On an occasion of a contamination fault there is actually a set of data lost, because it is actually overwritten by the upcoming set of data belonging to the next rising edge. There is no way of getting back the lost data, because they are overwritten. The original data, which were inserted into the stage before the contamination happened, are also not saved, since the upcoming data are saved in the register at the beginning of the stage. Therefore the whole execution needs to be redone. The same story goes for a collision fault as well. In such a case a rising edge will disappear, and the data which belonged to this edge will disappear as well. Also these data cannot be retrieved back anymore.

In order to be able to redo the whole execution, all inserted data must be saved in a memory, so that the data can be inserted later when an execution needs to be redone. When there is a contamination or collision fault, the control logic of the whole pipeline will let the original data be retrieved from the memory and inserted back again in the pipeline. In such a way the calculations in the pipeline are redone, and most likely there will be no faults this time, because all the logic will suffer from (slightly) different delays this time (Section 2.1).

Figure 4.1 shows a four stage pipeline with a piece of memory (FIFO). The figure is a very basic one, because its intention is only to make clear the different components that are used. The FIFO memory is used to be able to redo executions in case of a fault. Also some control logic can be seen, which is there to send a reset to all stages. Such a reset is necessary to not let the contamination or phase system signal a fault anymore. However, the reset signal is also used for other purposes as is told in the next section.

Another important thing to note regarding figure 4.1 is about the signals which come out of the middle of every inverter chain. Each signal coming out of a chain is an input for an OR-gate, and the output of all the OR-gates signals a contamination or collision fault. It must be noted that the output of an inverter chain in figure 4.1 is an OR of the contamination and collision fault signal coming from respectively the contamination detection and high phase system.

Figure 4.1: Overview of the circuitry needed for fault treatment.

4.2 Flushing the pipeline

This section describes the definition of a flush, and why such as flush is necessary. The next section describes how flushing can be done.

In case of a contamination or collision fault there is no way of telling which set of data is contaminated or lost. Rising edges will propagate with different speeds, and thus the clock period might be different from the propagation time of the chain in one stage. To give an example, imagine there is a contamination fault in the fourth stage of a pipeline. This does not necessarily relate to the data inserted four clock periods ago. The rising edge, and thus also the data, could have travelled very fast for example, so that it was actually the data inserted three clock periods ago. Another option could be for instance that the propagation times of the chains are larger in comparison to the clock period. In such a case it could be that the contamination fault relates to the data inserted five clock periods ago.

Summarizing, there is no way of telling which rising edge was inserted when, unless a sequence number is added to the pipeline which will identify the rising edge. Such a sequence number will consume quite some area and power, because in every stage flip-flops for saving the sequence number are needed. Also logic is needed to not let the sequence number travel very fast from the beginning to the end of the stage, because then contaminations will occur very soon. In order to avoid these costs, it is chosen not to identify the rising edges.

Since rising edges are not identified, it is not known which set of data must be reinserted in reaction to a fault. Consequently a total flush of the pipeline is needed. A flush means in this case that all the sets of data that were inserted, and whose results are not yet out of the pipeline, are inserted back again. There is no way of telling which set of data went wrong, so that is why everything is put in again.

4.3 Details on flushing

4.3.1 Bookkeeping to identify sets of data

Reinserting all the data into the pipeline again can be done with the help of a FIFO-memory. When there is no fault, the FIFO pushes the inserted data into its queue. The same data, which has been pushed in, will be popped out when the results of the pipeline calculations are known. Summarizing, when a rising edge is inserted into the pipeline, the belonging data are pushed into the FIFO, and when the same rising edge comes out of the pipeline, the belonging data are popped out of the FIFO. The FIFO thus only contains data which are inserted in the pipeline and of which the results of the calculations on these data are not out of the pipeline yet. If a flush is needed, all the data in the FIFO needs to be reinserted then, because the FIFO contains all the inserted data which are not out of the pipeline yet.

In order to pop the right data out of the FIFO, it must be known to which inserted set of data a rising edge belongs when it comes out of the pipeline. In the pipeline this cannot be determined like is said earlier. Outside of the pipeline this is possible though, if there is some bookkeeping done at the beginning and end of the pipeline. In case of no disappearance of a phase, the first rising edge which comes out of the pipeline belongs to the first inserted edge. The second edge belongs to the second inserted one, and the same applies for the third edge and all the other following rising edges. In such a way it can easily be determined which inserted data belongs to which out coming edge.

The bookkeeping is only possible when no phase disappears. Disappearance of a phase will mean a rising edge is missing, and the data which originally belonged to this one will suddenly belong to some other edge. Consequently the whole bookkeeping will be a mess. Disappearing of a phase can be detected however, so appropriate measures can be taken to preserve the bookkeeping.

4.3.2 Resetting in order to kill all high phases

To preserve the bookkeeping, there is a special role for the reset signal. The high phase system of section 3.4.4 is made in such a way that it will kill all the high phases in the pipeline chains if there is a reset. All the high phase systems in each chain will then keep inserting a '0' into the chain regardless the values in the chain. If such an insertion is done long enough, the whole chain will have a low value, and there will be no more high phases. Thus making the reset last long enough will kill all the high phases in the chains.

If a fault occurs, the control logic of the whole pipeline will send a reset to all pipeline stages. The high phase systems will receive this reset as well, and because of the fact that the reset will be long enough, the high phase systems will kill all the high phases. As a result, no more rising edges will come out of the pipeline after a fault. In fact, the first rising edge that will come out after this reset is the one belonging to the reinserted data. The control logic will reinsert data from the FIFO after the reset, so the rising edge belonging to these data will come out first, since all the other rising edges are wiped away.

4.3.3 Specific usage of the FIFO

The bookkeeping can always be maintained, because when one phase disappears a reset will be given, and consequently it is known that the first rising edge that comes out of the pipeline belongs to the reinserted data. The same thing goes for a contamination fault as well, because the same actions will be taken. The so called bookkeeping is very simple now. It only comes down to pushing data in the FIFO when a rising edge is put in, and popping data out when a rising edge comes out of the pipeline. If a rising edge comes out of the pipeline, the belonging data must be valid, so the data can be popped out, because the data will not have to be inserted again. As for the pushing, data must only be pushed when there is no flush, because in case of a flush the data in the FIFO must be inserted again. The data in the FIFO will have to be popped out with a flush and be inserted in the pipeline again, because it contains all the data which has not reached the output. This must not be a real pop action though, because the data may have to be reinserted again since the flush itself can give a fault as well.

In figure 4.1 the push and pop input of the FIFO are connected to respectively the clock input and *Valid* output of the pipeline. This is done to make clear that the pushing is controlled by the rising edges which are put in, and the popping is controlled by the rising edges which come out of the pipeline. It must be noted that in case of a flush no pushing is done by the FIFO, because then all the data, which is in the FIFO, will be inserted again.

4.4 RTL design of fault treatment logic

This section describes the RTL design for the fault treatment logic. Figure 4.2 shows the RTL design, which is discussed in this section. It does not show the control and reset logic however, because these are discussed later on.

There is one important thing to note before discussing the RTL design in more detail, and it concerns the control logic. It is important to locate the control logic near the chain of the first stage, because a reset needs to be given almost instantaneously. More on the reason for this is explained in 4.6 when the state machine of the control logic is discussed. On the other hand it is important to let the distance between the fault information and the input of the negative edge flip-flop be as small as possible. If the distance is small, a reinsertion can occur faster, which will improve performance as is discussed in 4.4.2. In order to satisfy both wishes, it is a wise thing to account for this when laying out the pipeline on the die. The beginning and end of the pipeline should be located next to each other, so that both wishes can be satisfied.

There is no discussion on the meta-stability detector in figure 4.2 in the next subsections, because the same reasoning as in section 3.2.4 can be applied here as well. It must be noted though that the meta-stability issue can be solved here in a different way as well. Concatenating multiple flip-flops behind each other can make the probability of a meta-stable output very small. This is done in many systems that sample asynchronous inputs. The problem with this solution is that the delay for the *Clkd_fault* will be multiple cycles. Such a thing will degrade performance, and therefore it is chosen to use a meta-stability detector.

Figure 4.2: The RTL design of the fault treatment logic.

The next subsections discuss important things regarding the RTL design. These are the following things:

- FIFO
- reinsertion of data
- constraint regarding added latency in RTL design
- glitch removal circuitry

4.4.1 FIFO

Figure 4.2 shows the RTL design for the fault treatment logic. The FIFO, which was earlier shown in figure 4.1, is made here of a piece of memory and a head and tail

counter/pointer. As the words suggest, the head keeps track of the head of the queue, and the tail pointer is responsible for pointing at the tail of the queue. The tail is updated by the rising edges which come out of the pipeline, and the head is updated by the edges which go in, because it is connected to the general clock. This corresponds to the FIFO usage described in the last section.

The temporary head pointer is used to serve as a head pointer during a flush. If there is a fault, at first instance the tail pointer is used to reinsert the right data into the pipeline again. After that the following sets of data from the queue need to be inserted. The temporary head pointer is used to accomplish that, because it will keep track of what is being inserted from the queue. Whether everything is inserted from the queue can be detected with the comparator. If so, new data from outside can be put in the pipeline again.

The memory must be big enough to be able to contain all the data which can be in the pipeline at the same time. It must be accounted for a situation where all the chains, and thus also the logic, are very slow, so that the pipeline will contain lots of data. Another important thing to note is that the memory must have a size which is a power of two. Such a thing is quite normal of course, but it is very important here. The pointers will have a size which is a logarithm of the size of the memory, and will work as a kind of modulo counter. For example, in case of a memory with size 16, all the pointers will count to 15 and start counting again at 0, because the counters are only 4 bits wide. It is important that the whole range of the pointer must point to a valid memory location.

4.4.2 Faults and reinsertion of data

A striking thing in the design is that a fault signal in the early stages of the pipeline will take a while to propagate to the end of the pipeline, where it is the input for the AND-gate. This fault signal needs to go through all the OR-gates at every stage, and the wiring to get from one stage to another may be quite long. When the pipeline has many stages it will certainly take a considerable amount of time, since then it can take several cycles perhaps. This will not hurt performance though, because in the meantime when the fault signal is travelling, the pipeline is still outputting valid data. When the fault signal reaches the end of the pipeline, the pipeline is not putting out valid data anymore. Then the performance of the pipeline is influenced though, because nothing is coming out anymore. It is important that the fault signal is sampled as soon as possible by the negative edge triggered flip-flop in figure 4.2, because only after that data will be reinserted. Data must be reinserted as soon as possible, because the pipeline is not outputting any data because of the fault, and this will hurt performance.

The occurrence of a fault is sampled by a negative edge triggered flip-flop, because the reinsertion must occur as fast as possible. After the sampling of the fault, it is important to start reinserting data as fast as possible to keep performance as good as possible. Figure 4.3 shows a time line with the events which take place in case of a fault and a following flush. One can see that only the half of a clock period is needed before data are reinserted. To accomplish this, the negative edge flip-flop controls also some multiplexers in order to reinsert data in the next rising edge. After this next rising edge,

Figure 4.3: Timeline with all the events during a flush.

the control logic has seen the fault and will change the multiplexers in such a way that the temporary head counter is used.

The time between detection fault sampling and reinsertion is thus half a clock cycle, but this must be made possible by the logic in figure 4.2 though. Most likely the critical path concerning this reinsertion is from the input of the tail pointer register, through the register and the adder, and then to the input of the temporary head register. In case of a large memory, the critical path could also be in the memory. Regardless where the critical path is, it must be smaller than half a clock period to assure a correct working of the fault treatment logic. If such a thing cannot be guaranteed, the RTL design and control logic will need to be adjusted, so that there will be one cycle in between the fault detection and reinsertion.

4.4.3 Constraint

It might happen that in the last stage a contamination fault occurs. Such a fault might be detected just before the rising edge comes out of the chain. The data belonging to this rising edge is invalid then, because it is contaminated. In such a case the AND-gate at the end of the pipeline is there to make sure the rising edge is destroyed, so that the data will not be used. To assure this, the following constraint has to apply:

$$\begin{aligned}
 \text{Added latency} &> 2 \times \text{delay NAND} + \text{delay inverter} + \text{delay SR-latch} + \text{Clk-Q delay ff} \\
 &\quad + \text{delay meta-stability detector} + \text{NOR-delay} + \text{delay SR-latch} + \text{delay OR} \\
 &\quad \quad \quad + \text{delay inverter}
 \end{aligned}
 \tag{4.1}$$

The constraint makes sure the fault signal reaches the AND-gate earlier than the rising edge in case of a contamination fault. In specific, a contamination fault which occurs just before the rising edge gets out of the last chain.

Figure 4.4 tries to make clear constraint 4.1. It shows the contamination detection system and clarifies where each delay in constraint 4.1 comes from. The latest possible contamination fault is when the reset from the *return path* is just a fraction later than the clocking of the flip-flop. Consequently the flip-flop will signal a contamination fault (or a meta-stable value), and this contamination fault will go to the end of the pipeline via the *fault path*. While all of this is happening, the out coming rising edge, which is in the added latency chain, is still travelling. The out coming edge may not reach the final AND-gate before the information from the *fault path* comes in. That is why the *added latency* must be bigger than the *return path* and the *fault path*.

Figure 4.4: Clarification of the added latency constraint.

Unfortunately, the *added latency* has a negative impact on performance. In normal operation the *added latency* has no impact on the throughput, but it increases the latency of the pipeline though. An increased latency is not a problem most of the time, because the throughput is often the most important criterion. The *added latency* influences the throughput however in case of a flush.

Imagine that there is a fault in a stage, but not in the last stage. At the same time a rising edge is at the last register of the pipeline. The fault signal will travel to the end of the pipeline, and will disable the valid output. If *added latency* is quite long, the fault signal gets to the AND-gate earlier than the rising edge, and consequently no valid data comes out of the pipeline anymore. In case of a shorter *added latency* the fault signal

might get later to the AND-gate than the rising edge. When this occurs, one set of valid data comes out of the pipeline, whereas with a longer *added latency* this would not be the case. Thus the *added latency* has its influence on the flush performance, because it makes the difference whether valid data can come out just before a flush.

The impact on performance caused by the *added latency* is most likely not very big, because flushes are not going to occur that often. Also the impact on a specific flush is not that very big. If a set of data cannot get out of the pipeline because of the longer added latency, it will have to be reinserted, and the duration of the flush is only lengthened with one set of data.

The designer can choose to reduce *added latency* though. It can be done by placing the NAND-gate, which causes the reset pulse, before the clocking point of the register. As a consequence, the signals in the return path will start travelling earlier, and the clocking of the flip-flop has to occur earlier as well in order to signal a contamination fault. Consequently, the fault path will also signal the fault earlier. The earlier signalling allows the *added latency* to be shorter. There is a downside about this construction though. When moving backward the NAND-gate, the other connections to the contamination detection system will have to be moved backward in the inverter chain as well. These other connections will have to be moved backward with a bigger distance to account for variations in the propagation of the rising edge. This in turn can have a negative influence on performance as well.

4.4.4 Glitch removal circuitry

The RTL design of figure 4.2 includes a glitch removal circuit. This circuit is needed, because the AND-gate connected to the valid signal can give a glitch. Such a glitch can occur when a fault is detected before the last stage. The glitch is created if a rising edge comes out of the AND-gate and a fraction later a fault signal comes in. The result is that the AND-gate gives a short high-value, because the output is made low again by the incoming fault signal.

A glitch may cause the flip-flops in the tail register not to function correctly. Also the upcoming logic connected to the valid signal may work incorrectly. How the flip-flops in the tail register will react on a glitch depends on the specific implementation of these flip-flops. With a number of flip-flop types some flip-flops will react on the glitch and some will not. Of course this is very problematic, since the tail register will not have a correct value anymore then.

Glitches can be removed with the help of a simple capacitor at the place of the glitch removal circuitry. Such a thing is equal to a low pass RC-filter which is shown in Figure 4.5. A possible problem though is that the rising edge becomes less steep, and that may be a problem for certain flip-flop implementations. Another glitch removal circuit, which does not have this problem, is shown in figure 4.6 and comes from [2]. Unfortunately this circuit is patented, so that may be a problem on its own.

The specific solution for the glitch problem is left up to the designer, because the glitch problem also depends on the specific implementation of the flip-flops in the tail register. Also the allowed length of the glitches depends on the implementation of the flip-flops, and this length will influence the specific glitch removal circuitry.

Figure 4.5: Low pass RC filter.

Figure 4.6: Glitch remover circuit from [2].

4.5 Resets given to all the stages

Figure 4.2 does not show the specific reset circuitry which is used in case of a flush. This specific circuitry is shown in figure 4.7 though. *Reset_1st_stage*, *Reset_2nd_stage*, and *Reset_other_stages* are controlled by the control logic, and are the reset input for the contamination, hold, and high phase system in the respectively first, second, and other stages.

Figure 4.7: Reset circuitry.

The most striking thing is the circuitry needed for resetting the systems in the first stage. This is needed to make an immediate reset possible of the first stage. If a fault is sampled by the negative edge flip-flop in figure 4.2, it will immediately reset the first stage, because *clkd_fault* will let the first stage reset. As is told in section 4.3.2, the high phase system will consequently remove the high phase from the chain if there is one. The data to be reinserted, along with its belonging rising edge, will already be inserted on the next rising clock edge, so that is why the first stage is reset as early as possible. Resetting will clear the inverter chain from high phases, so that the new reinserted data and edge can propagate through the first stage without any problems.

In order to clear all the chains from high phases, a reset to a specific chain in one stage must be given long enough. When all the high phases are removed, it can be guaranteed that the reinserted data will come out of the pipeline as first, and that it can travel through the pipeline without any holding of some of the hold systems. The first chain must thus also get a reset which is long enough, but the reset on the other hand must not interfere with the new reinserted rising edge. Therefore it is chosen to let the reset stop on the falling clock edge after the new reinserted rising edge. The reset given to the first stage has a duration of one clock cycle, so in most cases the old high phase has been removed.

Figure 4.8: Advanced reset circuitry.

In some specific AICBP setups it might happen that the new reinserted rising edge reaches the high phase system when there is still a reset. Such a thing might happen for example when the insertion circuit (see Section 3.4.4 and 3.4.5) of the high phase system is placed at the half of the chain, because in half a clock cycle the rising edge can surely reach the insertion circuit. Problem in this case is that the rising edge is not going to continue after the reset is gone, because the NOR-gate is not going to generate a reset pulse anymore. Whenever such a situation might occur, the circuitry of figure 4.8 can be used, because it prevents the rising edge from reaching the inputs of the NOR-gate in Figure 3.16 during a reset. The insertion of a '0' can be done with the circuitry in figure 3.11, and it is possible to combine the insertion at the end of the chain with the hold circuitry of the hold system (section 3.3).

In most AICBP setups the insertion circuit is not placed at the half of the chain (see section 3.4.3.2), and consequently the new reinserted rising edge may not be able to reach the insertion circuit in half a clock cycle under any circumstances. In such a case

the circuitry of figure 4.8 is not needed.

Another exceptional situation which may occur in some AICBP setups is that the rising edge is already behind the insertion point of the phase system when the reset is given to the first stage. This might occur when a rising edge is able to travel further than the insertion point in half a cycle. Such a thing is problematic since the rising edge can continue travelling and might have its negative influence in the future. The solution is also the circuitry of figure 4.8, because it contains a hold/insertion circuit (Figure 3.11) which will hold the rising edge, so that it cannot leave the stage. The reset will last for one clock cycle, so the high phase behind the fast rising edge is also killed then.

The other resets given to the other stages will also make sure the chains are cleaned from all high phases. The duration of the reset will be two clock cycles or more. This duration will make sure all the high phases will be gone, because in two clock cycles a rising edge, which is in front of an insertion point of the phase system in one chain, will certainly reach the insertion point in the next stage. In such a case the high phase, belonging to this rising edge, will be removed. The second stage is given a reset which only will last for one clock cycle, but this is not a problem. No rising edges will come out of the first stage due to the beginning of the reset at a negative clock edge or the use of the circuitry in figure 4.8.

In some extreme AICBP setups the circuitry in figures 4.7 and 4.8 might not be enough. Also the duration of two cycles for the reset might not be enough. This could be the case when for example the clock period is less than half of the propagation time of one chain in a stage. A solution would be to put more '0' insertion circuits in the chain, and to increase the number of cycles *reset_other_stages* will take. The specific solution is left up to designer, since this situation will occur only in more extreme AICBP setups.

More on when a certain reset is going to occur can be seen in the state machine discussed in the next section.

4.6 Control logic

In Figure 4.9 a finite state machine can be seen which describes the behaviour of the control logic. The finite state machine is a synchronous Moore machine ([12]), and the specific outputs belonging to a certain state are shown at the right side. Transitions will only happen at a rising clock edge, and transitions with a name attached to it will only happen in case of the event described in the name. Thus transitions without a name will always be executed on the next rising clock edge, and transitions with a name will only be executed on the next rising clock edge if the condition is true.

It is important to have the control logic and the negative edge flip-flop in Figure 4.2 located near the first stage, because then a reset of the first stage can be given almost instantaneously. As told in section 4.4, this is important to be able to reinsert data as fast as possible. On the other hand, the consequence is that it will take more time for the *reset_other_stages* signal to reach all the other stages, since they are located further away from the control logic. As a result, it will also take more time for fault signals to be reset in later stages. It might be the case that a fault is not reset before the control logic enters the *Recovery* state. This is problematic, because the control logic then reacts on an old fault, and that is of course not necessary. To solve this issue, some states can

Figure 4.9: Finite state machine of the control logic.

be inserted between *Recovery second cycle* and *Recovery*. These states will have to have the same outputs as the *Recovery* state, but they will not react on faults.

4.7 Conclusions regarding the fault handling

The discussed logic in this chapter is able to let the pipeline augmented with the inverter chain and systems do a re-execution in case of a contamination or phase fault. This guarantees that the pipeline always puts out valid data and that the execution order is still maintained. The drawback is that a re-execution costs time and thus it influences performance in a negative way. This negative influence becomes bigger when the pipeline contains more stages, since re-executions will take more time then.

The area and power overhead of the fault treatment logic is made up by the RTL design of Figure 4.2, reset logic discussed in Section 4.5, and the control logic discussed in Section 4.6. The precise overhead is difficult to tell, since it depends on the implementation of all the components and the technology which is used. An idea of the overhead can be retrieved by adding up all the different components as is done in Table 4.1.

Basic RTL design	(pipeline length) $\times$ OR, 1 AND, 1 OR, 1 NOR, 1 inverter, 3 multiplexers, 3 adders, 3 registers, 1 flip-flop, meta-stability detection circuit, glitch removal circuit, memory
reset logic	2 inverters, 2 AND's, 1 OR, 1 flip-flop
Control logic	3-bit state register, logic for determining next state, logic for determining outputs

Table 4.1: Hardware cost of the fault treatment system surrounding the AICBP.

The area and power overhead of the logic discussed in this chapter is relatively small in comparison with a whole pipeline, except for the FIFO-memory. This is probably the most expensive part of the AICBP and it could be a wise thing to integrate this FIFO-memory into a perhaps already existing input memory as is suggested in the next chapter.

The cost overview of Table 4.1 does not include the extra logic needed for interconnection with the surrounding logic. The cost of this one depends on the system into which the AICBP is integrated, and therefore nothing can be said about this. Something can be said though about the surrounding logic and this is done in the next chapter.

Interconnection and clock frequency

5

It may be obvious that the connection of the AICBP with the surrounding logic is different from what it would have been when a standard pipeline is used. Rising edges are not coming out of the AICBP precisely every clock period, due to the delay variations of both the combinational logic and the inverter chains. Also fault handling will have its effect on when valid data is coming out of the AICBP. This is one of the reasons why the interconnection with the surrounding logic is different and it is discussed in more detail in Section 5.1.

Another question, which also relates to the surrounding logic, is what the particular clock frequency for an AICBP should be. As will turn out in Section 5.2 this is not an easy question, and Section 5.2 explains why.

5.1 Interconnection with surrounding logic

As is stated earlier, the interconnection of the AICBP with the surrounding logic is different from a standard pipeline. The surrounding logic could be for example a memory system, a bus, or another island if the AICBP is used in the Casta DIVA platform [9]. Normally the logic at the output would react on a rising clock edge, but now it has to react on the rising edge of the *valid* signal in figures 4.1 and 4.2. This is not a problem mostly, but the time between two rising edges of the valid signal may be smaller or larger than the clock period. The logic has to account for the time between two rising edges being smaller than the clock period. The critical path in this logic thus needs to be smaller than it would have been when a clock signal is used for triggering. There is however a guaranteed minimum amount of time between two rising edges, and that one is assured by the hold systems.

The time between two rising edges may not only be smaller than a clock period, but on the other hand it may be much larger than a clock period due to a re-execution. The logic connected to the output of the AICBP needs to be aware that there may be a time span of multiple clock cycles between two rising edges. Therefore it might be a good idea to buffer the output data of the AICBP if this is problematic.

Also the logic which gives input data to the pipeline might have to be changed in comparison with a traditional pipeline. For example, if another IP-block, or island from the Casta DIVA platform, feeds data to the pipeline at every clock edge via a bus, the IP-block needs to be aware of re-executions. During a re-execution the AICBP cannot accept any data, because it needs to reinsert data. In such a case, the control logic of the AICBP must signal to this IP-block that sent data is not accepted. The *is_recovery* output signal of the finite state machine in Figure 4.9 can be used for that. An alternative would be to use a buffer, so that the IP-block can keep sending data, but this uses extra area on the die.

When a memory system or buffer is used as an input for a pipeline, it could be an option to discard the FIFO. If the contents of the memory are not going to change, the same data can be retrieved from the memory again in case of a reinsertion. This would be very advantageous, since the FIFO is probably going to consume the most area on the die of all the AICBP components.

5.2 The determination of the clock frequency

The question which remains is what clock frequency to choose for the AICBP. The goal of the input clock frequency is to have a maximum average output frequency. The average output frequency is made up by the number of valid outputs the AICBP can give within a certain amount of time. This is influenced by the input clock frequency and also by the number of re-executions. During a re-execution AICBP is not outputting anything, so this has a negative impact on the average output frequency. There is actually an optimal clock frequency for which the average output frequency is the highest. When increasing the clock frequency beyond this point, the number of re-executions increases in such a way that the average output frequency is worsened. The reason is that when increasing the clock frequency the sets of data are closer to each other in the pipeline, and the chance to get a contamination is increasing. Consequently, more contamination faults take place and thus more re-executions are required. Not only more contamination faults occur, but also more collision faults may occur depending on the chosen high phase length.

It is not easy to tell what the optimal clock frequency is, because it depends on a number of factors as follows:

- The contamination and critical path delay ratio plays an important role, because if this ratio is increased, contamination faults are more unlikely. Making the contamination delay larger may have its price though in terms of area and power, so this is a trade-off between performance and area and power usage.
- The specific delay variations influence the number of contamination faults. If small contamination delays occur more often, the number of contamination faults can increase. This may vary from die to die, so the number of contamination faults may differ per die, and consequently the optimal clock frequency can be different for each die. In order to know the specific delay variations and to determine the best clock period for the average die, one has to carefully study the combinatorial logic in the pipeline stages as well as the fabrication technology specific variations.
- The position of the hold system also has an influence on the number of contaminations since it determines how close the rising edges, and thus also the data, can get to each other. The position which can give the highest average output frequency depends in its turn on the contamination and critical path delay ratio and on the specific delay variations. The relation between these three factors becomes more clear when looking at the simulation results in Chapter 6.

Noteworthy is that the used clock period must not be smaller than the critical path delay of the RTL design in Figure 4.2. More specific, there is a critical path in the RTL design which may not be larger than half a clock period as is told in Section 4.4.2.

Simulation and performance results

6

In order to get an impression of the throughput improvement an AICBP can give, a simulation in VHDL is done. A pipelined array multiplier is augmented with all the systems an AICBP has. With this simulation setup a preliminary evaluation of the performance improvement is made. A simulation in VHDL is at a rather high-level, and to get a more precise evaluation of the performance a low-level simulation with for example SPICE is a very good idea. Unfortunately, a low-level simulation is out of the scope of this thesis, since such low-level simulations would require too much time.

First the pipelined multiplier is described, and after that the specific implementation of this multiplier in VHDL is enlightened. At the end of this chapter the simulation results, and thus also the performance results, are discussed.

6.1 Pipelined array multiplier

The principals of a pipelined array multiplier are explained in this section. The structure of a non-pipelined array multiplier is depicted in Figure 6.1. This particular multiplier accepts two operands with a size of five bits and the outcome will of course be 10 bits. All the rectangles in the figure, except for the lowest four blocks, are multiply-add blocks. Each of these multiply-add blocks multiplies one bit from the x operand with one bit from the a operand. The result of this multiplication is added to the other two inputs. The outputs are a result bit and a carry bit. By connecting all the multiply-add blocks like has been done in Figure 6.1, operand x and a are multiplied with each other. It is important to note that also an adder is needed to convert the final carry save format to a standard binary representation. This adder is composed by the lowest four rectangles in Figure 6.1. More information and explanation on this array multiplier can be found in [18].

The array multiplier in Figure 6.1 can be pipelined by inserting registers between two levels of multiply-add blocks. With a level it is meant a horizontal row of multiply-add blocks in Figure 6.1. Also registers must be placed on the lines of operand a to the multiply-add blocks to let every pipeline stage get the right operand. Operand x must also completely be passed on from one pipeline stage to another. This can be done by passing operand x from one register to another in every pipeline stage. In every pipeline stage one or more bits from this operand are used for calculating the partial products belonging to this pipeline stage. The bits that are already used for calculating a partial product do not have to be passed on to the next pipeline stage though. In order to enlight the just described pipelined array multiplier, Figure 6.2 has been made.

The array multiplier shown in Figure 6.1 has an operand size of 5, but this can easily be made larger. The consequence is that more multiply-add blocks will have to be used then. More blocks are needed not only to accommodate the extra width of the operand,

Figure 6.1: Example of an array multiplier from [18].

Figure 6.2: A stage of the pipelined array multiplier.

but also because more levels of multiply-add blocks will be needed. The reason for this is that more partial products will need to be calculated. If in a pipelined array multiplier

the levels of multiply-add blocks per pipeline stage are kept the same and the operand size is increased, the length of the pipeline is going to be longer. The reason is that there are more levels of multiply-add blocks then and if the number of levels are not changed in each stage, the total number of pipeline stages will be larger.

The reason why a pipelined array multiplier is chosen as a pipeline for simulation is that the number of stages can easily be varied. Such a thing can be done by changing the operand size and keeping the number of levels the same for every pipeline stage. In such a way a performance comparison can be made between different pipeline lengths.

One remaining question is how to implement the multiply-add blocks. There are a number of implementations possible, but one with gates which are representative for an average pipeline is preferred in this case. Therefore the implementation depicted in Figure 6.3 has been chosen as an implementation for the multiply-add blocks. It may not be the cheapest implementation in terms of area, but the implementation in Figure 6.3 is a two level sum of products implementation and is therefore quite a representative implementation.

Figure 6.3: Implementation of the multiply-add block.

6.2 VHDL implementation of the pipelined array multiplier

From [1] some VHDL code has been retrieved which describes a pipelined array multiplier. This code has been slightly modified and all the AICBP systems are added to it as well. This section describes some of the details of this VHDL implementation.

The VHDL implementation follows the structure of Figure 6.1 and in particular the

structure of Figure 6.2. As in Figure 6.2, each pipeline stage has four levels of multiply-add blocks in the VHDL implementation. With the help of VHDL generic statements the operand size, and thus also the length of the pipeline, can easily be changed.

In the VHDL implementation the inverter chains are made by using so-called signals and by assigning a delay to these signals an inverter chain can be simulated. Not only signals are assigned a delay to simulate a chain, but also all the gates in each stage are assigned a delay as well. The most interesting question is which delays should be assigned to the gates and the chain, but also when these delays are assigned is an interesting question as well. Also the correlation between the delay of the gates and the inverter chains is of interest.

6.2.1 Properties of the gate and inverter chain delays

The correlation between the delay of the inverter chain and the delay of the gates strongly depends on layout and technology. These factors are not known, and therefore a 100% correlation is chosen. This means the critical path will have the same delay as the inverter chain, and both will vary with the same amount. This is actually too optimistic in comparison with a real implementation, but since the correlation is not known and only an impression of the performance improvement has to be obtained, a 100% correlation is chosen. It can thus be said that:

$$\text{inverter chain delay} = \text{critical path delay} = 9 \times \text{gate delay} \quad (6.1)$$

To keep things fairly simple it is assumed that all the gates in each stage have the same delay. At least, that is the case for one particular set of data which is traveling through the pipeline stage. More is told on this later on.

When looking at the time domains of the mechanisms which influence delay, like is discussed in Section 2.1, the delay variance can roughly be split up into two parts. One part of this delay variance causes the delay of the gates and chain to vary each cycle. The other part of the delay variance will let the deviance of the delay from the mean delay be fixed throughout the whole simulation. The simulation will only take about 100,000 cycles and therefore the influence most of the mechanisms have on the delay is not going to change during simulation. Shortly, it can thus be said that there is a static and dynamic delay variation.

The VHDL simulation is set up in such a way that all the gates and inverter chain in one stage get a new delay every cycle according to a gaussian or normal distribution. This is thus the dynamic delay variation. Besides that there is also a delay variation which is not going to change during the simulation and that one is the static delay variation. Summarizing, the delay for all the gates in a particular stage is given by the stochastic variable $DELAY[stage_nr]$:

$$DELAY[stage_nr] = \mu_{gate_delay} + DYNAMIC_DELAY[stage_nr] + STATIC_DELAY[stage_nr] \quad (6.2)$$

μ_{gate_delay} gives the standard mean delay of a gate and each of all the random variables has a normal or Gaussian distribution. If μ_{gate_delay} is the mean delay of one gate, the delay of the whole pipeline stage, or else said the inverter chain delay, is:

$$\begin{aligned} \text{inverter_chain_delay}[stage_nr] &= 9 \times \text{DELAY}[stage_nr] = 9 \times \mu_{gate_delay} \\ &+ 9 \times \text{DYNAMIC_DELAY}[stage_nr] + 9 \times \text{STATIC_DELAY}[stage_nr] \end{aligned} \quad (6.3)$$

For $\text{DYN_DELAY}[stage_nr]$ and $\text{STATIC_DELAY}[stage_nr]$ applies:

$$\text{DYNAMIC_DELAY}[stage_nr] = N(0, \sigma_{dynamic}^2) \quad (6.4)$$

$$\text{STATIC_DELAY}[stage_nr] = N(0, \sigma_{static}^2) \quad (6.5)$$

In such a way the following distribution for $\text{DELAY}[stage_nr]$ is obtained:

$$\text{DELAY}[stage_nr] = N(\mu_{gate_delay}, \sigma_{total}^2) \quad (6.6)$$

It is assumed that $\text{DYNAMIC_DELAY}[stage_nr]$ and $\text{STATIC_DELAY}[stage_nr]$ are independent stochastic variables, so:

$$\sigma_{total}^2 = \sigma_{static}^2 + \sigma_{dynamic}^2 \quad (6.7)$$

In reality the two stochastic variables are not independent, but the precise correlation is quite difficult to obtain, since this depends heavily on the used technology and the precise implementation of the design. Therefore a worst-case assumption is made by assuming the two stochastic variables are independent.

For all simulations applies:

$$-3\sigma_{dynamic} \leq \text{DYNAMIC_DELAY}[stage_nr] \leq 3\sigma_{dynamic} \quad (6.8)$$

This is to prevent extreme low and high delays which are very rare. In reality these very extreme delays will not be very likely, because if all gates in a stage can get such an extreme value this will most likely be detected with the testing of the chip. It must be said though that this of course depends on the specific technology and testing method. Another reason to filter out these extreme values is the use of fixed delays for all the AICBP systems as is told in the next section 6.2.3. When the chain delay becomes extremely small, the '0' insertion point of the phase system will be placed in the next stage. This is problematic as will be better explained in Section 6.2.3.

6.2.2 Delay mechanism in VHDL simulation

At the beginning of the simulation each stage is given a random delay according to $\mu_{gate_delay} + \text{STATIC_DELAY}[stage_nr]$. This is the fixed delay and it is given to all the gates within one stage. It is not going to change during the simulation. To make the simulation repeatable, the specific fixed delay for each stage can be specified before simulation. Once the simulation is running, the dynamic delays are assigned to the gates as well. Every time a rising edge arrives at one stage, all the gates are given a new random dynamic delay value. This one is added to the static delay variation and that is how equations 6.2 and 6.7 are satisfied.

Noteworthy is that once a rising edge enters a new stage, not all gates are given a new delay value directly. This is because previous data may still be in the pipeline. If the new delay values are given to all the gates in the stage directly, this previous data may start propagating with a different speed than the belonging rising edge. Consequently things may go wrong then, because the edge may arrive earlier than the data. To prevent such a situation, gates only get a new delay when the rising edge is in the neighborhood. To give an example, if the edge is at the half of the chain, then the gates at $\frac{5}{9}$ of the critical path are given a new delay, and this delay belongs to the rising edge.

In order to be able to assign delays with a normal distribution, a function is needed that returns random values with a normal distribution. There is unfortunately no such function in VHDL. There is however a pseudo-random generator that generates random numbers with a uniform distribution. Luckily, this distribution can be transformed into a normal distribution with the help of the Box-Muller transform [7]. The pseudo-random generator generates a sequence of random numbers which are always the same if the same seeds are used. Consequently, the random numbers which are converted to a normal distribution do also have the same sequence if the same seeds are used. As a result, each simulation is repeatable which is very important.

6.2.3 Specific delay values

This section describes some particular delay values for specific parts of the simulated AICBP.

In the VHDL simulation the setup and hold time of the registers are omitted. First of all, it is unclear how big these setup and hold times should be, because these depend on the specific implementation of the flip-flop and also on the technology which is used. Besides that, the purpose of the VHDL simulation is to get a comparison with a traditional pipeline. If with this traditional pipeline the hold and setup times are also omitted, a good comparison can still be obtained.

In order to have a more simple delay model, the delay of the inverters in the multiply-add block (Figure 6.3) are ignored. Another assumption to simplify the delay model is that all gates in this block have an equal delay. In such a way the critical path through this block is three gates, and the smallest contamination delay is two gates. There are, like in Figure 6.2, 4 levels of multiply-add blocks in one pipeline stage. As a result, the critical path is 9 gates and the smallest contamination delay is 2 gates.

In order to get a good performance improvement, a larger contamination delay than 2 gates is desired as will turn out in Section 6.3. To get a larger contamination delay, a delay can be added to the most right lines in Figure 6.2 which connect the outcomes of the first and second level to the end of the pipeline stage. If enough delay is added, the smallest contamination delay is in the last level, because it will take three gate delays before the changes of the operands are visible at the outputs of the multiply-add blocks of this last level. To make this contamination delay larger than three gates, the connection to the operand register can be delayed as well.

A delay must be assigned as well to the direct connections between the operand registers. If no delay is assigned to these direct connections, the contamination delay would be zero and the performance would thus be dramatic. Therefore these connections

are assigned a delay, and this delay equals the wanted contamination delay/critical path fraction times the chain length. It means that the delay of these direct connections equals the delay of the path that determines the smallest contamination delay.

A remaining question is how big the delays are of all the specific AICBP systems mentioned in chapter 3. Also the influence the delay of these gates has on performance is interesting. In contrast to what one might think, the specific delays used for these systems do not really influence the performance of an AICBP. If bigger gate delays are used for these systems, *position clock point* (section 3.2.3), *release point* (section 3.3), and *'0' insertion point* (section 3.4.5) are just placed at a different position in the inverter chain and the performance will still be the same. However, if the delay of these gates can vary and the variance is quite big, the placing further on or earlier in the chain cannot be done that much. Consequently, performance is reduced, because for example contamination faults are signalled earlier and collision faults will occur more often as well. The reason for the last one is that high phases are made larger with a large gate delay variance, because the *'0' insertion point* is not shifted that much to the right in the chain.

Another performance influencing thing to note though is the *added latency* in section 4.4.3. If the gate delays in the systems are larger, then the *added latency* will be bigger and this will have a slight negative impact on performance as is told in 4.4.3.

The delay for all the gates in the systems of chapter 3 is fixed and it does not vary in time. The reason is that it is unclear how much these gates should vary and also what the correlation is with the delay of the other gates. In order to still test all the constraints in chapter 3 and to have a representative length of added latency (section 4.4.3) some fixed delay is taken. The particular fixed delay, which goes for every type of gate in the systems of chapter 3, is $\frac{1}{5}$ of the mean gate delay of one gate in the multiply-add block of Figure 6.3. There are two reasons for this:

- A gate delay which has a duration of $\frac{1}{45}(\frac{1}{5} \times \frac{1}{9})$ of the mean critical path delay is quite representative. For example, a pipeline with a 32-bit adder in one stage will have gate delays which are less than $\frac{1}{45}$ of the mean critical path.
- When a duty cycle of 20% is desirable, the gate delays cannot be made any larger. If they are made larger, the *'0' insertion point* will be placed according to equation 3.10 in the next stage or it interferes with the hold circuit of the hold system. This is at least the case with one of the smallest gate/chain delays possible in one of the simulation setups in the next section.

The delays for the gates and registers in the RTL design of Figure 4.2 are assumed to be zero. These gates will not influence the performance as long as the critical path in this RTL design is smaller than the clock period, or half of the clock period as is mentioned in 4.4.2. Therefore the delays are taken to be zero.

6.3 Simulation results

6.3.1 Simulation with only dynamic delays

The following parameters are chosen for the first simulation setup:

- High phase length = 0.2 (*wanted high phase* in equation 3.10). A high phase length of 20% is doable for most inverter chains, because there is still quite some space between the rising and falling edge and thus the disappearance of the high phase is very unlikely. The high phase is still a lot smaller than the standard 50%, so chances of a collision fault are thereby drastically reduced.
- contamination delay/critical path delay = $\frac{5}{9}$. Delays are added in the way like is said in Section 6.2.3 to get this contamination delay/critical path delay. It is noteworthy to mention that the paths at the end of the stage are made longer than $\frac{5}{9}$ of the chain length, because these paths are not very correlated to the $\frac{5}{9}$ part of the chain which is used for contamination detection.
- delay variability: $\sigma_{total} = \sigma_{dynamic}$ and $\frac{\sigma_{total}}{\mu_{gate_delay}} = 0.1$. A delay variance of $\frac{\sigma_{total}}{\mu_{gate_delay}} = 0.1$ is quite representative according to [14] and [17] for the variance of a logic stage, and this may actually be already too severe. The variance is assumed only to be dynamic in this first simulation. There are an infinite amount of simulations possible with all kinds of static delays, and therefore in this first simulation no static delay variation is taken. Instead of that the total delay variance of $\frac{\sigma_{total}}{\mu_{gate_delay}} = 0.1$ belongs to the dynamic delay variance.
- Simulation length = 100,000 clock cycles. The simulation is run for 100.000 clock cycles, and it is evaluated how much valid data comes out of the pipeline during this period. The specific simulation length of a 100.000 clock cycles will give performance results which are more or less the same with different seeds for the pseudo-random generator. More clock cycles will lengthen the simulation time which is especially problematic for a 16 stage pipeline and therefore a 100,000 cycles is chosen.

Figures 6.4, 6.5, and 6.6 show the simulation results of respectively a 4, 8, and 16 stage pipeline for this simulation setup. The values in these graphs are percentages of the clock period used in a traditional pipeline. It is assumed that this clock period is equal to $\mu_{gate_delay} + 3\sigma_{total}$. According to [5] the clock period of a traditional pipeline is larger than $\mu + 3\sigma$, so this assumption would have a negative influence on the performance results. On the other hand, the inverter chain is made just as long as the critical path length and no safety margin is added to the inverter chain. This in its turn should have a positive influence on the performance results. It is unclear whether these two assumptions cancel out each other his effect, because this depends heavily on technology and the specific implementation. Nevertheless, it is assumed that by making these two assumptions a more or less realistic result is retrieved, because one has a positive and the other one has a negative effect on the simulation results.

The horizontal axis of the graphs in the figures 6.4, 6.5, and 6.6 is the used (input) clock period, and the vertical axis is the average period at which the pipeline outputs data. Due to also the contamination faults the AICBP will not regularly output data, and therefore the average output period over 100,000 cycles is shown. The simulation results for different hold system positions are shown. More specific, the spacing between

Figure 6.4: Performance of a 4 stage AICBP (100,000 clock cycles)

Figure 6.5: Performance of an 8 stage AICBP (100,000 clock cycles)

two rising edges generated by the hold system is shown. This is equal to *wanted spacing* in Equation 3.6.

Appendix chapter A shows the same simulation results, but in this chapter also the number of contamination and collision faults can be seen. With the number of faults it is actually meant the number of times the final OR-output, which is the final OR of all the stage fault outputs in Figure 4.2, goes from low to high. As a result, if two contamination faults happen shortly after each other this will only be registered as one fault. Once the re-execution is being done and the fault signals are reset, a new fault can be registered for the graph which shows the number of faults. In such a way it is registered how many faults occur which actually do cause a re-execution. The number of faults are given as a percentage of the total amount of clock cycles in a simulation. If for example during a simulation run of 100,000 cycles a 1,000 faults are detected, the fault percentage is 1%. Faults which occur during a re-execution are thus also included.

6.3.2 Simulation with static and dynamic delays

In order to make a simulation with static delays a division between the static and dynamic delays has to be made. According to Table 2.1 in Section 2.1.2 at least 32% of the total delay variance is caused by mechanisms which influence the delay in the time domain of a clock period. There is still a part however (10%) which is not in the time domain of a clock period, but it is not a static delay either. To account for this one and still have

Figure 6.6: Performance of a 16 stage AICBP (100,000 clock cycles)

a reasonable simple simulation setup, a dynamic delay impact of 40% on the total delay variance is taken.

As said earlier in Section 6.2.1, the dynamic and static delay variations are treated as independent statistical processes. Therefore Equation 6.7 applies and the question now is how big $\sigma_{dynamic}$ and σ_{static} should be in order to have a total delay variance of $\frac{\sigma_{total}}{\mu_{gate_delay}} = 0.1$.

When assuming two independent statistical processes, two processes with $\sigma_{independent}$ are used. When looking at Table 2.1 it is stated that the delay impacts in the right column are on the 3σ value. Therefore it is said that:

$$\begin{aligned}\sigma_{dynamic} &= 0.4 \times \sigma_{independent} \\ \text{and} \\ \sigma_{static} &= 0.6 \times \sigma_{independent}\end{aligned}\tag{6.9}$$

This is because the dynamic part has an impact of 40% on 3σ or σ and the static part has an impact of 60%. Using Equation 6.7 the following equation can be made:

$$\sigma_{total}^2 = (0.4\sigma_{independent})^2 + (0.6\sigma_{independent})^2\tag{6.10}$$

The value for $\sigma_{independent}$ can be solved by using $\frac{\sigma_{total}}{\mu_{gate_delay}} = 0.1$. With Equation 6.9 the values $\frac{\sigma_{dynamic}}{\mu_{gate_delay}} = 0.06$ and $\frac{\sigma_{static}}{\mu_{gate_delay}} = 0.08$ are obtained.

It is rather difficult to do a simulation for all possible static delays. Therefore four particular interesting static delay situations are taken for a simulation. These are the following:

- Static delays are equal for each stage. For all stages applies $STATIC_DELAY[stage_nr] = 0$. This is certainly not a situation which is common, but it could occur on some dies.
- Increasing static delays. Each stage has a static delay which is bigger than the previous stage. For a four stage pipeline the following sequence of static delays for each stage is taken:
 $-\sigma_{static}, 0, \sigma_{static}, 2\sigma_{static}$

The first mentioned $2\sigma_{static}$ delay is applied to the first stage, the second σ_{static} to the second stage and so on and so forth. For an eight stage pipeline the following sequence is taken:

$$-\sigma_{static}, -\sigma_{static}, 0, 0, \sigma_{static}, \sigma_{static}, 2\sigma_{static}, 2\sigma_{static}$$

- Decreasing static delays. Each stage has a static delay which is smaller than the previous stage. The sequence for a four stage pipeline is:

$$2\sigma_{static}, \sigma_{static}, 0, -\sigma_{static}$$

And for an eight stage pipeline:

$$2\sigma_{static}, 2\sigma_{static}, \sigma_{static}, \sigma_{static}, 0, 0, -\sigma_{static}, -\sigma_{static}$$

- One stage with a long static delay. Static delays are equal for all stages, except for one. This one stage has a rather long static delay. The sequence for a four stage pipeline is:

$$0, 0, 2\sigma_{static}, 0$$

And for an eight stage pipeline:

$$0, 0, 0, 0, 0, 2\sigma_{static}, 0, 0$$

The simulation results are shown in appendix Section A.2.

6.3.3 Simulation with reduced contamination delay

A contamination/critical path delay ratio of $\frac{5}{9}$ may be quite expensive to achieve in terms of area and power, and that is why a simulation with a smaller contamination/critical path ratio is quite interesting. In appendix Section A.3 simulation results are shown for a contamination/critical path delay ratio of $\frac{1}{3}$. The rest of all the simulation parameters are the same as for the simulation in Section 6.3.1.

6.4 Simulation evaluation

This section evaluates the results of the simulations described in the last section.

6.4.1 General remarks

It can be seen in every simulation that there is an optimal clock frequency like is stated in Section 5.2. In figures 6.4, 6.5, and 6.6 at the optimal clock frequency the performance improvement is 35%, 28%, and 16% for respectively a 4, 8, and 16 stage AICBP.

A striking thing that can be seen is that the position of the hold system is affecting the performance. Placing the hold system earlier in the chain, like $\text{Spacing} = 0.6$ in figures 6.4, 6.5, and 6.6 worsens performance, because the occurrence of small contamination delays increases the number of contaminations. This also explains the difference between figures 6.4, 6.5, and 6.6. The 8 stage AICBP has more stages than the 4 stage AICBP at which a small logic delay may cause a contamination fault, and consequently the performance difference with other hold system positions is bigger. The same applies for

the difference between an 8 and 16 stage AICBP as well. Placing the hold system later in the chain will prevent small contamination delays causing a contamination fault, since the data and rising edge are hold longer at their position.

Holding the data and rising edge too long has a negative effect though as can be seen for the spacing = 0.8 case. The chances are in such a case quite big that the upcoming data are contaminating the hold data. The chance of such a situation happening increases as the clock frequency becomes higher. The sets of data are situated closer together in the pipeline, and if just one set of data is hold too long, the next set of data may contaminate it. Another side effect is that one set of data which is hold may cause other sets of data to be hold later on as well and they in turn may be contaminated. It is a kind of situation which is comparable to a traffic jam on a road.

6.4.2 Relation between faults and performance

Both contamination and phase faults cause the average output period to be higher than the input period. In appendix chapter A also the number of faults are shown next to the average output period, and it can be seen that there is a relation between the number of faults and the average output period. The number of faults is by the way given as a percentage of the total amount of clock cycles in a simulation and only faults which cause a re-execution are registered. The relation between the number of faults and the average output period can be roughly described with the following equation:

$$\begin{aligned} \text{average output period (\%)} &= \text{input clock period (\%)} \\ &+ \text{re-execution duration} \times \text{input clock period (\%)} \times \frac{\text{faults (\%)}}{100\%} \end{aligned} \quad (6.11)$$

re-execution duration is the number of cycles a re-execution takes. The equation makes sense, because the average output period is determined by the input clock period plus the time needed for all the re-executions. However, the equation does not take into account the fact that faults may occur during a re-execution as well, and consequently the re-execution may start all over again. The following equation does take this into account:

$$\begin{aligned} \text{average output period (\%)} &= \text{input clock period (\%)} \\ &+ \text{re-execution duration} \times \text{input clock period (\%)} \times \sum_{i=1}^{\infty} \left(\frac{\text{faults (\%)}}{100\%} \right)^i \end{aligned} \quad (6.12)$$

The summation is a convergent geometric series ([19]) and because of that the equation can be rewritten to:

$$\begin{aligned} \text{average output period (\%)} &= \text{input clock period (\%)} \\ &+ \text{re-execution duration} \times \text{input clock period (\%)} \times \left(\frac{1}{1 - \frac{\text{faults (\%)}}{100\%}} - 1 \right) \end{aligned} \quad (6.13)$$

This equation is still however nothing more than a theoretical approach of the average output period. One of the inaccuracies is that it assumes that if a fault occurs during a re-execution another full re-execution is needed. This is not the case though most of the times, because if a fault occurs during a re-execution the re-execution is just done all over again and most of the times this will not take *re-execution duration* amount of cycles.

Equation 6.13 can be rewritten to get the *re-execution duration* for all the different simulations. When the input period equals the mean stage delay, the *re-execution duration* turns out to be more or less 5, 9, and 17 cycles for respectively a 4, 8, and 16 stage AICBP in the different types of simulations. When reducing the input period, *re-execution duration* becomes bigger. This makes sense, because the delay of the stages stays the same while the input period is decreasing.

The reason for a re-execution duration of 5, 9, and 17 is that an average re-execution without overhead takes 4, 8, and 16 cycles and half of a cycle is needed for detection of the fault (Section 4.4.2). Also added latency (Section 4.4.3) worsens the re-execution duration, and therefore the re-execution duration is actually lengthened with almost one cycle.

6.4.3 Reducing delay variance

Reducing the delay variance just as in appendix Section A.2.1 really improves performance. Due to smaller delay variations contaminations are not going to occur that often, and this in turn improves the maximum average output period. The performance improvement for a 4 stage AICBP is even 46% in Section A.2.1. The ideal hold system position is also different. The delay variations are smaller and consequently the spacing between sets of data can be smaller without a contamination. Therefore the hold system can be placed earlier in the chain, because sets of data do not have to be hold that long in order to prevent a contamination.

Another effect of reducing the delay variance is that the performance difference between small and large pipelines becomes less. The performance difference between a 4 and 8 stage AICBP is only 4% at the best average output period in Section A.2.1. Due to the fact that a very small amount of faults occur, the negative impact of a long re-execution is less noticeable in the average output period. The performance difference is still big however for the less advantageous hold system positions.

6.4.4 static delay variance

Adding static delay variations of course has a negative effect on performance as can be seen in Section A.2. It must be said that the effects of a static delay variation are less worse for performance than a dynamic delay variation. For the simulations in Section A.2 a fraction of the dynamic delay variation is actually replaced by a static delay variation. As can be seen in Section A.2, the performance is much better in comparison with a simulation with fully dynamic delay variations and this applies even for different kinds of static delay variations. Static delay variations are thus not as bad for performance as dynamic delay variations. The reason is that the static delay variations apply for all the sets of data in the pipeline. Consequently, sets of data will not be really moved closer to

each other due to static delay variations. Of course when a certain stage in a pipeline has a small static delay and the following stage a large static delay, the data, or rising edges, are moved closer to each other. This is only a temporary situation most likely, because the spacing between the rising edges is restored when further on in the pipeline there is a slow stage followed by a fast stage.

As for the different static delay simulations in Section A.2, the simulation with the decreasing static delays gives better performance results than the one with the increasing delays. The reason is that for all the sets of data in the pipeline applies that the delay of the stages gets slower and slower. As a result, the rising edges and sets of data get closer to each other while they are approaching the end of the pipeline. Obviously, this will lead to more contaminations and thus less performance improvement. One other remarkable thing is that the performance difference between spacing = 0.5 and spacing = 0.6 is less with increasing static delays. It makes sense, because when rising edges get closer to each other it is often better to not hold the rising edge too long at its place, since the next rising edge may get too close.

6.4.5 Reducing contamination delay

Not very surprisingly the performance improvements are less when the contamination delay is reduced. When making the contamination/critical path delay ratio $\frac{1}{3}$, the performance improvement is only 21% for a 4 stage pipeline and 9% for an 8 stage pipeline. It means that there is a gap of 14% and 18% in comparison with the contamination/critical path delay ratio of $\frac{5}{9}$ for respectively a 4 and 8 stage pipeline. It is thus of utmost importance to make the contamination/critical path delay ratio as big as possible in order to make the performance improvement as big as possible. Of course making this ratio bigger has its price as well, because it will most likely be more expensive in terms of area and power. There is thus actually a kind of trade-off to be made between performance and cost in terms of area and power.

Another striking thing for the simulation with a contamination/critical path delay ratio of $\frac{1}{3}$ is that the ideal hold system position is different from the simulation with the ratio of $\frac{5}{9}$. As is stated in Section 5.2 the contamination/critical path delay ratio influences the ideal position of the hold system. In this case the ideal position is later in the chain, such that a set of data is hold longer in order to not let the hold data contaminate the data in front. Due to the smaller contamination delay the chances of a contamination are increased and therefore the data must be hold longer.

6.4.6 Relation between delay variance, contamination delay, hold system position, and average output period

When looking thus far at the discussed simulation results it is obvious that there is a relationship between the maximum average clock period, the delay variance, contamination/critical path delay ratio, and the ideal hold system position. Increasing or decreasing both the delay variance and contamination/critical path delay ratio has its effect on the ideal hold system position and the maximum average output. Table 6.1 makes clear the relation between these factors.

	Ideal hold system position	Maximum average output period
Increasing delay variance	moves towards end of chain	becomes bigger
Decreasing delay variance	moves towards beginning of chain	becomes smaller
Increasing contamination/critical path delay ratio	moves towards beginning of chain	becomes smaller
Decreasing contamination/critical path delay ratio	moves towards end of chain	becomes bigger

Table 6.1: Relation between delay variance, contamination delay, hold system position, and average output period

6.5 Concluding remarks regarding the simulation

As can be seen from the simulation results in appendix chapter A the performance of a pipelined structure can be considerably improved by even up to 46%. Noteworthy is that a lot of assumptions have been made in the simulations which do not fully match the circumstances which apply for a real chip. Therefore it is a good idea to do some low level simulations in the future. It may seem, while taking this fact into account, that 46% is the upper limit of the performance improvement. This is certainly not the case. The 46% performance improvement is achieved with a contamination/critical path delay ratio of $\frac{5}{9}$. This ratio can easily be increased, and the result will be even a better performance improvement with the current simulation setup. Even though performance might perhaps be worse than the performance achieved with the current simulation setup, the 46% performance improvement and beyond can still be achieved for a real chip by making the contamination/critical path delay ratio big enough.

Out of order execution

Whenever a fault occurs in the AICBP, a re-execution of all the data in the pipeline is done. At least, that is the scheme which is explained in Chapter 4. This re-execution of all the data is not very efficient however, because a lot of times only one set of data is contaminated. Thus there is actually no need for re-executing the rest of the data in the pipeline. This chapter proposes a technique which can partly eliminate this expensive re-execution and hence improve performance.

7.1 Out of order execution

The main reason why all the data in the pipeline is re-executed in case of a fault is to preserve the execution order. More specific, the sequence of computed output data must have the same order as the sequence of the belonging input data. This is necessary to identify the specific sets of data which come out of the AICBP. Besides that, the execution order must sometimes be maintained due to data dependencies as well.

There are however pipelined structures which allow by themselves out-of-order execution. Processors which are optimized to avoid stalls can issue instructions at a later position in the instruction stream. Once the instructions are computed the results are buffered, so that the results can be re-ordered when the results of the stalled instructions are available as well. In such a way the output data order equals the program order, and this may be important for interrupt handling. The out-of-order execution technique is fairly complex, and in particular the checking for data dependencies when allowing an out-of-order execution is quite complex.

When integrating the AICBP in a processor which can do an out-of-order execution, it could be an idea to exploit the out-of-order execution hardware for AICBP re-execution purposes. More specific, when a fault occurs for a specific set of data in the pipeline, this set of data can be re-executed on its own without re-executing all the other data in the pipeline. The out-of-order execution hardware can decide if the single set of data can be re-executed on its own without violating data dependencies. Also the hardware can manage the buffering of all the output data until the re-execution of the single set of data is done.

How the precise integration of AICBP with the out-of-order execution hardware must be done is out of the scope of this chapter and thesis. The goal of this chapter is only to explain how out-of-order execution within the AICBP can be made possible. The integration of AICBP with the out-of-order execution hardware can be very complex and also depends on the specific processor chosen to be integrated within. Therefore the emphasis of this chapter is only on making the out-of-order execution possible for the AICBP itself.

7.2 Out of order execution for AICBP

In order to re-execute a specific set of data, it must be known which specific set of data must be re-executed. As is explained in Section 4.2, identifying the sets of data within the pipeline is rather difficult and that can only be done in a simple way at the end of the pipeline. Therefore a setup with an augmented contamination signal is chosen. An extra signal next to the inverter chain is added, and this signal signals if the belonging data is contaminated. Such a setup is shown in Figure 7.1. Every contamination detection system in each stage checks whether a contamination occurs, and if so, it uses the extra augmented inverter chain to signal this. When a set of data comes out of the AICBP the final outcome of the extra augmented inverter chain can be checked to see if the outgoing data is contaminated. If this is the case, a re-execution can be issued for this particular set of data. The advantage of this scheme is that the re-execution only has to be done for this particular set of data and not for the rest of the pipeline.

Figure 7.1: Simple overview of the logic needed for out-of-order re-execution.

An inverter chain is used for the transportation of the contamination fault signal to prevent the fault signal from racing through the stages. For example, if no inverter chain is used a fault signal may race through two flip-flops at once and overwrite the contamination fault information belonging to another set of data or rising edge. To prevent such overwriting the contamination fault inverter chain must have a delay which is longer than the smallest contamination delay or else said the shortest path through the logic. The fault signal actually equals a set of data which is travelling through a stage, and to prevent a contamination of the contamination fault signal itself the chain must be longer than the smallest contamination delay.

The augmented chain for contamination detection can not be used for detecting the disappearance of a phase in the inverter chain. If a phase disappears, the signal in the augmented contamination inverter chain is not going to propagate into the next stage, because the flip-flop connected to this chain will not be clocked. Therefore the already existing fault signalling is still used, and the fault handling described in Chapter 4 is still done in case of a collision fault.

To support the scheme of Figure 7.1 the contamination detection system has to be adjusted. The next section describes the adjusted contamination detection system and its constraints.

7.3 Out of order contamination detection system

7.3.1 General working

The wanted behaviour for this particular contamination detection system is more difficult than the one in Section 3.2. It must signal a contamination, but it must stop signalling when the next rising edge comes by. This is to prevent the following set of data from being marked as contaminated as well. This kind of behaviour makes the contamination detection system more complex, and there are also more constraints to comply to.

Figure 7.2: The contamination detection system for out-of-order re-execution.

The design of the out-of-order contamination detection system is depicted in Figure 7.2. The system in Figure 7.2 can detect a contamination in the same way as described in Section 3.2. When detecting a contamination, a high value is put on the input of the upper right flip-flop and this flip-flop will clock this high value. By doing so

the set of data, which is clocked by the register on the right side of Figure 7.2, is marked as contaminated. This particular high value, or marking of a contamination, will travel along with the rising edge and data to the end of the pipeline. At the end of the pipeline the contamination will be noted and a re-execution for the particular data will be done.

Shortly after detecting a contamination, the flip-flop connected to the SR-latch will be reset by a pulse which is generated when the rising edge belonging to the contaminated data enters the next stage. The data which belong to the rising edge which has just passed the NAND-gate connected to the SR-latch will now not be signalled as contaminated. It might be the case though that these data will be contaminated as well, and then the upper right flip-flop is given a high value again. Consequently this set of data will be marked as contaminated as well then.

When detecting a contamination two rising edges are actually involved. These edges get too close to each other and this is signalled. It may be the case though that a third edge is involved as well. Before the edge belonging to the contaminated data can get out of the current stage, a third edge may arrive at the contamination detection system. The lower flip-flop in Figure 7.2 will detect this, and it will signal a contamination fault which is equal to the contamination fault in Chapter 4. The consequence of signalling this particular contamination fault is that a re-execution of all the data in the pipeline is done just like is illustrated in Chapter 4. There are thus two different contamination faults: One that will only cause the re-execution of the particular contaminated data, and one that is equal to the contamination fault in Chapter 4. This latter contamination fault is still there, because it is quite difficult to let the design of Figure 7.2 still work without a full re-execution in case of a third edge. The occurrence of a third edge at the contamination detection system is quite rare however, so that is why a full re-execution is done in case of such a third edge.

The contamination detection system in Figure 7.2 is actually working as a kind of shift-register. The SR-latch and the two flip-flops keep track of how many rising edges there are between the contamination detection system and the end of the stage. If there is one edge this is no problem, but if there are two a contamination fault is signalled. In case of three edges a contamination fault is signalled as well, but this fault will cause a full re-execution of all the data in the AICBP.

7.3.2 Constraints

There are a number of constraints where this contamination detection system must comply to, and therefore it may be quite complex to implement this specific contamination detection system. The first of the number of constraints is the following:

$$\begin{aligned} & \text{distance NOR-NAND} + \text{delay NAND} > \text{delay NOR} + \text{delay clk-Q ff} \\ & + \text{delay meta-stability detection} + \text{delay OR} + \text{delay inverter} + \text{delay NAND} \end{aligned} \quad (7.1)$$

When a rising edge arrives at the contamination detection system it lets two flip-flops clock and one SR-latch to be set. Constraint 7.1 is there to make sure no reset can occur while the SR-latch is being set, because it makes sure there is enough time to make the input of the NAND-gate connected to the reset of the SR-latch low. This assures

the SR-latch cannot be reset anymore once the rising edge is setting the SR-latch. No resetting means no internal racing of the SR-latch [12], and it also assures a correct working of the contamination detection system. A reset is not prevented however when the output value of the flip-flop in Constraint 7.1 becomes '0' after clocking. This is not problematic, because when this is the case no reset will occur since there is no rising edge in front that will leave the stage.

The following constraint is there to make sure the reset, caused by a rising edge leaving the current stage, will only reset the flip-flop connected to the SR-latch.

$$\begin{aligned} \text{width reset pulse} &< \text{delay added latency} + \text{delay reset ff} \\ &+ \text{delay meta-stability detection} + \text{delay OR} + \text{delay inverter} \end{aligned} \quad (7.2)$$

By making sure the width of the reset pulse is not too big, it can be assured that not both the flip-flop and SR-latch are reset when the flip-flop has a high output value. To keep the principle of a shift register, it is necessary to only let the flip-flop be reset in case the flip-flop has a high output value.

A kind of special situation may occur when the flip-flop is triggered by the NOR-gate and a reset is given at the same time. To still have a functioning system, the flip-flop needs to be constructed in such a way that the clocking is disabled when a reset is given. It actually comes down to disabling the clock input when the reset signal is high. When the flip-flop is constructed in such a way, a situation might occur that when the reset signal is gone the output of the NOR-gate is still high. Consequently the flip-flop will still sample its input. This is no disaster, but it must be prevented that the flip-flop can clock twice. The following constraint has to apply to prevent this:

$$\text{width NOR-pulse} < \text{width reset pulse} \quad (7.3)$$

When the NOR-pulse is smaller than the reset pulse, the flip-flop cannot be triggered twice, because the output of the NOR-gate cannot be high before and after the reset pulse. Another constraint regarding the situation of both the pulse of the NOR-gate and reset occurring at the same time is the following:

$$\text{Added latency} > \text{delay NAND} + \text{delay SR-latch} \quad (7.4)$$

After the reset signal is gone the output of the NOR-gate may still be high and consequently the flip-flop may be triggered. To let the flip-flop sample the right value it must be made sure that the effects of the reset on the SR-latch are visible at the input of the flip-flop. Things may go wrong for example when the output of the SR-latch is still high when the flip-flop samples its input because of the interfering reset and NOR-pulse. Actually the flip-flop should have sampled a low value, because the reset will finally set the SR-latch to zero. By adding latency to the reset of the flip-flop it is made sure the effects of the reset are visible at the input of the flip-flop.

In some cases it might be that the flip-flop is implemented in such a way that it is only triggered by the rising edge of the pulse generated by the NOR-gate. Consequently it will not be triggered at the end of a reset pulse when the output of the NOR-gate is still high. This may become problematic when a third edge arrives at the contamination detection system while two rising edges have not left the current stage. When the first

rising edge gives a reset and the third edge generates a NOR-gate pulse simultaneously, the flip-flop connected to the SR-latch will register a low value. This is problematic because this actually signals that there is only one rising edge after the contamination detection system while there are actually two.

To solve this problem the flip-flop which registers the third edge can be placed more upfront in the chain. By placing the flip-flop far enough in front, the just described situation will not be a problem, because a contamination fault which causes a total re-execution is signalled then. To assure this the following constraint needs to apply:

$$\begin{aligned} \text{distance } clk\text{-}NOR &> \text{width reset pulse} - \text{delay } NOR - \text{delay reset ff} \\ &\quad - \text{delay meta-stability detection} - \text{delay OR} \end{aligned} \quad (7.5)$$

It makes sure the third edge will register a contamination fault before the pulse of the NOR-gate can interfere with the reset.

An unanswered question thus far is where to place the contamination detection system in the chain. The reasoning for where to place it is quite similar to the explanation in Section 3.2.3 for Constraint 3.3. The constraint for where to place the first to the chain connected input of the NOR-gate is as follows:

$$\begin{aligned} \text{Position lower input } NOR &= \frac{\text{Smallest contamination delay}}{\text{Critical path delay}} \times \text{number of chain inverters} \\ &\quad + \frac{\text{placement deviation} - \text{delay } NOR - \text{hold time register}}{\text{inverter delay}} \end{aligned} \quad (7.6)$$

Where *placement deviation* is defined as follows:

$$\text{placement deviation} = 2 \times \text{delay } NAND + \text{added latency} \quad (7.7)$$

The position of all the other surrounding connections to the chain can be deduced from this constraint with the help of constraints 7.5 and 7.1. However, the constraint regarding *placement deviation* needs to be adjusted, as will turn out in the upcoming explanation.

In Figure 7.2 it can be seen that the flip-flop which registers the contamination fault belonging to the current edge is placed at a later point in the chain than the register. This is to make sure the fault signal of the current stage can reach the input of the flip-flop in time before it is clocked. When placing the NOR-gate in the chain according to Constraint 7.6, the flip-flop has to be placed later than the register in the chain by at least the following amount:

$$\begin{aligned} \text{Delayed ff clocking} &> \text{placement deviation} + \text{delay } clk\text{-}Q \text{ ff} \\ &\quad + \text{delay meta-stability detection} + 2 \times \text{delay OR} + \text{wiring delay} + \text{delay setup ff} \end{aligned} \quad (7.8)$$

This is to make sure the latest comparison between the NOR-pulse and reset pulse is sampled by the upper right flip-flop in Figure 7.2. Constraint 7.8 is unfeasible however, because the following constraint has to apply as well:

$$\begin{aligned} \text{Delayed ff clocking} &< 2 \times \text{delay NAND} + \text{Added latency} + \text{delay reset ff} \\ &+ \text{delay meta-stability detection} + 2 \times \text{delay OR} + \text{wiring delay} - \text{delay setup ff} \end{aligned} \quad (7.9)$$

The latter constraint guarantees that the effects of the reset caused by the rising edge leaving the stage are not seen by the upper right flip-flop in Figure 7.2. If they are seen however, then the upper right flip-flop will never sample a contamination fault, because the reset caused by the edge leaving the current stage will reset the flip-flop connected to the NOR-gate, and this will always be visible for the upper right flip-flop. Unfortunately, Constraint 7.8 interferes with Constraint 7.9. The solution is to make *placement deviation* smaller, and this comes down to placing the first to the chain connected input of the NOR-gate earlier in the chain. The precise magnitude of *placement deviation* in order to satisfy constraints 7.8 and 7.9 is as follows:

$$\begin{aligned} \text{placement deviation} &= 2 \times \text{delay NAND} + \text{added latency} - \text{delay clk-Q ff} \\ &\quad - 2 \times \text{delay setup ff} \end{aligned} \quad (7.10)$$

The value for *placement deviation* is retrieved by subtracting $\text{delay clk-Q ff} + 2 \times \text{delay setup ff}$ from the original Constraint 7.7, and $\text{delay clk-Q ff} + 2 \times \text{delay setup ff}$ is the difference between both right sides of constraints 7.8 and 7.9.

7.4 Fault handling

The logic surrounding the pipeline which makes possible the fault handling is more complex than in Chapter 4. Next to a full re-execution of all the data in the pipeline, it must support a single re-execution of a single set of data. Therefore the RTL design in Figure 4.2 in Section 4.4 needs to be expanded to support a single re-execution.

Figure 7.3 depicts an RTL design of the fault treatment logic which can handle an out-of-order execution. The precise working and details are not explained, because the final RTL design depends on the specific system or processor into which it is integrated. The design of Figure 7.3 is just an example of how the fault treatment logic can be implemented, and it is actually a fairly simple design which is only used to test the possible performance improvement an out-of-order execution AICBP can give. Figure 7.4 depicts the *Pointer difference* unit from the RTL design in Figure 7.3, and Figure 7.5 shows the finite state machine belonging to the RTL design. The basics of the RTL design are only discussed however, so that still the final simulation results can be understood.

7.4.1 Basics of the fault treatment logic

The RTL design in Figure 7.3 is actually quite a simple design for out-of-order fault treatment logic although it looks very complicated. More advanced fault treatment logic becomes even more complicated and also depends on the specific out-of-order execution scheme for the specific processor into which AICBP is integrated. Therefore the most simple fault treatment logic for out-of-order execution is used in this thesis.

The RTL design is made in such a way that only one out-of-order execution is allowed at a time. When a set of data comes out of the pipeline which is contaminated, this set of data is re-executed by inserting the original data into the pipeline. It takes a while before the re-inserted data comes out of the pipeline. If another contaminated set of data during this period comes out of the pipeline it is not re-executed in an out-of-order fashion. Instead of an out-of-order execution a re-execution of all the data is done in such a case.

The reason for allowing only one out-of-order execution is that when re-executing a particular set of data it must be registered when the results belonging to this set of data come out of the pipeline. In the design of Figure 7.3, the circuitry connected to *Saved tail pointer* and *Counter out of order* is responsible for detecting and signalling when the out-of-order re-executed data is coming out of the pipeline. When allowing multiple sets of data to be executed out-of-order at the same time, more of such circuitry is needed, because for each out-of-order executed set of data such circuitry is needed. To make things less complicated and to take a fairly easy RTL design for the fault treatment logic, only one out-of-order execution is allowed at a time. Disadvantage is that not the best performance improvement is achieved, because allowing multiple out-of-order executions simultaneously gives a better performance improvement. Moreover, a full re-execution is done with the design of Figure 7.3 when a second contamination fault comes in. To make matters worse, the first set of data to be re-inserted with a full re-execution belongs to the set of data with the first contamination fault. Thus when a set of data is going to be re-executed out-of-order it might be re-executed again along with the original following sets of data.

To summarize the last explanation: A full re-execution of all the data from the first contamination fault until the most recent set of data is done in case of a second fault. It does not matter whether it is a fault belonging to a specific edge or a contamination/collision fault coming directly from one of the stages. The reason is that the bookkeeping of when a specific re-executed set of data comes out of the pipeline is reasonably simple. This option is chosen to keep the RTL design as simple as possible and to consider eventual performance improvements as a kind of baseline improvement.

7.4.2 Constraints of the fault treatment logic

Three constraints are noteworthy to mention, because without these the RTL design in Figure 7.3 will not function. These constraints concern the latencies which must be added in three places in the RTL design.

$$\text{Added latency tail clocking} > \text{delay MUX} + \text{delay setup tail register} - \text{delay OR} \quad (7.11)$$

$$\text{Added latency race prevention} > \text{delay NAND} + \text{delay setup ff} - \text{delay SR-latch} \quad (7.12)$$

$$\text{Added latency} > \text{delay AND} + \text{delay clk-Q ff} + \text{delay inverter} \quad (7.13)$$

7.5 Simulation results

The out-of-order contamination detection system has been tested and simulated in VHDL along with the RTL design of Figure 7.3. The simulation setup in VHDL is the same as described in Section 6.2. Consequently, the performance results of the simulation can be compared to the simulation results of an AICBP without out-of-order re-execution.

The performance results are depicted in figures 7.6 and 7.7 for both a 4 and 8 stage AICBP with out-of-order re-execution. In appendix Section A.4 the number of faults are shown as well. The simulation setup is the same as the one in Section 6.3.1, so the results in figures 7.6 and 7.7 can be compared to figures 6.4 and 6.5 to see what the performance improvement is.

The performance improvement is not very big, because for both a 4 and 8 stage AICBP the performance improvement is 2%. For the hold system position which gives spacing = 0.6 the performance improvement is sometimes 3%. This is certainly not much and there are actually two reasons for this:

- Single out-of-order execution. As is explained in Section 7.4.1, only one set of data is allowed to be re-executed out-of-order. When a second contamination or collision fault occurs during an out-of-order execution, a full re-execution is done. This re-execution starts with inserting again the out-of-order re-executed data followed by the original stream of input data. When this second fault occurs, the performance is actually worsened at this particular moment in comparison with the simple re-execution scheme described in Chapter 4. This second fault lets the final performance results be less than the potential performance results.
- Changed location of contamination detection system. Due to constraint 7.10 contamination faults happen more often, because the flip-flop which is responsible for detecting a contamination fault is moved forward in comparison with the original contamination detection system. Therefore slightly more faults happen as can be seen as well in the fault graphs in appendix Section A.4. It is important however to note that the fault graphs of sections A.1 and A.4 are not really comparable, because the sequence of execution is different. The fault graphs of the out-of-order execution scheme do give a clear indication though that the fault percentage has increased.

Summarizing, the performance improvement is not spectacular, but the out-of-order scheme is capable of improving performance while the number of faults (Section A.4) has increased.

7.6 Conclusions regarding out-of-order execution

The performance improvement is not very big in the simulations, but in the simulations only one set of data can be executed out of order. If this is changed the performance improvement can be a lot better. An indication can be retrieved by looking at Equation 6.11 which gives a raw performance estimation. Equation 6.11 is repeated here:

$$\begin{aligned}
& \text{average output period (\%)} = \text{input clock period (\%)} \\
& + \text{re-execution duration} \times \text{input clock period (\%)} \times \frac{\text{faults (\%)}}{100\%}
\end{aligned} \tag{7.14}$$

The gain of the out-of-order scheme is to let *re-execution duration* approach 1 whereas it is normally more or less *pipeline length* + 1 at a clock period equalling the average inverter chain delay. Of course *re-execution duration* can never become exactly 1, because still collision faults can occur and a full re-execution is still issued in case of a third rising edge at the contamination detection system.

Logic surrounding the pipeline which can handle multiple out-of-order executions is more complicated than the logic in the RTL design of Figure 7.3. The precise implementation and overhead depend on the already existing out-of-order execution logic of the processor into which AICBP is integrated. Due to the larger complexity, and also having no processor to integrate it within, no implementation for multiple out-of-order executions is made for this thesis. It may be interesting for future work to make such an implementation and perhaps integrate it into a processor with out-of-order execution capabilities. The potential speedup is quite good and it is therefore likely worth the effort.

Figure 7.3: The RTL design of the out-of-order fault treatment logic.

Figure 7.4: *Pointer difference* unit from the RTL design in Figure 7.3.

Outputs state machine:

Standard outputs:		Normal operation:		Normal operation, no FIFO save	
reset_1st_stage	= '0'	no different outputs		fifo_update	= '0'
reset_2nd_stage	= '0'				
reset_other_stages	= '0'				
enable_fault_info	= '1'	Recovery first cycle:		Out of order recovery first cycle:	
is_recovery	= '0'	reset_1st_stage	= '1'	reset_cntr_comparator	= '1'
fifo_update	= '1'	reset_2nd_stage	= '1'	is_out_of_order_recovery	= '1'
reset_cntr_comparator	= '0'	reset_other_stages	= '1'	is_recovery	= '1'
is_out_of_order_recovery	= '0'	enable_fault_info	= '0'		
reset_first_fault	= '0'	is_recovery	= '1'	Out of order recovery second cycle:	
		fifo_update	= '0'	is_out_of_order_recovery	= '1'
				fifo_update	= '0'
Reset state:		Recovery second cycle:		Out of order recovery:	
reset_1st_stage	= '1'	reset_1st_stage	= '1'	is_out_of_order_recovery	= '1'
reset_2nd_stage	= '1'	reset_other_stages	= '1'		
reset_other_stages	= '1'	enable_fault_info	= '0'	Normal operation, reset 1st fault	
fifo_update	= '0'	is_recovery	= '1'	reset_first_fault	= '1'
reset_cntr_comparator	= '1'	fifo_update	= '0'		
		reset_first_fault	= '1'		
First cycle:		Recovery:			
reset_1st_stage	= '1'	is_recovery	= '1'		
reset_2nd_stage	= '1'	fifo_update	= '0'		
reset_other_stages	= '1'				
reset_cntr_comparator	= '1'				
reset_first_fault	= '1'				

Figure 7.5: The finite state machine of the RTL design in Figure 7.3

Figure 7.6: Performance of a 4 stage AICBP with out-of-order re-execution

Figure 7.7: Performance of an 8 stage AICBP with out-of-order re-execution

Data dependent delay compensation

8

The AICBP design discussed thus far is actually a design which compensates for critical path variations and not for data dependent delay variations. This chapter introduces an extension for AICBP which can also compensate for data dependent delay variations. The gain is a better throughput, but the price is that extra area and power is used.

8.1 Basic idea

Quite often the logic delay for a set of data is smaller than the critical path delay in a particular stage. Consequently, the computed data must wait for a short amount of time before they are clocked, because the clocking moment is determined by the critical path. Therefore time is wasted by not clocking or letting through the already computed data, and this in turn reduces the throughput. It is quite difficult however to adjust the clocking moment to the data delay, because one has to account for all kinds of data patterns and its belonging delay. Instead of adjusting the clocking moment it is chosen to introduce a transparency period for flip-flops. Actually now the flip-flops cannot be called flip-flops anymore, because they are actually pulsed latches now. The purpose of the transparency period is to pass already computed data on to the next stage earlier. The transparency window starts earlier than the original clocking moment of the register, and consequently data which are already computed can enter the next stage earlier.

One may wonder what the purpose is of data entering the next stage earlier, because the rising edge in the inverter chain still determines when to make the latch transparent. This rising edge is still propagating through the chain with the speed of the critical path, and hence the throughput will not really improve by just making a latch transparent for a while. For example, if data enter the next stage earlier, the clocking or transparency period of the next register in the next stage will occur at the same time instance as before, so there is actually no gain. Without changing anything extra the throughput will not improve, because it is still determined by the propagation of the rising edge in all the inverter chains.

8.2 Basic components

A key thing for improving throughput is to let the clocking, or transparency period, occur earlier in the next stage when the data delay in the current stage is short. When the computed data are available quite early and are passed on to the next stage early because of the transparency window, it is useful to let the register at the end of the next stage clock earlier. In such a way throughput is really improved. To let the next register clock earlier, circuitry can be used which can actually speed up the propagation of the rising edge in the inverter chain. Such circuitry is discussed in the following section.

8.2.1 Time giving and time taking inverter chain

Figure 8.1 illustrates a standard inverter chain augmented with some extra circuitry. The extra circuitry can make the output of every even inverter high when $\overline{\text{give}}$ is made low. The nice thing is that if a rising edge is going through the inverter chain depicted in Figure 8.1 it can be moved to the end of the chain by just making $\overline{\text{give}}$ low. As a result, the propagation is speeded up at once with the length of the remaining inverter chain connected to the augmented circuitry. With the remaining inverter chain it is meant the piece of inverter chain in Figure 8.1 where the output of the even inverters is low and where the rising edge has not come by yet.

Figure 8.1: Time giving chain.

Figure 8.2: Time taking chain.

The circuitry in Figure 8.1 is speeding up the rising edge and is thus actually giving time. Therefore the circuitry in figure 8.1 is referred to as a time giving chain. A circuit which does the opposite of the time giving chain can easily be made by placing the augmented circuitry at the output of the odd inverters instead of the even inverters. Such a scheme is depicted in Figure 8.2. By doing so the rising edge is actually delayed,

because when $\overline{\text{take}}$ is made low the rising edge is placed at the beginning of the inverter chain if the rising edge is in the chain of Figure 8.2. It may come to no surprise that the circuitry of Figure 8.2 is referred to as a time taking chain, since it does the opposite of a time giving chain. A time taking chain may not seem relevant at this moment, but it is used in Section 8.3 and is therefore mentioned here.

8.2.2 Input transition detection flip-flop

A remaining question is still how to determine when the data delay is shorter than the critical path delay. It must be known after all, since it is necessary to determine whether the propagation of the rising edge can be speeded up. In order to make a general approach and without having to have knowledge of the used combinatorial logic, it is decided to check this at the input of the register. Checking whether the data delay is smaller than the critical path delay can be simply done by checking if one of the inputs of the register changes during the transparency period. If none of the inputs change during this period the data delay is short and the data have arrived at the input of the register before the transparency period started. On the other hand, if data arrive during the transparency period, then at least one of the inputs of the register will change. The specific moment of the arrival of the data can be determined by checking when the last input in time changes its value. For example, when one input still changes near the end of the transparency period, it can be said that the data delay almost equals the delay of the critical path.

To determine the data delay magnitude it is thus necessary to have a register which can signal when one of the inputs is changing during the transparency window. To realize such a register, extra circuitry needs to be added to every flip-flop in the register. This circuitry must detect if the input of a flip-flop is changing during the transparency period. If so, the circuitry must signal such a change, so that the propagation speed of the rising edge can be adjusted. More on how to do this is explained in the next Section 8.3, but first the added circuitry to the flip-flop is explained.

The specific circuitry which must be added to the flip-flop may also depend on the specific flip-flop implementation. However, the circuitry presented here can be added to almost every flip-flop, and the structure of it is depicted in Figure 8.3. In this figure an ep-SFF flip-flop [20], or actually a latch, can be seen with an inverter, XNOR, and NAND augmented to it. These last three gates can generate a pulse when the input changes during the transparency period. These three gates can be used regardless the implementation of the flip-flop, but they do need to have a pulse which marks the transparency period. The ep-SFF flip-flop is shown in Figure 8.3, because it might be a good candidate for the flip-flop implementation, because it is a fast and small implementation ([13]). Moreover, the pulse generating mechanism can be used for multiple flip-flops, and the three inverters within the pulse mechanism can be taken from the inverter chain. However, most likely multiple pulse generating mechanisms are needed when the register contains a lot of flip-flops, so the advantage is smaller than it seems.

The inverter, XNOR, and NAND in Figure 8.3 can be implemented by the circuitry in Figure 8.4. The circuitry is based on the transistor level XOR/XNOR implementations given in [22], and it generates a small negative pulse when the input of the flip-flop

Figure 8.3: An ep-SFF from [20] with circuitry with signals input changes.

changes. The presented circuitry in Figure 8.4 is merely a suggestion on how to make a circuit which can detect input changes, because it has not been simulated and tested into detail. The circuitry might for example not be able to detect very small glitches, and this in turn may be problematic because these glitches perhaps can cause some switching in the next stage. A more detailed investigation on this kind of circuitry and also on the specific flip-flop circuitry is actually needed, but this is out of the scope of this thesis project.

Figure 8.4: Circuitry which can generate a pulse on an input change.

Every *Late Arrival pulse* of each flip-flop can be connected to a NAND-gate as in Figure 8.5. In such a way a *Late Arrival pulse* for the whole register is achieved, and this signal is used in the data delay compensation system. Instead of using a single NAND-gate, multiple levels of gates can be used to generate the *Late Arrival pulse*.

8.3 Implementation of data dependent delay compensation

The components discussed in the last section are vital for the implementation of the system which makes data dependent delay compensation possible. The design of this system is shown in Figure 8.6 where the time taking and time giving chain are used. Input transition detection flip-flops are used as well, but these are not explicitly shown

Figure 8.5: Combining the $\overline{\text{Late Arrival}}$ pulse of every flip-flop into one signal.

since they are within the register. Actually flip-flops are not used, because pulsed latches are used instead. Nevertheless, next to the pulsed latches used within the register also input transition detection circuits are included such that a *Late Arrival pulse* can be generated. The design shown in Figure 8.6 is situated at the register between two stages and applies for all the registers in the pipeline except for the first and last one.

The first important thing to note about the design of Figure 8.6 is about the length of the transparency period of the latches within the register. The length of the transparency period, or transparency window called in a different way, can be dynamically adjusted. This is very important since it prevents contaminations. As can be seen in the design, the transparency window starts when the rising edge comes in from the left. At least, that is the case when both *Toggle system output* and *Output Q SR-latch hold system* are high. *Toggle system output* and *Output Q SR-latch hold system* are there to detect if the previous rising edge is already at the next register. It is chosen only to make the transparency window larger when the previous rising edge is already at the next register or beyond. When the previous rising edge is not at the next register, the latches will only be transparent during a three inverter delay which is comparable to other soft edge flip-flops ([13]). By making the transparency window smaller if the previous rising edge is not at the next register, contaminations are prevented. Imagine that the transparency window should open early when the previous rising edge, and thus also the previous data, is still in front of the next register. By opening up early, signals are allowed to enter the next stage earlier, and as a consequence these signals may contaminate the previous data earlier as well. As a result, more contamination faults will occur and this has a negative impact on performance.

The *Toggle system output* and *Output Q SR-latch hold system* signals can signal if the previous rising edge is at the next register. The implementation for the circuitry which generates the *Toggle system output* signal is shown in Figure 8.7. The circuitry contains a number of toggle flip-flops which toggle their output value when a rising edge comes by. If all the toggle flip-flops have the same output value, the rising edge must be at the next register. Three toggle flip-flops are used to still let the circuitry work when the chain before the next register contains two rising edges. The last toggle flip-flop is placed after the hold circuit, so it is assured that the previous rising edge is not hold at its position at the next register when the transparency window of the current register is

Figure 8.6: Implementation for data dependent delay compensation.

made larger. Making the transparency window larger when the previous edge or data is hold is not a wise thing, because this may also lead to more contaminations.

The function of the time taking and time giving chain is to speed up the propagation of the rising edge when the transparency window is bigger than three inverters. The working of these two chains is as follows: If a rising edge comes in from the left in Figure 8.6 and both *Toggle system output* and *Output Q SR-latch hold system* are high, the *take* signal is high, so that the rising edge which goes into this chain is not delayed. Without delaying the rising edge in the time taking chain, the NAND-gate connected to the time giving chain receives two rising edges simultaneously. Consequently, the rising edge in the inverter chain is moved from the beginning of the time giving chain to the end of the time giving chain. This moving of the rising edge from the beginning to the end of the time giving chain can only be done when the rising edge in the time

Figure 8.7: The toggle system.

taking chain is not delayed or set back. When it is delayed due to for example a smaller transparency window, the rising edge in the time giving chain is not moved forward that much. In this case the rising edge is already somewhere in the time giving chain before it is moved forward. The key thing to note here is that the speeding up of the rising edge is proportional to the length of the transparency window.

When the data arrive later than the start of the transparency window, the rising edge must not be speeded up that much. When data arrive later it will also enter the next stage at a later point, and therefore the clocking of the next register must occur later as well. To let this clocking occur later, the rising edge in the time giving chain must not be moved forward that much as without the later arrival of the data. In order to achieve this kind of behaviour the input transition detection circuit of Section 8.2.1 is used. By using this circuit for every flip-flop and combining the outputs of all the circuits just as in Figure 8.5 a *Late Arrival pulse* is retrieved. This *Late Arrival pulse* can let the time taking chain delay or place back the rising edge which is within. Summarizing, when data arrive later than the start of the transparency window, a *Late Arrival pulse* is generated which in turn lets the rising edge in the time taking chain to be placed back. As a result, the time giving chain will not place the rising edge forward that much.

Summarizing, the design in Figure 8.6 just speeds up the data and rising edge when the previous data and rising edge are not very close. This speeding up is only possible when the data delay is smaller than the critical path delay. By speeding up data and rising edge the distance with the following rising edge becomes larger. This is quite advantageous, because such a larger distance reduces the chance of a contamination in the upcoming stages. Reducing this chance of contamination leads to less contaminations and thus also a performance improvement.

It is important that the system of Figure 8.6 knows as soon as possible if the previous rising edge is beyond the hold circuit at the next register. In such a way the system can let the transparency window begin as soon as possible. The information of the previous

rising edge being at the next register is given by *Toggle system output*, and therefore the distance between the last toggle flip-flop in the next stage and the system at the current register must be as short as possible. If this distance would be quite large, it takes a while before *Toggle system output* becomes high when a rising edge comes by, and as a result the transparency window starts much later than possible. The consequence is that rising edges are not moved forward as much as is actually possible. To prevent this it may be a wise idea to lay out the beginning and ending register of a stage close to each other. As a result, the distance between the last toggle flip-flop of a stage and the data dependent delay compensation system at the beginning of a stage is quite small.

8.4 Constraints for the data dependent delay compensation system

The design of Figure 8.6 also has a number of constraints which must be valid in order to get a working system. The first constraint defines how much the distance between the start of the time taking chain and the first to the chain connected NAND-gate is.

$$\begin{aligned} \text{Start Time taking chain} &> \text{delay NAND} + \text{delay SR-latch} + 2 \times \text{delay NAND} \\ &+ \text{delay generate LA pulse} + \text{delay NOR} + \text{Late Arrival pulse width} \end{aligned} \quad (8.1)$$

The constraint makes sure that once a rising edge enters the time taking chain it can immediately be hold or placed back by a *Late Arrival pulse* generated by the register. Making *Start Time taking chain* less than the right term in Equation 8.1 would cause a problem, because a *Late Arrival pulse* occurring at the end of the transparency window can not set back the rising edge in the time taking chain. The rising edge has simply already left the time taking chain then and can therefore not be set back.

The length of the Time taking chain is as follows:

$$\begin{aligned} \text{Length Time taking chain} &> \text{Length largest transparency window} \\ &- \text{Late Arrival pulse width} \end{aligned} \quad (8.2)$$

Late Arrival pulse width is subtracted, because it is added to the right term in Equation 8.1. With *Length largest transparency window* it is meant the length of the transparency window if the previous rising edge is beyond the next register. Its size is determined in the following way:

$$\begin{aligned} \text{Length largest transparency window} &= \text{NAND to NAND Chain length} \\ &+ \text{delay NAND} - (\text{delay NAND} + \text{delay SR-latch} + \text{delay NAND}) \end{aligned} \quad (8.3)$$

The constraint assumes that the effects of the reset of the SR-latch are seen at the lower input of the NAND-gate connected to the latch before the upper input becomes high. This can be guaranteed with the following constraint:

$$\begin{aligned} \text{Distance NOR-NAND} &> \text{delay NOR} + \text{delay NOR} + \text{delay SR-latch} \\ &+ \text{delay NAND (3 input NAND)} - \text{delay NAND (2 input NAND)} \end{aligned} \quad (8.4)$$

When looking at the input of the NOR-gate connected to the $\overline{take}$ input of the time taking chain, it can be seen that some latency has to be added to the lower input signal. The delay is necessary to make sure that the rising edge is hold for a short time when the transparency window is slightly smaller than the largest transparency window possible. As a result, the rising edge will not be placed forward that much in the time giving chain as with the largest transparency window. This is exactly the kind of behaviour which is vital, because else the rising edge would be placed forward too much. To ascertain this reduced forward placing in case of a smaller transparency window, the following constraint has to apply:

$$\begin{aligned} \text{Added latency} &> 2 \times \text{delay NAND} + \text{delay generate Late Arrival pulse} \\ &+ \text{Late Arrival pulse width} - \text{delay NAND} \end{aligned} \quad (8.5)$$

A constraint regarding the time giving chain is the following one:

$$\text{Distance NAND-Time giving chain} < \text{delay NAND} \quad (8.6)$$

This is to prevent the time giving chain from placing the rising edge at the end of its chain while the rising edge is not even in the time giving chain yet. The consequence of such an action is a negative glitch in the inverter chain, and to prevent this Equation 8.6 must apply.

Equation 8.6 does not specify where the time giving chain should start. The following constraint specifies this:

$$\text{Begin taking chain to NAND} < \text{Length Time taking chain} \quad (8.7)$$

Without obeying this constraint the rising edge can be placed forward too much.

The length of the time taking chain is clarified in Equation 8.2, but the length of the time giving chain is not clarified yet. One should say that the length of the time giving chain should equal the length of the largest transparency window or the time taking chain. This cannot be the case however, because problems occur when the combinatorial logic in the stage after the register is quite slow. Imagine that the data arrive at the beginning of the largest transparency window possible. The data will immediately enter the combinatorial logic after the register. It may happen that the delay of the logic and the time giving chain is slower than the delay of the time taking chain. However, because the data arrived quite early, the rising edge is placed forward with the length of the time giving chain. The amount of placing forward is too much, because it is based on the speed of the time taking chain which is faster than the time giving chain. It can actually be said that the rising edge is in such a case in front of the data, and this is problematic because it will let the next register clock too early.

To account for the situation of a faster time taking chain than a time giving chain the following constraint has to apply:

$$\begin{aligned} \text{Length time giving chain} &< \frac{\text{smallest delay time taking chain} + \text{width LA pulse}}{\text{largest delay time giving chain}} \\ &\times \text{Length largest transparency window} \end{aligned} \quad (8.8)$$

The just described situation is prevented by making the length of the time giving chain smaller than the time taking chain. It is made smaller with the fraction $\frac{\text{smallest delay time taking chain} + \text{width LA pulse}}{\text{largest delay time giving chain}}$, because it accounts for the worst-case scenario where the time taking chain is very fast and the time giving chain is very slow. Disadvantage of this constraint is that the rising edge cannot be placed forward that much as is often possible.

The final constraint defines where the first toggle flip-flop of the toggle system in Figure 8.7 must be placed. It must be placed before the release point of the hold system such that not both the *Toggle system output* and *Output Q SR-latch hold system* signals can become high for a short period when a rising edge travels by. The amount of placing before the release point of the hold system is as follows:

$$\begin{aligned} \text{Distance toggle to release point} &> \text{delay toggle ff} + \text{delay OR} + \text{delay NAND} \\ &\quad - (\text{delay NAND} + \text{delay AND} + \text{delay SR-latch}) \end{aligned} \quad (8.9)$$

8.5 Simulation

A simulation in VHDL has been done of an AICBP with the system of Figure 8.6 at every register in the pipeline. The first and last register in the pipeline do not have the system of Figure 8.6 though. The simulation parameters are the same as is said in Section 6.2, but there are some extra parameters which need some explanation.

8.5.1 Input data

The precise data which are inserted in the pipeline influence performance, since the data will determine the computational delay of the logic in each stage. To be more specific, the computational delay of the current set of data is also determined by the previous set of data which was inserted into the stage. If the previous set of data is almost the same as the current set of data, the computational delay will likely be quite small in comparison to the delay of the critical path. Of course these things depend on the specific combinatorial logic and data, but the point is that if subsequent sets of data are correlated, the computational delay will most of the time be quite small relative to the critical path delay.

It is thus very important what the correlation is of subsequent sets of input data, since this can influence performance. Correlated sets of data will improve performance, due to the smaller computational delays in each stage which in turn the data dependent delay compensation system can exploit. Therefore two kinds of simulations are done: One with correlated input data, and one with uncorrelated input data. The correlated input data consists of sets of data which are all the same, meaning that in every cycle the same set of data is inserted. The uncorrelated data is composed by letting each set of input data to be generated by the pseudo-random generator in VHDL.

8.5.2 Transparency window length

The transparency window length, or *length largest transparency window* in Equation 8.3, is also adjustable by just varying *NAND to NAND chain length* in Figure 8.6. It is interesting to see what kind of performance improvements are achieved with different transparency window lengths. Therefore simulations have been done with a transparency window length of 20%, 10%, and 0%. With for example 20% it is meant that the *length largest transparency window* from Equation 8.3 is 20% of the total chain length in one stage. The *length largest transparency window* is thus also varying in time just as the total chain length does. This makes sense, because in reality *length largest transparency window* will also vary. The simulation has been set up such that the *length largest transparency window* is always 20%, 10%, or 0% of the chain length regardless the actual length of the inverter chain in the simulation. A transparency window length of 0% means that there is no transparency period, and this situation is comparable to the already existing basic AICBP as described in Chapters 2, 3, and 4.

8.5.3 Contamination detection system

In the simulations of Chapter 6 only one contamination detection system per stage is used. As a consequence, some paths through the logic are not very correlated with the inverter chain onto which the contamination detection system is connected. The solution is to add some delay to these paths in order to not let any undetected contamination occur. This is explained in more detail in Section 6.2.3.

Adding extra delay to paths in the logic is in this case not desirable, since it lengthens the computational delay of the data in a stage. This lengthening in its turn can reduce the speedup which can be achieved by using a data dependent delay compensation system. Therefore in this simulation two contamination detection systems are used. Using two of these systems can be done in a way as described in Section 3.2.5. The inverter chain which serves as an input for the second contamination detection system is located at the end of the stage, so it has a 100% correlation with the delay of the gates at the end of the pipeline stage.

8.5.4 Simulation setup

The data dependent delay compensation system improves the performance the most when all the rising edges are not situated close to each other. More specific, it can only improve performance if two rising edges are not in the same stage, because only then it can accelerate the propagation of a rising edge. Summarizing, performance cannot be improved if almost all the time each stage contains two rising edges, because then no rising edge will be accelerated. In view of that, applying the data dependent delay compensation system to AICBP systems which have a clock period lower than the average stage delay makes no sense. A simulation with a contamination/critical path ratio of $\frac{1}{3}$ is therefore done, because in this simulation the clock periods are most of the time longer than the average stage delay.

The results for a simulation with this setup and with a transparency window length of 0%, 10%, and 20% can be found in appendix Section A.5. Simulations with both

correlated and uncorrelated input data are done.

8.6 Evaluation of the simulation results

An important thing to note regarding the simulation with 0% transparency window is that the performance results are not comparable to the performance results of the simulations done earlier in Section 6.3.3 and appendix Section A.3. One should expect them to be the same, but they are not since two contamination detection systems are used with this simulation. Also the hold circuit position is slightly changed due to the application of the data dependent delay compensation system. This in turn causes the simulation results to be slightly different as well.

The performance improvement for a simulation with a 10% transparency window and uncorrelated input data is not that very spectacular. The maximum performance improvement for respectively a 4 and 8 stage AICBP is 2.5% and 4.5%. The performance improvement for the same simulation but with correlated input data is a bit better, and it is 3.5% and 8%. The latter results make sense, because due to the correlated input data the computational delay is more often small, and this in turn gives the data dependent compensation system the possibility to speed up the rising edge more often.

The most striking thing about the simulation results is that the performance for the simulation with a 20% transparency window is not improved. In fact, the performance is actually a bit reduced. The reason for this is that the rising edges are actually placed forward too much with a transparency window of 20%. With this transparency window it occurs that a rising edge is placed forward, but it later on leads to a contamination fault because it gets too close to the rising edge in front. Of course, the rising edge is not placed directly too close to the in front rising edge due to the working of the toggle system, but later on in the pipeline it may get closer to the rising edge in front. Especially with the hold system position which leads to a spacing of 0.7 this is the case.

8.7 Conclusions

The data dependent delay compensation system is able to improve performance of an AICBP but unfortunately not in a spectacular way. The magnitude of the performance improvement also depends on the maximum length of the transparency window. Making this length too big does not lead to the best performance improvement. On the other hand, making the length too small makes the performance comparable to a basic AICBP without the data dependent delay compensation system.

The overhead of the data dependent delay compensation system is quite big in both area and power. The biggest portion of the overhead is in the input transition detection flip-flop of Section 8.2.2. The circuitry needed for input transition detection has to be added to each flip-flop. Especially if the bit width of the register is quite large, the overhead will be quite considerable in terms of area. Also the power overhead is not small due to the appliance of the input transition detection circuit, because the circuit switches on every input change and this applies for every flip-flop. The overhead of the design in Figure 8.6 and the toggle system (Figure 8.7) is comparable to other systems

of the basic AICBP design, and this means that the overhead is not that big.

Despite the minor performance improvement and large overhead the data dependent delay compensation system may still be interesting for future research. Perhaps the performance can be improved, but more important perhaps a cheaper implementation for the input transition detection circuit in terms of area and power can be made. The latter will really reduce the overhead and make the data dependent delay compensation system worth using.

Design steps

This chapter introduces a general design methodology which can be used for making an AICBP. It shortly describes the steps designers can follow to make an AICBP. With an AICBP it is meant the basic AICBP without the extensions from chapters 7 and 8. The design steps are described below and give a short description on what the designer should do step by step. These steps are most of the time just a summary of the things that are said in earlier chapters.

1. Make the shortest path through the logic for each stage as big as possible such that the smallest contamination delay possible for each stage is as large as possible. By doing so actually a trade off is made between performance and on the other hand area and power consumption. It can be chosen to not make the contamination delay as large as possible yet and to get back to this step later on.
2. Place next to the critical path the inverter chain. The inverter chain for each stage should have the same length as the critical path in each stage plus some safety margin as described in Section 2.3. The length of the inverter chain is allowed not to be the same for every stage, since that will not jeopardize the correct working of the pipeline. For the performance it is most likely the best thing to keep the same length for all the inverter chains.
3. Place the high phase system at a position in the inverter chain such that the resulting high phase will not be too small (Section 3.4.3.2). On the other hand the resulting high phase must not be too large, because else performance will be reduced. The position of the high phase system can be the same in each stage if all the inverter chains have equal length.
4. Place the contamination detection system for every stage at the appropriate point in the chain (Section 3.2.2). If the inverter chain in front of the contamination detection system is probably not laid out on the die close to the shortest path through the logic, two approaches can be taken. One is to use a second contamination detection system and the other is to place the contamination detection system at a later point in the chain (both described in Section 3.2.5).
5. Connect the hold system to a position in the chain which will give the most advantageous spacing between rising edges for the average die. (*Release point* in Section 3.3) It is rather difficult to establish the most advantageous spacing and therefore this step will need to be redone most likely. However, by carefully studying the combinatorial logic in the pipeline stages as well as the fabrication technology specific variations something can be said about where the hold system should be placed more or less.

6. Determine the most ideal clock frequency for the average die by doing some simulations. Preferably these simulations should be done at a lower level than the VHDL simulations of Chapter 6. Simulations at the transistor level should give quite a realistic result.
7. Evaluate the average output clock frequency of the simulations in the last step. By going back to step 5 and redoing the rest of the subsequent steps again, a performance comparison at this point can be made between different hold system positions. The best hold system position for the average die can be chosen by just comparing the average output clock frequency of different hold system positions.
8. Evaluate the performance for the best hold system position. If the performance is not as desired, the smallest contamination delay can be made larger. As a consequence step 1 needs to be redone followed by the rest of the consecutive steps.

The above steps are optimized to get the best performance for a die with average delay characteristics. It may very well be that dies with less advantageous delay characteristics will have an average output frequency which will be far worse than the one of an average die. When looking at the performance figures of appendix chapter A, one can see why this is a situation which might occur. One solution is to take a more conservative input clock frequency. Another solution is to place the hold system a bit more up-front, because then the average output frequency does not reduce that much when delay characteristics are less favourable.

Recommendations and conclusions

10

The final remarks regarding this thesis are made in this chapter. First of all, there are a number of improvements which can be applied to the AICBP and which do not have been discussed thus far, because these improvements have not been developed yet. These possible improvements are discussed in the upcoming section on future recommendations. After this section conclusions are drawn for this thesis.

10.1 Recommendations for future work

The most obvious work to do in the future is to do a low level simulation of the AICBP. It is interesting to see if the performance results of a low level simulation match the performance results of Chapter A. Also implementing the AICBP on a real die would be very interesting, because then some real performance figures can be retrieved.

Besides a better simulation there are some improvements though which can be made to AICBP itself as is explained now. The main goal of the AICBP is to have an average output frequency which is as high as possible. The factors which influence this are the input clock period and the hold system position as discussed in sections 5.2 and 6.4.6. Also the contamination/critical path ratio and the specific delay variance are very important for determining the average output frequency, but these are factors which the designer can not easily adjust most of the time. The input clock period and the hold system position on the other hand are relatively easy influenced by the designer. The designer must choose the right magnitude for both of these factors in order to get the best performance. The problem is that the magnitude for achieving the best performance may differ per die and also in time. The reason is that logic delay characteristics may differ per die and in time. It is therefore a good idea to investigate if the input clock period and the hold system position can be dynamically adjusted in order to get the best performance improvement for each die.

A simple solution for determining the most ideal input clock period could be to count the number of valid outputs over a large amount of clock cycles. With that the average output period on the chip itself can be calculated, and once this is determined the input period can be varied to see if the average output period increases or decreases. Another possibility to determine the most ideal input clock period is to count the number of faults and to adjust the input clock period depending on this number. It is not easy though to determine the ideal input clock period only with the number of faults. The specific number of faults applies only to the current input clock period and not to another input

clock period. Therefore it is not that easy to determine with the help of Equation 6.11 the ideal input clock period, because the number of faults, or the fault percentage, may be totally different for another input clock period.

Varying the hold system position in order to get the best performance possible can easily be done by connecting the hold system (*Release point* in Figure 3.12) to the chain at multiple places. By selecting one of the connections with a multiplexer, the hold system can be connected to the most ideal position in the chain. The big question though is what the ideal position should be and how to determine this dynamically in a rather simple way. It is a very interesting question and it may be subjected to future research.

Another big performance improvement may come from improving the contamination detection system. Currently, in order to detect a contamination fault the system uses the worst case contamination delay, or in other words, the shortest path through the logic. However, this is not always necessary though, because depending on the data the contamination delay may be larger than the shortest path. As a consequence, some sets of data are marked as contaminated while this is not the case, because the upcoming set of data has a contamination delay longer than the shortest path. To prevent this false marking of a contamination, a contamination detection system should be designed which takes into account the data which is traveling through the stage. By doing so, the real contamination delay can be determined. It is not easy though to design such a system which is not too expensive and too complicated.

The last thing to note regarding future work is about developing an AICBP design for pipelines which have loops. More specific, it is meant that some pipelines have outputs of a particular stage register which serve as an input for the logic in a previous stage. In other words, it means that a particular pipeline stage needs to have output data of a stage further on in the pipeline. The difficulty with this kind of pipeline integrated within AICBP is that the data needed from a stage further on may not be present at all times due to the asynchronous clocking. A solution could be to use a hold system to guarantee that the data from the stage further on are there and are used for computation when the current stage clocks its data. The biggest difficulty is however in the re-execution caused by a fault. During a re-execution the first set of data inserted into pipeline needs to have at some point in the pipeline data from further on in the pipeline. The problem is that with the original re-execution scheme these data are not there, because the inserted set of data is the first one in the pipeline. Bottom line is that when doing a re-execution the whole pipeline actually needs to be restored to a certain state. This makes the re-execution more complicated. An AICBP design which is suited for these kinds of pipelines has not been designed thus far, since it is more complicated and it is questionable how often these kinds of pipelines are used.

10.2 Conclusions

A design for variability has been introduced in this thesis which can observe and compensate for delay variations. This design, called Adaptive Inverter Chain Based Pipeline (AICBP), is able to improve performance, or throughput, considerably. It is able to cope with delay variations, because the average output clock period can be made equal to the

average critical path delay of a pipeline stage. AICBP is able to improve performance even further and when doing so the AICBP design has actually some similarities with wave pipelining [8].

The cost of the basic AICBP design is relatively low and the most expensive thing in terms of power and most likely area is probably the inverter chain. The overhead of the contamination detection system, hold system, and phase system is rather small in comparison with the already existing registers and combinatorial logic. The main reason for this low cost is that the basic AICBP design only operates on the clock signal and not on the data.

The key thing which affects the performance of the AICBP the most is the contamination delay in each logic stage of the pipeline. More specific, the shortest contamination delay possible, or else said the shortest path through the logic, mainly determines the performance improvement possible. Making the shortest path through the logic large enough in order to get a good performance improvement may be quite expensive though. Therefore it strongly depends on the already existing logic whether the overhead of the AICBP design is small.

There are a number of issues though which may cause some implementation problems depending on the specific fabrication technology and low-level implementation used. The meta-stability detection circuit (Section 3.2.4) and the glitch removal circuit (Section 4.4.4) are examples of circuits which may give rise to some problems. Also the compliance to all the constraints belonging to the contamination detection system, hold system, and phase system may cause some problems. It may for example be the case in some fabrication technology that some gates have an extreme delay variance which in its turn causes that not all constraints can be satisfied.

The extensions to the basic AICBP design presented in chapters 7 and 8 can improve performance. The performance improvement achieved with the out-of-order re-execution scheme is not spectacular, but if multiple out-of-order re-executions are allowed, performance will most likely be improved in a more spectacular way. Nevertheless, the overhead is quite considerable and the performance improvement also depends on the processor into which it is integrated. The performance improvement of the data dependent delay compensation scheme is also not spectacular whereas the overhead is quite big. Therefore the basic AICBP design is still to be preferred, because its overhead is relatively low and the performance improvement without the extensions is already quite good. However, the extensions may be of interest in the future for certain specific applications, or perhaps these extensions can be improved such that the performance improvement is more spectacular.

Bibliography

- [1] <http://www.cs.umbc.edu/~squire/download/>.
- [2] Shahid Ali and Shivraj G. Dharne, *Glitch removal circuit*, US patent no. 6894540 (2003).
- [3] D. Andrade, F. Martorell, A. Calomarde, F. Moll, and A. Rubio, *A new compensation mechanism for environmental parameter fluctuations in CMOS digital ICs*, Microelectronics Journal **40** (2009), no. 6, 952–957.
- [4] Shekhar Borkar, Tanay Karnik, Siva Narendra, Jim Tschanz, Ali Keshavarzi, and Vivek De, *Parameter variations and impact on circuits and microarchitecture*, Annual ACM IEEE Design Automation Conference (2003), 338 – 342.
- [5] Keith A. Bowman, Steven G. Duvall, and James D. Meindl, *Impact of Die-to-Die and Within-Die Parameter Fluctuations on the Maximum Clock Frequency Distribution for Gigascale Integration*, IEEE Journal of solid-state circuits **37** (2002), no. 2, 183–190.
- [6] Keith A. Bowman, James W. Tschanz, Nam Sung Kim, Janice C. Lee, Chris B. Wilkerson, Shih-Lien L. Lu, Tanay Karnik, and Vivek K. De, *Energy-efficient and metastability-immune timing-error detection and recovery circuits for dynamic variation tolerance*, IEEE International Conference on Integrated Circuit Design and Technology and Tutorial, 2008 (2008), 155–158.
- [7] G. E. P. Box and Mervin E. Muller, *A note on the generation of random normal deviates*, The Annals of Mathematical Statistics **29** (1958), no. 2, 610–611.
- [8] Wayne P. Burleson et al., *Wave-pipelining: A tutorial and research survey*, IEEE transactions on very large scale integration (VLSI) systems **6** (1998), no. 3, 464–474.
- [9] Sorin Cotofana and Cor Meenderinck, *Casta DIVA - a design for variability platform*, International Semiconductor Conference, 2008. CAS 2008. **2** (2008), 373–376.
- [10] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, and Trevor Mudge, *Razor: A low-power pipeline based on circuit-level timing speculation*, Proceedings of the 36th International Symposium on Microarchitecture (MICRO-3603) (2003).
- [11] Chang hyun Kim and Ki whan Song, *Integrated circuit devices having metastability protection circuits therein*, US patent no. 6384619 (2002).
- [12] Randy H. Katz, *Contemporary logic design*, Benjamin/Cummings publishing company inc., 1993.
- [13] Ben Kuiper, *An overview of flip-flops*, 2009.

- [14] Sani Nassif, Kerry Bernstein, David J. Frank, Anne Gattiker, Wilfried Haensch, Brian L. Ji, Ed Nowak, Dale Pearson, and Norman J. Rohrer, *High Performance CMOS Variability in the 65nm Regime and Beyond*, IEEE International Electron Devices Meeting, 2007. (2007), 569–571.
- [15] S.R. Nassif, *Delay variability: sources, impacts and trends*, Proceedings of the IEEE International Solid-State Circuits Conference (2000), 368–369.
- [16] Liang-Teck Pang and Borivoje Nikolic, *Measurements and analysis of process variability in 90 nm cmos*, IEEE Journal of Solid-State Circuits **44** (2009), no. 5, 1655–1663.
- [17] Liang-Teck Pang and Borivoje Nikoli, *Measurement and Analysis of Variability in 45nm Strained-Si CMOS Technology*, Custom Integrated Circuits Conference, 2008 (2008), 129–132.
- [18] Behrooz Parhami, *Computer arithmetic, algorithms and hardware designs*, Oxford university press, 2000.
- [19] James Stewart, *Calculus, early transcendentals*, Thomson Learning, inc., 2003.
- [20] J. Tschanz, S. Narendra, Z. P. Chen, S. Borkar, M. Sachdev, and V. De, *Comparative delay and energy of single edge-triggered and dual edge-triggered pulsed flip-flops for high-performance microprocessors*, International Symposium on Low Power Electronics and Design (2001), 147 – 152.
- [21] Augustus K. Uht, *Going Beyond Worst-Case Specs with TEAtime*, Computer **37** (2004), no. 3, 51–56.
- [22] Jyh-Ming Wang, Sung-Chuan Fang, and Wu-Shiung Feng, *New efficient designs for xor and xnor functions on the transistorlevel*, IEEE Journal of Solid-State Circuits **29** (1994), 780 – 786.

Simulation results

This appendix chapter shows all the simulation results in more detail.

A.1 simulation with only dynamic delay variation

Simulation parameters:

- High phase length = 0.2 (*wanted high phase* in equation 3.10).
- contamination delay/critical path delay = $\frac{5}{9}$.
- delay variability: $\sigma_{total} = \sigma_{dynamic}$ and $\frac{\sigma_{total}}{\mu_{gate_delay}} = 0.1$.
- Simulation length = 100,000 clock cycles.

Figure A.1: Performance of a 4 stage AICBP with on the left side the average output period and on the right side the number of faults.

Figure A.2: Performance of a 8 stage AICBP with on the left side the average output period and on the right side the number of faults.

Figure A.3: Performance of a 16 stage AICBP with on the left side the average output period and on the right side the number of faults.

A.2 Simulation with static and dynamic delays

A.2.1 No static delay variations

Simulation parameters:

- High phase length = 0.2 (*wanted high phase* in equation 3.10).
- contamination delay/critical path delay = $\frac{5}{9}$.
- delay variability: $\frac{\sigma_{dynamic}}{\mu_{gate_delay}} = 0.06$ and $\frac{\sigma_{static}}{\mu_{gate_delay}} = 0.08$. In this simulation for all stages `STATIC_DELAY[stage_nr] = 0` applies.
- Simulation length = 100,000 clock cycles.

Figure A.4: Performance of a 4 stage AICBP. Left: average output. Right: number of faults.

Figure A.5: Performance of an 8 stage AICBP. Left: average output. Right: number of faults.

A.2.2 Increasing static delays

- High phase length = 0.2 (*wanted high phase* in equation 3.10).
- contamination delay/critical path delay = $\frac{5}{9}$.
- delay variability: $\frac{\sigma_{dynamic}}{\mu_{gate_delay}} = 0.06$ and $\frac{\sigma_{static}}{\mu_{gate_delay}} = 0.08$. The following sequence for static delays applies:
 4 stages: $-\sigma_{static}, 0, \sigma_{static}, 2\sigma_{static}$
 8 stages: $-\sigma_{static}, -\sigma_{static}, 0, 0, \sigma_{static}, \sigma_{static}, 2\sigma_{static}, 2\sigma_{static}$
- Simulation length = 100,000 clock cycles.

Figure A.6: Performance of a 4 stage AICBP. Left: average output. Right: number of faults.

Figure A.7: Performance of an 8 stage AICBP. Left: average output. Right: number of faults.

A.2.3 Decreasing static delays

- High phase length = 0.2 (*wanted high phase* in equation 3.10).
- contamination delay/critical path delay = $\frac{5}{9}$.
- delay variability: $\frac{\sigma_{dynamic}}{\mu_{gate_delay}} = 0.06$ and $\frac{\sigma_{static}}{\mu_{gate_delay}} = 0.08$. The following sequence for static delays applies:
 4 stages: $2\sigma_{static}, \sigma_{static}, 0, -\sigma_{static}$
 8 stages: $2\sigma_{static}, 2\sigma_{static}, \sigma_{static}, \sigma_{static}, 0, 0, -\sigma_{static}, -\sigma_{static}$
- Simulation length = 100,000 clock cycles.

Figure A.8: Performance of a 4 stage AICBP. Left: average output. Right: number of faults.

Figure A.9: Performance of an 8 stage AICBP. Left: average output. Right: number of faults.

A.2.4 One stage with a long static delay

- High phase length = 0.2 (*wanted high phase* in equation 3.10).
- contamination delay/critical path delay = $\frac{5}{9}$.
- delay variability: $\frac{\sigma_{dynamic}}{\mu_{gate_delay}} = 0.06$ and $\frac{\sigma_{static}}{\mu_{gate_delay}} = 0.08$. The following sequence for static delays applies:
 4 stages: 0, 0, $2\sigma_{static}$, 0
 8 stages: 0, 0, 0, 0, 0, $2\sigma_{static}$, 0, 0
- Simulation length = 100,000 clock cycles.

Figure A.10: Performance of a 4 stage AICBP. Left: average output. Right: number of faults.

Figure A.11: Performance of an 8 stage AICBP. Left: average output. Right: number of faults.

A.3 Simulation with reduced contamination delay

- High phase length = 0.2 (*wanted high phase* in equation 3.10).
- contamination delay/critical path delay = $\frac{1}{3}$.
- delay variability: $\sigma_{total} = \sigma_{dynamic}$ and $\frac{\sigma_{total}}{\mu_{gate_delay}} = 0.1$.
- Simulation length = 100,000 clock cycles.

Figure A.12: Performance of a 4 stage AICBP. Left: average output. Right: number of faults.

Figure A.13: Performance of an 8 stage AICBP. Left: average output. Right: number of faults. The performance results for spacing=0.7 are not shown, since they do not improve the performance.

A.4 Out of order execution

The simulation results for out of order execution as discussed in Chapter 7.

- High phase length = 0.2 (*wanted high phase* in equation 3.10).
- contamination delay/critical path delay = $\frac{5}{9}$.
- delay variability: $\sigma_{total} = \sigma_{dynamic}$ and $\frac{\sigma_{total}}{\mu_{gate_delay}} = 0.1$.
- Simulation length = 100,000 clock cycles.

Figure A.14: Performance of a 4 stage AICBP with out of order re-execution. Left: average output. Right: number of faults.

Figure A.15: Performance of an 8 stage AICBP with out of order re-execution. Left: average output. Right: number of faults.

A.5 Data dependent delay compensation

The simulation results for the AICBP augmented with the data dependent delay compensation system as discussed in Chapter 8.

A.5.1 Transparency period of 0%

- High phase length = 0.2 (*wanted high phase* in equation 3.10).
- contamination delay/critical path delay = $\frac{1}{3}$.
- delay variability: $\sigma_{total} = \sigma_{dynamic}$ and $\frac{\sigma_{total}}{\mu_{gate_delay}} = 0.1$.
- Simulation length = 100,000 clock cycles.
- Length largest transparency window = 0% of total chain length.
- Input data = uncorrelated.

Figure A.16: Performance of a 4 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.

Figure A.17: Performance of an 8 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.

A.5.2 Transparency period of 10% and uncorrelated input data

- High phase length = 0.2 (*wanted high phase* in equation 3.10).
- contamination delay/critical path delay = $\frac{1}{3}$.
- delay variability: $\sigma_{total} = \sigma_{dynamic}$ and $\frac{\sigma_{total}}{\mu_{gate_delay}} = 0.1$.
- Simulation length = 100,000 clock cycles.
- Length largest transparency window = 10% of total chain length.
- Input data = uncorrelated.

Figure A.18: Performance of a 4 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.

Figure A.19: Performance of an 8 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.

A.5.3 Transparency period of 10% and correlated input data

- High phase length = 0.2 (*wanted high phase* in equation 3.10).
- contamination delay/critical path delay = $\frac{1}{3}$.
- delay variability: $\sigma_{total} = \sigma_{dynamic}$ and $\frac{\sigma_{total}}{\mu_{gate_delay}} = 0.1$.
- Simulation length = 100,000 clock cycles.
- Length largest transparency window = 10% of total chain length.
- Input data = correlated.

Figure A.20: Performance of a 4 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.

Figure A.21: Performance of an 8 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.

A.5.4 Transparency period of 20% and uncorrelated input data

- High phase length = 0.2 (*wanted high phase* in equation 3.10).
- contamination delay/critical path delay = $\frac{1}{3}$.
- delay variability: $\sigma_{total} = \sigma_{dynamic}$ and $\frac{\sigma_{total}}{\mu_{gate_delay}} = 0.1$.
- Simulation length = 100,000 clock cycles.
- Length largest transparency window = 20% of total chain length.
- Input data = uncorrelated.

Figure A.22: Performance of a 4 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.

Figure A.23: Performance of an 8 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.

A.5.5 Transparency period of 20% and correlated input data

- High phase length = 0.2 (*wanted high phase* in equation 3.10).
- contamination delay/critical path delay = $\frac{1}{3}$.
- delay variability: $\sigma_{total} = \sigma_{dynamic}$ and $\frac{\sigma_{total}}{\mu_{gate_delay}} = 0.1$.
- Simulation length = 100,000 clock cycles.
- Length largest transparency window = 20% of total chain length.
- Input data = correlated.

Figure A.24: Performance of a 4 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.

Figure A.25: Performance of an 8 stage AICBP augmented with the data dependent delay compensation system. Left: average output. Right: number of faults.

List of Publications

- B. Kuiper, S. D. Cotofana, “Adaptive Clock Scheduling for Pipelined Structures”, in *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH’09)*, San Francisco, July 2009

Curriculum Vitae

Ben Kuiper was born in Leiderdorp, the Netherlands on the 20th of August 1984. He completed his secondary education at the Bonaventura college in Leiden in 2002. After that he pursued the degree of Bachelor of ICT (cum laude) at the Rijswijk University of Professional Technical Education. In 2006 he started with the bridging program for the M.Sc. degree in Computer Engineering, and currently he is pursuing the M.Sc. degree in Computer Engineering. Besides his education he worked part time as a software and hardware engineer at different companies. His main interests include embedded systems, digital design, computer architecture, and also control systems.