ASYNCHRONOUS WAVE PIPELINES FOR ENERGY EFFICIENT GIGAHERTZ VLSI

Vom Fachbereich Informatik der Technischen Universität Darmstadt genehmigte

DISSERTATION

zur Erlangung des akademischen Grades eines Doktor-Ingenieurs (Dr.-Ing.) von

Dipl.-Inform. Oliver Hauck aus Neustadt a. d. Weinstraße

Referenten der Arbeit: Prof. Dr.-Ing. Sorin A. Huss
Prof. Dr.-Ing. Wolfram H. Glauert

Tag der Einreichung: 27.12.2005
Tag der mündlichen Prüfung: 24.04.2006

Darmstadt 2007
D17
# Contents

Acknowledgments

1 Introduction
   1.1 Why Asynchronous ? ........................................ 8
   1.2 Why Wave Pipelining ? .................................... 9
   1.3 Thesis Outline ........................................... 9

2 Related Work
   2.1 Clocking and Latching .................................... 11
   2.2 Dynamic and Self-Resetting CMOS ..................... 12
   2.3 Wave Pipelining ......................................... 14
   2.4 Asynchronous Pipelines ................................ 15
   2.5 Summary ................................................ 17

3 Principles of Asynchronous Wave Pipelines ............... 19
   3.1 Wave Pipeline Timing Constraints ....................... 19
   3.2 Asynchronous Wave Pipeline (AWP) Architecture .... 25
   3.3 Pass Gate Circuits for AWPs ........................... 26
   3.4 AWPC MOS Circuit Design ................................ 28
      3.4.1 Balancing NMOS Logic Pulldown Paths .......... 33
      3.4.2 Single-Rail vs. Dual-Rail ......................... 34
      3.4.3 Single-Rail Design ................................ 36
      3.4.4 Dual-Rail Design ................................ 39
      3.4.5 Low Power Design ................................ 47
      3.4.6 Pulse-Width Control ............................... 50
   3.5 Analysis of AWPC MOS Delay ............................. 52
      3.5.1 Data-Dependent Gate Delay ......................... 53
      3.5.2 Device Process Variation ........................... 63
   3.6 Power Analysis .......................................... 69
   3.7 Embedding the AWP into Synchronous Systems ..... 71
   3.8 Summary ................................................ 73

4 64-Bit Pipelined Parallel Adder ............................ 75
   4.1 Overview ................................................. 75
   4.2 Brent-Kung/Kogge-Stone Adder Architecture ......... 75
   4.3 Adder Building Blocks .................................. 77
      4.3.1 Initial Propagate/Generate Gate .................. 77
      4.3.2 Group Propagate/Generate/Buffer Gate .......... 77
4.3.3 Final XOR ......................................................... 78
4.4 On-Chip Test Circuitry ........................................... 80
  4.4.1 VCO and Pulse Generators ................................. 81
4.5 Pattern Shift Register .......................................... 81
  4.5.1 Multiplexor-Based Pattern Generation ................. 82
4.6 Measurement Results ........................................... 83
4.7 Summary ......................................................... 87

5 Elliptic Curve Crypto Chip ..................................... 89
  5.1 Cryptography Background ................................... 89
    5.1.1 Elliptic Curves ........................................... 90
    5.1.2 Optimal Normal Basis Multiplication ................ 91
  5.2 Chip Implementation ......................................... 94
    5.2.1 Overview ................................................. 94
    5.2.2 Register File ............................................. 94
    5.2.3 Operand Registers ....................................... 97
    5.2.4 Massey-Omura Multiplier ............................... 97
    5.2.5 Result Sampling Register ............................... 97
    5.2.6 Pulse Counter ............................................ 98
    5.2.7 AWP XOR Test Circuit .................................... 99
  5.3 Measurement Results ......................................... 101

6 Conclusion ......................................................... 105
  6.1 Contributions ............................................... 105
  6.2 Outlook ....................................................... 106

A List of Original Publications ................................ 107

B Supervised Diploma Theses .................................... 109
Acknowledgments

When embarking on a doctoral thesis one profits from the support of a number of people. This is the place to say thanks:

To my parents and my country: For enabling my education.

To my advisor, Professor Sorin A. Huss: For hiring me and giving me the opportunity for research and the freedom to find my own way; and for his patience and support during the long period while I was at Infineon to finish the job.

To my co-advisor, Professor Wolfram H. Glauert: For his willingness to take the work of reading and commenting on the thesis.

To my manager at Infineon, Dr. Norbert Janssen: For letting me use the CAD software and testing infrastructure which was cruciable for completing the design and testing of the ECC chip; especially for granting me access to the EBeam facility.

To my family: My wife Bianca is still with me even though it sometimes looked like it would never end; fortunately, my 1½ year old son Adrian didn’t really notice that I was late with the work but will certainly enjoy that I have my time back for him now.
Chapter 1

Introduction

The relentless scaling of CMOS technology has resulted in feature sizes being measured in nanometers and timing in picoseconds. Meanwhile wafers have grown to 12 inches diameter, high end 64-bit processors burn 100W [NGS05], device counts are in the hundreds of millions, and the cost for new fabs now amounts to billions of dollars. The last decade has seen feature sizes shrinking from 0.5μm to 65nm and microprocessor clock frequencies rising from 100MHz to over 3GHz with integer units running at about 7GHz. Pipelined floating point multiplier frequency is approaching 10GHz [BJM+05]. The increase in clock frequency beyond that offered by technology scaling is due to aggressive use of pipelining for reducing the gates per cycle.

Pipelining is a fundamental design paradigm to increase throughput of digital circuits. Compared to parallelism it makes better use of the hardware. As an example, consider some combinational logic having latency of 10ns and 1ns difference between fastest and slowest path, rf. Figure 1. Then without pipelining the logic would be idle 90% at any time because of the 10ns latency only 1ns is occupied by the evaluation wave propagating through the circuit. Pipelining fully exploits the circuit by injecting new data tokens as soon as possible. In theory 10 tokens can be active in the logic at any time thus increasing the throughput by an order of magnitude. The lower limit for the cycle time to achieve correct operation, 1ns in this example, is determined by the spatiotemporal spread of the activity of any token in the logic.

![Figure 1. Pipelining principle.](image)

When implementing pipelining the principle task is to keep the data tokens separated from one another to insure correct operation. There are three different approaches to achieve this. The conventional approach is the synchronous pipeline where the logic is
divided into pipeline stages that are separated by latches or flipflops receiving a common clock signal. The pipeline throughput is determined by the longest path in any stage plus latching overhead. At short cycle times the latching overhead becomes significant. A fine-grain synchronous pipeline spends more devices on sequential elements than for logic. Clock generation and distribution power dominates the total power budget. Finally, latches and flipflops add to the latency of the pipeline. Synchronous pipeline frequency has been pushed into the region of diminishing returns illustrated by the recent move of major microprocessor developers into multi-threading and multi-core architectures [NGS05].

The competing implementation approaches to pipelines are asynchronous and wave pipelines. Briefly, in asynchronous pipelines the logic is as well partitioned into pipeline stages but the global clock signal is replaced by local communication between neighboring stages using a handshake protocol. This lends elasticity to asynchronous pipelines, i.e. the data rates at the input and output are decoupled from each other. Furthermore, every pipeline stage has its own local cycle time rather than being determined by the slowest pipeline stage.

In wave pipelining the logic is not partitioned into stages altogether. Rather, the circuit is designed to have nominally equalized path delays. In a delay balanced logic data tokens stay coherent when propagating through the logic and intrinsic capacitance provides for temporary storage. Very high throughput at low latency can be achieved. This comes at the cost of complicated design and difficult timing.

1.1 Why Asynchronous?

Asynchronous design can reap the following benefits:

Lower power: Asynchronous circuits avoid global clock infrastructure. Handshaking results in power dissipation when performing useful work only.

Less peak power and EM radiation: Unlike synchronous systems that exhibit large power spikes at every clock edge, power consumption in asynchronous circuits is not tied to a global clock and therefore more diffused. This relieves from stress on the power lines.

Higher performance: The local nature of asynchronous circuits can tighten timing margins. Performance is not dominated by the worst case, lower average case delay can be exploited.

Modularity of design: Asynchronous components equipped with handshaking allow for a building block approach to system design. An ensemble of such handshaking blocks does not depend on global timing issues.

The past decade has seen a rising interest in asynchronous design. Apart from the ASYNC conference series inaugurated in 1994 the topic has been featured in a special issue of the Proceedings of the IEEE [IEEE99] and several textbooks have been published, e.g. [SF01]. Asynchronous design is not likely to replace synchronous design in the large as in a commercial environment any departure from the well supported synchronous tool flow must be justified with significant benefits to be gained from asynchronous design. This may change as more tools supporting asynchronous design become available. Asynchronous design has been incorporated in an Itanium cache design [WWM+05]. There are designs that successfully exploit average case latency [SRG+01, BKYK99]. A prototype example of this class
1.2 Why Wave Pipelining?

Wave pipelining is promising for reasons similar to the ones mentioned above for asynchronous design:

**Very high throughput:** Pipeline cycle time is not determined by the longest path in any stage but by the degree to which path delays can be equalized.

**Lowest latency:** In applications where both throughput and latency count wave pipelines excel as no latches or flipflops block the road.

**Reduced average and peak power:** Like asynchronous design, wave pipelines cut down the clock and latch overhead. Synchronization is needed only at the input and output, the inwards of the wave pipeline do not load the clock tree.

**No partitioning overhead:** Wave pipelined logic has no dead time that impacts cycle time or latency. In contrast, when logic cannot be partitioned into stages of equal delay for a synchronous pipeline all but the slowest stage will have some idle time during the pipeline cycle. This seriously impacts also latency as this dead time occurs in all but one stages down the pipeline.

**Non-integer degree of pipelining:** Wave pipelines are not restricted to integer numbers of tokens or data waves. It is perfectly possible to have, e.g., 1ns latency and 700ps cycle time, therefore effectively 1.43 waves.

The delay balancing of the pipeline over all possible patterns and accounting for environmental variations and noise is very complex and has its own costs. Consequently the method has so far mainly been restricted to regular caches [HSR+98, HAA+00].

1.3 Thesis Outline

The emphasis of the thesis is on circuit design for high speed datapaths using asynchronous wave pipelines. First, asynchronous timing for the wave pipeline is shown to have advantages over clocked wave pipelines. Second, a self-resetting logic called AWPCMOS is developed that overcomes the drawbacks of previously proposed circuits for wave pipelining. The ideas are put into silicon by test vehicles from arithmetic and cryptography, respectively.
CHAPTER 1. INTRODUCTION

In most other research “asynchronous” means handshaking and elastic operation whereas in this work it means absence of a global clock only. Not to be constrained by a rigid clock is the essential of asynchronous design. The view held in this thesis is that the benefit of full handshake and elasticity depend on the application. It is not a good idea just always to replace the clock by handshaking. After all, handshake signals need timing margins like clock signals. Rather, the optimum is probably an application specific mix of as much local timing as possible and as much global timing as necessary.

- Chapter 2 provides an overview on related work relevant to this thesis. Trends in clocking and latching are reviewed as well as self-resetting techniques. Prior work on asynchronous and wave pipelines is summarized.

- Chapter 3 is the core of the thesis. It lays out in detail the asynchronous wave pipeline method. Starting from general wave pipeline timing constraints, the choice of the circuit style AWPCMOS is motivated. Subsequently, AWPCMOS is analyzed in detail.

- Chapter 4 presents a Giga-Hertz 64-bit carry-lookahead adder with on-chip test circuitry fabricated in 0.6μm CMOS. The building blocks of the chip will be detailed together with simulation and measurement results.

- Chapter 5 describes the design of a Giga-Hertz 270-bit finite-field multiplier for elliptic-curve cryptography in 0.35μm CMOS.

- Chapter 6 gives a summary of the results of this work and concludes with an outlook.
Chapter 2

Related Work

The work presented in this thesis has intersections with several other areas of VLSI design. In the following the thesis will be put into context with these related disciplines. Specifically, a brief summary of state-of-the-art clocking and latching methodologies will be given. Then, dynamic circuit styles as used in high performance microprocessors with an emphasis on self-resetting techniques will be reviewed. Following, wave pipeling approaches are described. Finally, available architectures for asynchronous pipelines are explained.

2.1 Clocking and Latching

The clocking methodology directly impacts the performance of VLSI systems. In high performance systems with cycle times below one nanosecond, clocking is intimately related to logic and latching. This is especially true when clocked dynamic circuit styles are used. The achievable performance is limited primarily by power consumption, not clock skew.

The design objectives for the clock signal are low skew, low jitter and nearly 50% duty cycle [Bailey01]. Usually a high speed internal clock is derived from the slower external clock by a frequency multiplying PLL. This root clock has to be amplified and distributed globally over the die. There are fundamentally three approaches being used: RC matched trees, grids or a combination of both [RMW+01, RCE+02]. Because hold time violations occur in short paths it is imperative that the local skew, i.e. between nearby latches, be controlled. Global skew between chip corners will always be larger but this is not dramatic as no signal can travel over the chip in a single cycle.

Grids tie together the final clock buffer outputs resulting in an averaging effect that benefits local skew. Furthermore, the clock grid can be designed upfront and independently of the logic circuits whereas matched trees have to be recalibrated and rerouted whenever the logic and hence the clock load situation changes thus negatively impacting design productivity. The grid’s capacitance however increases the total clock power. Furthermore the buffers driving adjacent points of the grid must be well matched in delay in order that the buffers do not fight each other resulting in increased current and reduced clock drive.

Jitter in the clock signal reduces the available cycle time for logic because of buffer delay variations in the global clock distribution network due to power-supply noise. In addition, across-chip channel length variation (ACLV) inevitably introduces jitter. For these reasons it is desirable to keep the latency of the global clock distribution short.

Clock power can be tremendous. For the first two Alpha processors it amounted to 40% of the total power [GBP+98b]. As power will ultimately limit further increases of
performance in VLSI systems it is worthwhile looking for alternatives. Advanced approaches to reduce the clock power consumption and skew/jitter include inductance resonating with the clock frequency [CSR05] and standing-wave oscillators [MYH+03], respectively.

The majority of high performance systems employs a single-phase edge-triggered latching methodology, as opposed to multi-phase level-sensitive latching. For dynamic logic however there is a method called skew-tolerant domino which uses multi-phase clocks to absorb skew and eliminate latches [HH97]. Instead of master-slave style flipflops pulsed latches are often used. Their advantage is less area and clock load and the potential for time-borrowing [PBS+96].

2.2 Dynamic and Self-Resetting CMOS

Dynamic or precharge logic has traditionally been used in high performance processors where static logic cannot fulfill the latency and cycle time demands. It implies, however, liabilities such as the need for clocking and manual design and decreased robustness due to charge sharing issues. Furthermore, single-rail domino logic is restricted to monotone boolean functions like AND, OR. Functions including inversion like XOR can be implemented with dual-rail domino which is a complete logic family. Another dynamic logic flavor is delayed-reset logic. Figure 2 shows an example from the guTS pipeline [S+98], the first processor to reach 1.0GHz in 1998. It can be seen that the precharge is not global but propagated along the datapath. This allows to get rid of the footer in all but the first gate which improves speed. When using a global precharge the footer is needed to prevent cross-current in downstream logic during precharge.

![Figure 2](image-url)  
*Figure 2. Original Figure 3 of [S+98].*

The problem with dynamic logic is that a performance advantage over static logic is only achieved with appropriate clocking of precharge and evaluate which inevitably entails
2.2. DYNAMIC AND SELF-RESETTING CMOS

power consumption and timing margins. Reducing the clock load of dynamic circuits by controlling the precharge/evaluate locally is the idea behind self-resetting or SRCMOS. The earliest reference to self-resetting circuitry appears to be the March 1988 US patent 4,728,827 by Ann K. Woo of AMD [Woo88]. The aim of the patent was to enjoy the benefits of dynamic logic in a PLA context, i.e. faster output transition compared to a pseudo-NMOS version, without the need for a clock; applications to dynamic circuits in general, let alone wave pipelining, were not considered.

![Figure 3](image1.png)

Figure 3. Original Figure of [Woo88].

A feedback inverter timing chain senses evaluate discharge and subsequently initiates precharge without the need of an external clock. Because conceptually the precharge follows its triggering evaluate such form of logic is also called postcharge logic. The first design to aggressively exploit SRCMOS was an SRAM by Chappell et al. [CCS+91].

Figure 4 shows the use of self-resetting circuits in a Pentium 4 processor [HUS+01]. The timing chain at the top of the figure triggers precharge by a high input at the feedback NAND. Only when the gate has been discharged by the preceding evaluation the NAND fires and activates the precharge PMOS. The gate itself terminates the precharge when the precharge node is high again. This relaxes the timing chain as an accurate precharge pulse is not needed.

![Figure 4](image2.png)

Figure 4. Original Figure 9 of [HUS+01].
2.3 Wave Pipelining

Wave pipelining has been first described by Cotten [Cotten69] under the name *maximum rate pipelining*. Burleson et al. in [BCK+98] give a nice overview and trace the history of the method and wave pipelined designs. The very influential “bubble pipelined” ECL SRAM paper at ISSCC 1991 by Chappell et al. from IBM Yorktown Lab [CCS+91] started a whole wave of such pipelined RAM and cache designs which continues to present [THT+95, HAA+00]. The latest reference of a RAM design using wave pipelines techniques — in this instance not in the datapath but for CAS latency control — dates from ISSCC 2004 [LJK+05]. As memories are regular structures containing only few unique circuits like decoders, muxes, and sense amps they lend themselves well for wave pipelining because managing the delay variations is feasible.

For generic datapath circuits the delay variations due to PVT and data dependency become acute. There has not been much progress before Michael Flynn of Stanford University in the Nineties revived the technique for SNAP — the Sub-Nanosecond Arithmetic Project. The PhD thesis of Wong deals with a delay tuning method targeted for bipolar ECL circuits which have favorable properties for wave pipelining. The tuning consists of two phases, rough tuning and fine tuning. During rough tuning the circuit gets levelized and short paths are padded to the longest path. Fine tuning then minimizes delay variation by controlling bias voltages for the current mode gates. While the details of the tuning method are specific to ECL the principle approach is the same for CMOS. Nowka’s PhD thesis [Nowka95], as well under the supervision of M. Flynn, deals with delay minimization methods in CMOS. As the final work in this thread, Klass in his PhD thesis [Klass94] develops equations for wave pipelining timing considering environmental changes. However, both Nowka and Klass only consider static logic styles like those shown in Figure 5.

C. T. Gray, R. Cavin, W. Liu and other researchers from North Carolina State University develop timing constraints for single and multiple stage wave pipelines including feedback [GLC94b]. A 250MHz adder in 2μm CMOS has been fabricated [LGF+94]. The research of the NCSU group is nicely summarized in a book [GLC94a].

Pass transistor logic at first sight looks promising for wave pipelining and has been the subject of several works [GN95, PS97, SZ98]. In Chapter 3 we will see that pass gate logic actually has severe drawbacks and causes more delay variation than one might expect.

W. Burleson is the only one to consider dynamic logic for wave pipelines with wave domino logic [LB95]. As wave domino still relies on a clock signal that is propagated along the logic and synchronizes data at every stage this is more of an alternative clocking for domino logic similar to clock-delayed logic than wave pipelining.
2.4 Asynchronous Pipelines

Sutherland’s influential Turing Award paper on Micropipelines [Sutherland89] is often taken as the seed of the renaissance of asynchronous circuit design culminating in the inaugural ASYNC conference in 1994. However, it is preceded by two little known papers by Terada et al. emphasizing the very same ideas of clockless data-driven operation, elasticity by handshaking protocol, and modularity of design [KTT+88, YSK+89]. However no overview of asynchronous design can be complete without mentioning Seitz’ seminal chapter 7 on
system timing in the Mead/Conway book [Seitz80] which anticipates, e.g., the bundled data representation which is the basis for the major part of single-rail datapaths. The original Micropipeline is reproduced in Figure 6 and works as follows.

![Micropipeline Diagram](image)

Figure 6. Original Figure 4 of [DW95].

Initially R(in) rises to signal new data available at D(in). When the leftmost stage is free then the upper left C-element fires causing the capture-pass register to capture the data. The Cd output rises when the capture is done which is acknowledged via A(in). After a fixed delay modeling the worst-case path of the logic R(1) rises. When the subsequent stage has completed previous capture it is ready to receive new data thus asserting A(1) passing new data into the logic. The independent handshaking at the input and output sides gives rise to elasticity, i.e. this structure can just be dropped in at any place obeying to the handshake protocol. When empty it behaves like combinational logic, when full it stalls. The fact that there is only local communication between neighboring stages results in short cycle times. It implies as well, though, that in the full pipeline case new data at the input has to wait until acknowledge from the output side has been rippled through. Therefore the local nature of communication is not guaranteed to yield advantages on system level.

The impact of Micropipelines lies more in the framework than the details given in [Sutherland89]. It was soon recognized that the two-phase or transition signaling protocol is not well suited for implementation. In practice four-phase realizations prevail [DW95].

A significant development in asynchronous pipelines is the work by T. Williams. He shows how a self-timed pipeline can be constructed using dual-rail domino logic so that no latches are needed and the handshaking does not enter the critical path which is the so-called zero-overhead principle [Williams01, WH91].

Williams’s work has been subsequently developed further and optimized by M. Singh in a series of papers [SM00a, SMM00b, SMM01, STR+02]. The important difference between the proposed asynchronous wave pipeline and the work of Singh is that in his pipeline the handshake circuitry in every stage has to clock the pipeline logic over the width of the datapath. The same is true for a recent wave pipeline-related circuit style called “surfing” where the “fast” signals have to supply every gate like a clock [YWG05]. In the work presented here only the input stage is clocked to launch data into the pipeline.

Sutherland and his coworkers from the SUN asynchronous research group proposed a new pipeline architecture called GasP at ASYNC 2001 [SF01, CLJ+01, Ebergen01, SL01]. The basic pipeline stage is shown in Figure 7.
The GasP architecture was proposed only after self-resetting AWPCMOS in [HKH00]. It is somewhat similar to AWPCMOS in that it uses self-resetting circuitry and emphasizes balanced delays. Only in the limit where no logic is present in the datapath, i.e., FIFO, it achieves the same cycle time as AWPCMOS, roughly six inverter delays.

2.5 Summary

Summarizing, we have seen that achieving low skew/jitter while limiting power with ever shrinking cycle times in synchronous clock distribution becomes increasingly infeasible. Self-resetting CMOS is attractive as it optimizes the timing and reduces the clock load of dynamic logic. Wave pipelining can achieve very high throughput at reduced clock/latch overhead but is hampered by ineffective static circuit styles and timing complexities. The latter problem can be alleviated by operating wave pipelines asynchronously. Existing asynchronous pipeline approaches have the advantage of elasticity. Their throughput, like conventional synchronous pipelines, is still determined by the longest logic path. Thus they achieve short cycle times only as FIFOs — when no logic is present in the pipeline — or in the case of fine-grain gate-level pipelines. However in the latter case the overhead due to handshake power and timing margins are substantial — like in synchronous pipelines.

The asynchronous wave pipeline as proposed in this thesis always delivers six inverter cycle time, independent of pipeline logic. The self-resetting nature of the proposed AWPCMOS logic keeps the advantage of dynamic logic while avoiding an external clock. This comes at the cost of increased design complexity and less flexibility.

Ultimately, this thesis can be characterized as the generalization of the approach of [CCS91] from memory to general datapath circuits.
Chapter 3

Principles of Asynchronous Wave Pipelines

The schemes reviewed in the preceding chapter all have their respective shortcomings. The synchronous approach suffers from power and area overhead and performance loss resulting from a high-speed clock and storage elements. This is alleviated somewhat by circuit styles as SRMOS or Skew-Tolerant Domino. The conventional wave pipeline is hindered by difficult timing and suboptimal circuit styles. Finally, asynchronous pipelines obeying a pure request-acknowledge handshake protocol are at risk of introducing more overhead from handshaking and completion detection than they may benefit.

These considerations led to the idea of combining asynchronous operation with wave pipelining. Two factors are key to the success of this undertaking: first, we have to get rid of the global clock without incurring the traditional asynchronous overhead; second, to make wave pipelining more generally usable, a better circuit style has to be invented.

3.1 Wave Pipeline Timing Constraints

We begin with a review of timing constraints for wave pipelines following [GLC94a, GLC94b]. A single stage system is shown in Figure 8. Wave pipelined combinational logic is enclosed by rising edge-triggered input and output registers skewed by $\Delta_i$ and $\Delta_o$ with respect to a global clock $GCLK$.

![Figure 8. Single stage wave pipeline system.](image-url)
The combinational circuit consists of nodes interconnected by gates. This is modeled by a set \( G \) of gate output nodes and a set \( C \) of node pairs for the connections. For example, if \( i, j \in G \) then \((i,j) \in C\) means that node \( i \) is input to a gate driving node \( j \).

For the circuit we define timing parameters:

\[
\begin{align*}
t_{\text{max}} & : \text{max delay from input register to output register} \\
t_{\text{min}} & : \text{min delay from input register to output register} \\
t_{\text{max}}(i) & : \text{max delay from input register to internal node } i \\
t_{\text{min}}(i) & : \text{min delay from input register to internal node } i \\
t_{\text{slack}}(i) & : \text{min stable time of inputs to gate with output node } i \\
\end{align*}
\]

For the clock and registers we define timing parameters:

\[
\begin{align*}
T_{\text{clk}} & : \text{clock period} \\
\Delta_i & : \text{intentional input clock skew} \\
\Delta_o & : \text{intentional output clock skew} \\
\Delta_i & : \Delta_o - \Delta_i \\
t_{\text{skew}} & : \text{unintentional clock skew at input and output register} \\
t_d & : \text{clock-to-Q delay of register} \\
t_{\text{setup}} & : \text{setup time of register} \\
t_{\text{hold}} & : \text{hold time of register} \\
\end{align*}
\]

For notational convenience a simpler formulation of clock skew is used than in [GLC94b]. The notation used here means that a nominal clock edge may be early or late by some time \( t_{\text{skew}} \) and that this is so both at the input and output register. The space-time diagram in Figure 9 shows that a wave launched by the input clock at time \( \Delta_i \) is captured by the output clock edge at time \( t = \Delta_o + T_{\text{clk}} \). The gray cones illustrate how the path delay difference \( t_{\text{max}} - t_{\text{min}} \) increases while the wave propagates through the logic. In general the output clock captures data \( k \) clock cycles after the launching edge:

\[
t_{\text{sample}} = k \cdot T_{\text{clk}} + \Delta_o \tag{3.1}
\]

The correct sampling of data in a wave pipeline gives rise to a two-sided timing constraint. First, the latest data must arrive in time at the output register,

\[
\Delta_i + t_{\text{skew}} + t_d + t_{\text{max}} \leq t_{\text{sample}} - t_{\text{skew}} - t_{\text{setup}} \tag{3.2}
\]

Second, the earliest data must not collide with the previous wave,

\[
\Delta_i - t_{\text{skew}} + t_d + t_{\text{min}} \geq t_{\text{sample}} - T_{\text{clk}} + t_{\text{skew}} + t_{\text{hold}}, \tag{3.3}
\]

with \( t_{\text{sample}} \) defined in (3.1). Equations (3.2) and (3.3) can be verified by consulting Figure 9.
3.1. WAVE PIPELINE TIMING CONSTRAINTS

![Diagram of wave pipeline timing constraints]

**Figure 9.** Space-time diagram for single stage system with $k = 1$.

Rearranging (3.2) and (3.3) we get

$$
\Delta_i + 2t_{\text{skew}} + t_d + t_{\text{max}} + t_{\text{setup}} \leq t_{\text{sample}} \leq \Delta_i - 2t_{\text{skew}} + t_d + t_{\text{min}} + T_{\text{clk}} - t_{\text{hold}}
$$

(3.4)

By transitivity, we get the following *register constraint* for the clock period from (3.4):

$$
T_{\text{clk}} \geq t_{\text{max}} - t_{\text{min}} + t_{\text{setup}} + t_{\text{hold}} + 4t_{\text{skew}}
$$

(3.5)

Equation (3.5) says that the cycle time of a wave pipeline is determined by the maximum path delay difference plus sequencing overhead and clock skew. Several factors contribute to path delay difference, namely logic depth measured in gates, interconnect delay and parasitic effects due to data-dependent gate delays, cross-coupling and process, voltage and temperature (PVT) variations. The challenge is to control them all in order to minimize $t_{\text{max}} - t_{\text{min}}$.

In addition to the register constraint (3.5) an equivalent constraint has to hold for every internal node. Consider the space-time diagram in Figure 10 showing a collision of waves at an internal node. The path delay difference increases to a maximum at half of the logic depth and decreases towards the output register. While the register constraint (3.5) is satisfied malfunction occurs caused by the collision of waves in the logic. In order for the wave pipeline to function it is necessary to have the waves separated at *every node*, and not just the output register. Figure 10 shows further that path delay variation is irreversible. While $t_{\text{max}} - t_{\text{min}}$ is decreasing in the second half of the logic this is of no help because the bottleneck remains at the node with the largest $t_{\text{max}} - t_{\text{min}}$. 
The bound for the clock period due to the \textit{internal node constraint} is given by

\[ T_{\text{clk}} \geq \max_{(i,j) \in C} (t_{\text{max}}(i)) - \min_{(i,j) \in C} (t_{\text{min}}(i)) + t_{\text{stable}}(j) + 2t_{\text{skew}} \quad \forall j \in \mathcal{G} \quad (3.6) \]

The internal node constraint (3.6) is similar to the register constraint (3.5). Maximum and minimum delay are computed as maximum and minimum of $t_{\text{max}}$ and $t_{\text{min}}$, respectively, of all inputs of gate with output node $j$. The sequencing overhead is replaced by $t_{\text{stable}}$ and the skew has to be budgeted for only twice because only the input register has to be considered.

The feasibility regions for different values of $k$ are illustrated in the constraint space diagram in Figure 11 relating $\Delta$ and $T_{\text{clk}}$. For clarity we set register overhead and clock skew to zero and assume that the internal node constraint is not active, i.e. the clock period will be determined by the register constraint. Solving (3.2) and (3.3) for $\Delta$ we get

\[ t_{\text{max}} - k \cdot T_{\text{clk}} \leq \Delta \leq t_{\text{min}} - (k - 1) \cdot T_{\text{clk}} \quad (3.7) \]

Using (3.7) we can compute the feasibility regions for $k$. For example,

\[ k = 0 \quad \Rightarrow \quad t_{\text{max}} \leq \Delta \leq t_{\text{min}} + T_{\text{clk}} \]

\[ k = 1 \quad \Rightarrow \quad t_{\text{max}} - T_{\text{clk}} \leq \Delta \leq t_{\text{min}} \]

Figure 11 shows that the limit $T_{\text{clk}} = t_{\text{max}} - t_{\text{min}}$ can be reached for every $k$, however for $k > 1$ the circuit does not work for every $T_{\text{clk}} > t_{\text{max}} - t_{\text{min}}$. This becomes evident if we write (3.7) as

\[ \frac{t_{\text{max}} - \Delta}{k} \leq T_{\text{clk}} \leq \frac{t_{\text{min}} - \Delta}{k - 1} \quad (3.8) \]

Note that a conventional pipeline is characterized by $\Delta = 0$, $k = 1$. This is consistent with the notion that conventionally there is a global clock only without skew ($\Delta = 0$) and
the data is latched one clock edge after the launching edge \( k = 1 \). For this case (3.8) immediately gives \( T_{clk} \geq t_{max} \) as expected for a conventional pipeline.

Equation (3.8) shows further that for larger \( k \) the feasible frequency range decreases. In [GLC94a, chapter 6.4.2] it is shown that operation in the high-\( k \) regime is also more prone to parametric variations. This together with the discontinuous frequency range makes wave pipelining with \( k > 1 \) not attractive.

The parameter \( k \) is not directly set but determined by \( \Delta \) and \( T_{clk} \). For a given \( T_{clk} \) the intentional skew \( \Delta \) is the only design parameter. Due to (3.1) a lower value for \( k \) means a higher \( \Delta \) and vice versa. So far the skew \( \Delta \) has been treated as an ideal value. The physical realization of delay elements, however, is always problematic because realizable delays are subject to limited granularity and accuracy. Figure 11 shows that the permissible deviation in \( \Delta \) decreases with shrinking clock period. For a given \( T_{clk} \) the range for possible values for \( \Delta \) is given by (3.7) as \( t_{min} - t_{max} + T_{clk} \) and is independent of \( k \).

The case of a zero-skew wave pipeline \( (\Delta = 0) \) is not attractive either as it avoids the delay elements but requires large \( k \) to achieve high frequency.

Finally, the above mentioned cases \( k = 0, k = 1 \) allow operation at any frequency up to the performance limit \( f_{max} = 1/(t_{max} - t_{min}) \). The choice \( k = 0 \) requires \( \Delta = t_{max} \), a matched delay, which is a common structure in asynchronous pipeline design.
Figure 11. Constraint space diagram.
3.2 Asynchronous Wave Pipeline (AWP) Architecture

As is evident from the discussion of their timing constraints, wave pipelines do in general have a discontinuous range of operating frequencies. However, it is highly desirable that all buildings blocks and hence the whole system are able to run in a continuous range from some minimum frequency up to the performance limit. This is the normal behaviour of both synchronous and asynchronous systems. There is one case, namely $k = 0$ and $\Delta = t_{\text{max}}$, where this holds for the wave pipeline as well, so we adopt the generic structure in Figure 12 for the Asynchronous Wave Pipeline, thereafter called AWP.

![Figure 12. Asynchronous Wave Pipeline architecture.](image)

This structure consists of a wave pipelined combinational logic realizing some Boolean mapping $f$ and a request delay line. The request is taken from the asynchronous world, the acknowledge in contrast is abandoned. This results in the AWP being inelastic, i.e., it doesn't tell when it is ready to accept new data. While asynchrony is often regarded as synonymous to elasticity of operation, in this work it just means absence of a global clock. While elasticity is a nice feature, it is not guaranteed to yield performance benefit on system level, but is certainly responsible for much of the overhead of asynchronous circuits.

The request, on the other hand, is mandatory, as it signals when data is valid and can be latched. The request information propagates along the datapath through a delay element that models the latency of the logic. Data and associated request must be coherent, and thus the request delay line is wave pipelined like the datapath. Propagating the request information with the data has two important consequences to the notion of asynchrony emphasized in this work. First, it frees us from the global clock as only enough drive for one data word has to be provided. The request signal may therefore be seen as a local clock. Second, the coherence of data and request allows for arbitrary data streams. As opposed to a periodic synchronous clocking, the data rate may be lowered to accommodate the environment. Furthermore, it is possible to have data bursts separated by idle phases with no activity. This is not possible if the same clock is used for launching and latching data.

In asynchronous pipeline schemes two approaches exist for deriving $\text{req}_{\text{out}}$ from $\text{req}_{\text{in}}$. In the bundled data model [Sutherland89], [Seitz80, p. 256] a delay element is used. Opposed to the AWP, this delay is only used between two pipeline stages and thus not pipelined by itself. The other method is to use a completion detector. As the latter, best realized by some tree structure, has a latency penalty of $O(\log n)$ and an area penalty of $O(n)$, with $n$ being the data path width, this work does not consider completion detectors.

Critical to the working of the AWP is the coherence of data and request to enable latching. This has to be assured by the implementation of the delay line. The dotted lines...
in Figure 12 between the delay line and the logic indicate that the delay has to track the logic. It is possible to have no physical connection at all between request line and logic until latching takes place. Then the achievable throughput is determined by the accuracy of the delay. The accuracy requirement can be relaxed by introducing some amount of physical coupling between data and request. This comes at the price of loading the request line and represents a step back to conventional pipelines. The optimum may lie somewhere in the middle.

The case \( k = 0, \Delta = t_{\text{max}} \) is mentioned in [Klass94, pp. 34–36, and [Nowka95, p. 105]. It is recognized that the discontinuity of operating frequency is avoided thereby but the fact that it enables aperiodic operation is not further exploited.

While synchronization of the wave pipeline is eased by the asynchronous operation, the circuit style for implementing logic has to allow tight control over delays. The sources for delays are data dependency, variations due to process, voltage and temperature (PVT), and noise including IR drop and crosstalk.

Obviously the AWP architecture relies on circuits that allow delay control without overhead. Two circuit families have been experimented with, static pass transistor logic and a novel self-resetting logic called AWPCMOS. First, pass transistor logic is described and its drawbacks resulting in the development of AWPCMOS.

### 3.3 Pass Gate Circuits for AWPs

At the beginning of our investigation into AWP experiments with pass transistor logic have been performed. There are three circuit styles of potential interest for wave pipelining. These are complementary pass transistor logic (CPL) [Y+90], double pass transistor logic (DPL) [S+93, PS97], and wave pipelined transmission gate logic (WTGL) [SZ98, SMM96]. All three are of complementary nature and feature gate delays nominally independent from inputs. In CPL either one of the NMOS switches is activated and passes the desired logic level to the inverter output buffer. A disadvantage of the shown AND/NAND gate is that it loads the preceding gate asymmetrically as \( A, \overline{A} \) are used twice as inputs but \( B, \overline{B} \) only once. The inverter provides output drive and compensates for NMOS threshold drops. This as well leads to different gate delays.

DPL does not need an output inverter as the gate is designed so that always a NMOS is on when a zero has to be passed and a PMOS is on for a one. DPL does not suffer from threshold drop but the gate delay is not exactly the same for all input situations. E. g., if \( A = B = 1 \) then the AND output is connected to two PMOS devices passing signals which are one. If \( A = 1, B = 0 \) then the AND output is connected to \( GND \) via a NMOS and to \( B = 0 \) via a PMOS. The latter case has smaller delay as the connection to a fixed supply has less series resistance. DPL being a pure pass transistor structure saves power but needs an inverter after some stages to counteract series device delay.

Finally, WTGL provides full swings by using transmission gates. The output inverters are only for providing drive strength and could be omitted. As WTGL then saves two devices over DPL with similar delay characteristics, it is the best choice of the three circuit families.
3.3. **PASS GATE CIRCUITS FOR AWPS**

![Diagrams of pass gate circuits](image)

a) Complementary pass transistor logic (CPL)

b) Double Pass Transistor Logic (DPL)

c) Wave-Pipelined Transmission Gate Logic (WTGL)

**Figure 13.** AND/NAND structures in pass transistor styles: 
   a) CPL  b) DPL  c) WTGL

However, there are the following drawbacks common to all pass transistor circuits:

- Different gate delays depending on whether a logic device just becomes conducting by an input changing value or was already conducting before. This effect characterizes all static logic which is based on levels. E. g., the pulldown path of a static 2-NAND...
CHAPTER 3. PRINCIPLES OF ASYNCHRONOUS WAVE PIPELINES

gate comprises two series NMOS devices. The pulldown delay is largest when both
gate inputs switch from zero to one simultaneously. It is smaller when the NMOS
connected to the output had a one at its gate before the NMOS connected to GND
is switched on. This is because the channel in the former device has already formed.
The smallest pulldown delay results from the NMOS at GND activated before the
one at the output side, because the grounded device has time to discharge in advance
the stacked node between the series devices.

In addition to the varying delay of the devices themselves, the load they present to
preceding gates’ outputs varies due to data dependent gate capacitance [GBP+98b].
The effective gate capacitance depends on the source and drain voltages.

- Different gate delays, depending on whether signals are connected to gate or source /
drain terminals of a transistor. This is because the RC load that the inputs present
is different: a gate normally has a higher capacitance than a source/drain region but
infinite series resistance. In contrast, a signal connected to source/drain is loaded by
a finite series channel resistance resulting in a larger RC time constant.

- Different gate delays, depending on whether a signal or supply voltage is passed. This
is because the connection to a supply has less resistance and is permanently “on”.
Thus, it is always faster than passing an input signal.

Apart from these inherent problems, CPL suffers from source-follower delay due to
threshold drop and body effect whereas DPL and WTGL are hindered by slow PMOS logic
devices. Only DPL features balanced input loads. Long unbuffered pass transistor chains
are particularly sensitive to variations in effective gate length \( L_{eff} \) and threshold voltage \( V_t \).

Furthermore, common with all static circuits that at best refresh, but not regenerate
logic values, pass gate structures are prone to the pulse shrinking problem. This means
that at high frequencies, when changing signal levels result in narrow pulses, asymmetrical
propagation delays for rising and falling transitions in a long logic path can lead to total
decay of such pulses and thus to logic failure.

If we were to avoid all the above mentioned problems, our preferred logic circuits would
have the following properties: pulsed logic, i.e., logic paths with series devices are only
activated by all transistors changing from off state to on; inputs are coupled to gates only;
pullup and pulldown paths that end at supplies only; only NMOS devices are to be used
for logic, and only for connecting to GND; the pulse is being regenerated at every gate.

It turns out that these are precisely the defining properties of what will be our cir-
cuit style of choice for AWPs. Pulsed operation means local precharge which eases delay
balancing as we will see in the next section.

3.4 AWPCMOS Circuit Design

Dynamic precharge logic is attractive for wave pipelining as only the NMOS pulldown
section of a gate has to be balanced while the PMOS precharge pullup is fixed. However,
the precharge and evaluate devices need a clock signal. A global clock is clearly inacceptable
for our goal. Even SRCMOS [HGS+99] and Wave Domino [LB95] load a local clock with a
full slice of the data path. Ideally, we would like to have a gate with a fully local self-reset
and minimal overhead. Figure 14 shows such a gate performing a logical AND. We assign
the name AWPCMOS to this kind of circuit which will be our preferred logic family for implementing AWPs.

![AWPCMOS Circuit Diagram](image)

**Figure 14.** Basic AWPCMOS AND gate.

This is a pulsed gate, i.e., it accepts pulses as inputs and generates a pulse on the output side. The operation is as follows, cf. Figure 15. Initially, the precharge node $X$ is at $VDD$. As long as there are no pulses on $a$ and $b$ there is no path to $VSS$, and $X$ will stay at $VDD$. After the output drive inverter $Inv$ 1 we have $y = 0 = 0 \cdot 0 = a \cdot b$ which activates the *keeper* device $P1$. The keeper counteracts charge sharing when either one of $a$ or $b$ has a pulse. The output $y$ is fed back over the feedback inverter $Inv$ 2. We have at the feedback node $Z = 1$ which disables the *precharge* device $P2$. This is a self-consistent situation: as $X$ is still precharged, there is nothing to do for $P2$.

Now consider the case where two pulses arrive at $a$ and $b$ with enough overlap so that the logic devices $N1$ and $N2$ are both conducting for a short time. This will pull down node $X$ resulting in a rising edge at the output $y$. Note that the pulldown has to fight the keeper which goes off not before the output has gone high. This will inevitably draw some current. However, this is the standard method to increase the robustness and is found in conventional single-rail footless domino circuits as well. The rising edge at the output is fed back via $Inv$ 2 and causes a falling edge at node $Z$ which in turn activates the precharge transistor. $P2$ pulls up node $X$ again at which time $a$ and $b$ should have shut off. Otherwise crowbar current would occur. $X$ being pulled up subsequently leads to $y$ going to low completing the output pulse, then reactivating $P1$ and finally shutting off $P2$. Altogether $y = a \cdot b$ holds in this case also, but unlike level-based logic a pulse is generated at the output.

The operation of the logic gate is self-resetting and thus better visualized by talking about postcharging instead of precharging. The gate has a stable state at $y = 0$ and replies to disturbance on its inputs with a pulse at the output. The postcharge supplies the lost charge after emitting the pulse to reach the stable state again.
Figure 15. AWPCMOS AND gate simulation.

1. $t_{\text{pulldown}, X}$
   Time from asserting input pulses to pulling down the precharge node $X$ through NMOS logic.

2. $t_{\text{rise, inv\_out(output\_load)}}$
   Time from pulling down $X$ to rising transition at the output.

3. $t_{\text{fall, inv\_feedback}}$
   Time from rising output transition to falling transition of feedback inverter.

4. $t_{\text{pullup}, X}$
   Time from falling transition at gate of precharge PMOS to pullup of node $X$.

5. $t_{\text{fall, inv\_out(output\_load)}}$
   Time from pulling up $X$ to falling transition at the output.

6. $t_{\text{rise, inv\_feedback}}$
   Time from falling output transition to precharge shutoff.
3.4. AWPCMOS CIRCUIT DESIGN

The latency for an AWPCMOS gate is given by

\[ t_{\text{latency}} = t_{\text{pull down, } X} + t_{\text{rise, inv out}}(\text{output load}). \]  \hspace{1cm} (3.9)

Equation (3.9) shows that the latency overhead over domino logic is only the slight load of the weak feedback inverter.

The cycle time is given by

\[ t_{\text{cycle}} = t_{\text{pull down, } X} + t_{\text{rise, inv out}}(\text{output load}) + t_{\text{fall, inv feedback}} + t_{\text{pull up, } X} \]

\[ + t_{\text{fall, inv out}}(\text{output load}) + t_{\text{rise, inv feedback}} \] \hspace{1cm} (3.10)

corresponding to a three-inverter ring oscillator cycle which is six inverter delays. In (3.10) there are four inverter delays, i.e.

\[ t_{\text{rise/fall, inv out}}(\text{output load}) + t_{\text{rise/fall, inv feedback}}. \]

As the output load is in the loop, cycle time is of course impacted by this load. Decoupling the loop from the load by inserting a buffer would, however, increase the latency which is not an option. The remaining terms in (3.10) are the pullup/pulldown delays of the precharge node. The pullup delay \( t_{\text{pullup, } X} \) is less than an inverter delay in a comparable ring oscillator as the feedback inverter has to drive only the single PMOS gate. Furthermore, when the gate is correctly designed so that no crowbar current occurs, the precharge PMOS does not have to fight a NMOS pulldown as is the case in an inverter. The same reasoning as for the precharge PMOS device applies for the pulldown delay \( t_{\text{pull down, } X} \) which entails only NMOS. This is of course nothing but the well-known advantage of dynamic logic, that is gaining speed by presenting less input capacitance than static logic. Finally, the two NMOS series connection is necessary for the logic AND function. This pulldown is thus slower than one NMOS in an inverter. However, as above, the NMOS logic pulldown is still fast as the precharge should have already completed and there is no PMOS to fight.

Altogether the cycle time of an AWPCMOS gate given by (3.10) matches quite accurately the cycle of a three-inverter ring oscillator. The actual frequency depends mainly on the output load presented by interconnect capacitance. As is evident from Figure 15 a frequency as high as 2GHz in a 0.35\( \mu \)m CMOS process can be achieved when the load due to interconnect is low. It is important to note that not overblown power-FETs for speed at any cost but devices with reasonable sizes are used in this example: 4\( \mu \)m wide NMOS logic and PMOS precharge and 8\( \mu \)m/4\( \mu \)m for the output inverter. In AWPCMOS this cycle cannot be shortened further because then a pulse cannot be fully formed. Thus it can be argued that this cycle of six inverter delays is actually the absolute minimum possible in any VLSI system as it will be even harder to distribute such high frequencies over greater distances than from one AWPCMOS gate output to a subsequent gate’s input. Most high-speed VLSI systems get their clock from an on-chip PLL. At the core of the PLL there’s a VCO often realized as a ring oscillator with at least three inverting stages. Even schemes that double-pump the clock with local clock choppers typically incorporate a delay of three inverters which gives a six inverter cycle, e.g. see left side of Figure 4.

It can be expected that any clocked scheme that approaches a six inverter cycle or even tries to go below quickly gets infeasible because of horrendous clock power. This view is also expressed in [Harris99, pp. 9-10] and [H+02]. The 1997 SIA roadmap predicts cycle times of 6 FO4 delays, corresponding to 6GHz in 70nm CMOS for year 2009 and 5 FO4 cycle, 10GHz in 50nm CMOS in 2012, respectively.

\[ ^{4}\text{fanout of 4, delay of an inverter driving 4 similar inverters} \]
The NMOS pulldown part in AWPCMOS logic has no footer device which improves both latency and throughput. It brings with it, however, liabilities regarding cross current. In synchronous domino where the precharge is global and the evaluate ripples, omitting the footer does not create problems during evaluation as all inputs are precharged low and rise monotonically. But during precharge there will be cross current in all gates for the time their inputs are still high, i.e. the time it takes the predecessor gates to complete precharge and have their outputs low. This effect is worst in later stages and thus affects especially long paths.

In AWPCMOS we have the asynchronous version of the same problem: the inputs must be already low when precharge begins. To this end, we compare the output pulse width generated by an AWPCMOS gate and the maximum allowable input pulse width that still guarantees cross current-free operation.

The output pulse width $t_{pw, out}$ measured between $V_{th,NMOS}$ corresponds to arcs 3, 4, and 5 in Figure 15 and is computed as

$$t_{pw, out} = t_{fall, inv\_feedback} + t_{pullup, x} + t_{fall, inv\_out}(output\_load). \quad (3.11)$$

The maximum allowable input pulse width $t_{pw, in}$ at a subsequent gate is the time from the input rising above $V_{th,NMOS}$ to the precharge PMOS gate falling below $V_{DD} - V_{th,P.MOS}$. At this point the input must be below $V_{th,NMOS}$ again. This time corresponds to arcs 1, 2, and 3 in Figure 15 and is computed as

$$t_{pw, in} = t_{pulldown, x} + t_{rise, inv\_out}(output\_load) + t_{fall, inv\_feedback}, \quad (3.12)$$

where $output\_load$ is the output load of the subsequent gate. For no cross current to appear we must have

$$t_{pw, out} \leq t_{pw, in} \quad (3.13)$$

everywhere in the pipeline. Equation (3.13) is called the AWP Stage Condition and holds if and only if

$$t_{pullup, x} + t_{fall, inv\_out}(output\_load) \leq t_{pulldown, x} + t_{rise, inv\_out}(output\_load). \quad (3.14)$$

Note that this inequality is independent from $t_{fall, inv\_feedback}$.

Comparing the cycle time (3.10) and latency (3.9) suggests to skew the output inverter towards a faster rising edge at the cost of a retarded falling edge while keeping the inverter input gate capacitance constant. This would reduce latency while not affecting the throughput as the effects of reduced rising and increased falling transition delays cancel out in the cycle time equation (3.10). Unfortunately, this is not an option due to the stage condition equation (3.14): reducing $t_{rise, inv\_out}$ on the right side forces $t_{fall, inv\_out}$ on the left side of the inequality to be lowered as well jeopardizing the skewing of the output inverter.

Assuming for a moment uniform fanout and interconnect load, i.e.

$$output\_load = output\_load', \quad (3.15)$$

then using a symmetric inverter with $t_{inv\_out} = t_{fall, inv\_out} = t_{rise, inv\_out}$ simplifies the stage condition (3.14) further to

$$t_{pullup, x} \leq t_{pulldown, x}. \quad (3.16)$$

In this ideal case the reduced stage condition in (3.16) shows that in an AWP precharge must not be slower than evaluate. The inherent series connection in the NMOS pulldown
3.4. AWPCMOS CIRCUIT DESIGN

helps to satisfy (3.16) but it shows as well that the PMOS precharge must be sized in the order of the logic NMOS devices due to reduced transconductance.

These observations are of course consistent with the notion that short cycle times lead to narrow pulses necessitating both sharp rising and falling edges of pulses propagating through the pipeline.

The ideal equation (3.16) serves as a starting point for an algorithm which sizes AWPCMOS gates according to practical purposes.

3.4.1 Balancing NMOS Logic Pulldown Paths

In the AND gate of Figure 14 delay variation in the NMOS logic part is not an issue provided that the input pulses are coherent. This is because there is only one possible path in a series structure. This is different in OR structures comprising parallel paths. The logic value of an OR is true when at least one of its inputs is true. However the number of logic paths actually switching influences the gate delay as depicted in Figure 16.

\[ y = a + b + c + d + e \]

![Figure 16. Delay variation in an OR gate due to parallel paths.](image)

The output traces labelled or \( \triangledown \) correspond to \( n \) inputs being one. The delay variation between or \([a]\) and the other cases is significant, i.e. \( >100\,\text{ps} \). The solution here is to serialize OR structures to preclude parallel paths at the cost of increased latency. Short paths are padded to the maximum number of series devices using dummy transistors that are permanently on. This process is aided by the formalism of binary decision diagrams (BDDs [Bryant86]).
BDDs represent boolean functions by directed graphs where the inner nodes are input variables and the leaf nodes are either zero or one. The edges are labelled by either zero or one depending on the value of the variable they start from. Thus a BDD allows boolean function evaluation by starting from the root and working through the graph directed by the value of the input variables all the way down to a leaf node which gives the value. The BDD can be directly mapped to hardware using multiplexer circuits.

Figure 17 shows a full-adder example. For the sum function every of the four possible paths has three series devices assuring equal delays. The carry function, however, has one path with only two series devices. For delay equalization a dummy device — illustrated by the dashed circle — is used sized a little smaller than the other logic devices. The exact value for the gate width can best be determined by analog simulation.

This multiplexer-style logic can nominally be delay balanced but has one important consequence: now even a monotone function like OR needs inverted inputs for the multiplexer control and this necessitates dual-rail design.

### 3.4.2 Single-Rail vs. Dual-Rail

We have already seen examples of static differential pass transistor logic in Section 3.3. The differential nature helped in balancing delays but of course introduces overhead. The same applies for dynamic logic.

Principally, in dynamic logic we can design single-rail and dual-rail circuits. The above example in Figure 14 was a single-rail two-input AND gate.
Conventional single-rail domino logic can only implement monotone, i.e. non-inverting functions. As logic functions are not monotone in general, e.g. XOR, dual-rail domino was invented. Dual-rail domino with its monotone complementary outputs can realize any boolean function. This comes at the cost of increased switching activity and interconnect capacitance as illustrated in Figure 18.

![Figure 18](image)

Eight cycles are shown whereas the static data is updated every rising clock edge. Assuming a 50% probability for switching the output this results in four switching events. The same data sequence is shown for single-rail and dual-rail domino where the precharge low state occurs at clock low and evaluate at clock high. In contrast to the static case where data is valid for the whole cycle the gray periods represent precharge phases which carry no valid data. It can be seen that for single-rail precharged data a sequence of ones is the worst case as this results in toggling between precharge low and evaluate high. The single-rail scheme has ten events. Finally, the dual-rail scheme has a precharge and an evaluate event in every cycle, independent of the data, which results in 16 events. If the interconnect for dual-rail bit pairs is balanced then the current profile of dual-rail domino circuits is nearly constant irrespective of processed data. This property is exploited in cryptographic hardware to provide strength against Differential Power Analysis (DPA).

On the basis of switching activity precharging costs roughly twice the power of static logic and dual-rail costs twice the power of single-rail. However this is pessimistic compared to reality because crowbar and glitch current existing in complementary static logic are not accounted for. Dynamic circuits save power by presenting only about 30% input capacitance compared to static logic. Finally, only dynamic circuits provide the opportunity to work with reduced voltage swings. It is obvious that a good clocking scheme is all important in dynamic logic to keep the precharge phase away from the critical path.

Dual-rail circuits can sometimes share devices between the complementary NMOS logic paths, e.g. in XOR gates. This allows dual-rail circuits without having to double the devices for the NMOS logic part. However, for functions like wide OR gates with \( n \) inputs dual-rail circuits have the disadvantage that while one NMOS path is short (parallel OR with only one series device) the complementary path is long (\( n \) series devices for NOR).

The mentioned restriction of single-rail domino circuits to monotone functions does only apply if all gates receive the evaluate clock simultaneously. If, in contrast, the clock is cascaded and allows every gate to evaluate before the next gate evaluates, then it is possible to realize inverting functions. Very fast circuits result if the logic can take advantage of high-fanin OR structures.
CHAPTER 3. PRINCIPLES OF ASYNCHRONOUS WAVE PIPELINES

In AWPCMOS there is a pulsed version of the monotonicity problem present. Figure 19 shows what happens when an inverter at the output is used to produce an inverting function as depicted by the dashed curve. When \( \pi = 0 \) and \( b = 1 \) are combined at the illustrated series connection to form the logic function \( \pi \cdot b \) then unstable operation results because a rising pulse transition crosses a falling transition where the circles are. While \( \pi \cdot b \) should yield zero, the precharge node \( P \) discharges at least partially, leading probably to an erroneous pulldown and logic one result.

![Figure 19. Violation of the monotonicity rule in AWPCMOS.](image)

The above mentioned delayed clock method is no option because in AWPCMOS there is no clock and therefore no cure for the monotonicity problem other than dual-rail design. This is a disadvantage of the AWP technique that must be accepted.

3.4.3 Single-Rail Design

There are a few cases where the logic function to be built is monotone and single-rail AWPCMOS can be fruitfully applied. A notable example is the carry-lookahead tree in the Brent-Kung adder described in Chapter 4.

While the design of the basic single-rail gates has been shown we have not talked about how to latch the results at the output side. This is more of a problem in a single-rail AWP than in a dual-rail one. It is caused by data bits with zero value being indistinguishable from no data at all signalled by precharge zero values. Thus it is not possible from looking at the data alone whether a new result should latched or the output register not updated making an explicit matched delay line necessary. This is shown in Figure 20.

![Figure 20. Necessity for matched delay line in single-rail AWPCMOS.](image)
3.4. AWPCMOS CIRCUIT DESIGN

Remember that the asynchronous property of the AWP consists in accommodating idle phases where no data has to be processed. This is signalled by the absence of request pulses which causes the pipeline to be held in precharge and the power dropping to leakage only. In the single-rail case the AWP looks like Figure 12. If we neglect the pipelining for a moment this is the bundled data plus delay model well known in asynchronous design. Because this touches the issue of Completion Detection, which is of fundamental importance in asynchronous design, we review completion detectors in the following subsection.

Completion Detectors

In every circuit we have to know when the voltage on wires represents a valid signal, i.e. to know when the computation of that signal has completed. Otherwise wrong values are taken and the circuit does not execute the algorithm for which it was designed. Thus a timing reference is needed. Basically there are two options: either there is a global timing reference which tells you when the data is valid, or the timing information is embedded in the data itself. Of course the global timing reference in the first case is nothing but the clock in synchronous design. The latter case characterizes asynchronous design. Crucial to the performance of asynchronous circuits is an efficient means of realizing such embedded timing.

The only situations where the completion detection overhead may be tolerable are, first, when average-case delay is much less than worst-case delay, and, second, when the completion detector is pipelined. In the first case, when the completion detection latency is added to the logic latency there can still be a net performance improvement in the average case. In the latter situation, the completion detection latency does not impact the system as long as only throughput is important and the completion detector is no cycle time bottleneck. However, situations with significant data-dependent delay are not too common. On the other hand, in a pipeline there can not be much delay variation anyhow as such would reduce the possible cycle time. A completion detector is therefore quite an overkill to compensate for the small delay variations that the pipeline will allow. A typical situation which is different from both mentioned cases is the integer adder in the ALU of a microprocessor. This adder is in the cycle-limiting path of the processor but not pipelined. While there in principle is data-dependent delay in such an adder, they can be built now with under 500ps delay so that a completion detector would have a hard time not to deteriorate the performance.

The following completion detection schemes are known today:

- Single-rail bundled data with matched delay is the simplest and probably the most widely used approach today. It has been used, e.g., in Philips’ asynchronous version of the 80C51 microcontroller [GBP+98a]. That chip also demonstrates the main problem with this approach in that it uses 100% margin in the delay elements effectively halving the performance.

- Delay-insensitive dual-rail or 1-of-4 encoding equipped with a completion detection circuit does not rely on matching delays [Williams01, YBV+98]. The completion detector is best realized as a tree of combinational logic or C-elements. This method, however, incurs a latency penalty of $O(\log n)$ and an area penalty of $O(n)$, with $n$ being the data path width. This is impractical for wide data paths.

- In speculative completion detection [NYB97] one of several fixed delays is selected depending on the input values. The overhead is increased by these multiple delay
CHAPTER 3. PRINCIPLES OF ASYNCHRONOUS WAVE PIPELINES

lines. Furthermore, the effectiveness of the technique hangs on the application-specific abort logic that selects the delay line to be used.

- A natural way to detect when a combinational logic circuit is done would be to measure when the switching activity has ceased. Techniques for current sensing and activity monitoring imply a high cost in area and power [GBK97, BG97].

In summary it can be said that even though optimizations have been made [Cheng98] the completion detection problem has not been solved satisfyingly and this will likely remain the case for the future. The completion detection overhead is the single most important issue barring asynchronous design from entering the mainstream.

Data Converter and Pulse Catcher

When a single-rail AWP design is used the delay line must be very accurately matched to the logic. Whereas in a bundled asynchronous datapath there is a one-sided timing constraint for the delay — it must be no smaller than the worst-case logic delay — the pipelining creates a two-sided timing constraint in the delay line of a single-rail AWP. Any mismatch between logic and delay line translates into more latency and increased cycle time or failure.

An AWP is embedded into a larger system. If we assume that input data is read from a static register and the results should be written again into some static register we need circuits for translating data between the level and pulse domains. This is shown for one data bit in Figure 21.

![Diagram of level to pulse converter and pulse catcher](image)

Figure 21. Single-rail AWP with interface to static environment.

The operation is as follows. Input data is supplied as levels at \( D_{in} \) and the validity of the data is signaled with the rising edge of the request pulse \( req_{in} \). The \( req_{in} \) signal may be considered as a local clock. The levels are then converted by launching a pulse into the combinational logic when both \( req_{in} \) is asserted and \( D_{in} = 1 \). For \( D_{in} = 0 \) no pulse is generated.

Because the delay is matched to the logic, data and request pulses arrive together at the pulse to level converter or pulse catcher. The coherency of data and request depends on the accuracy of the delay matching which is always less than perfect. In a single-rail AWP design it may be the case that the inaccuracy of the delay element dominates the logic path delay variations inside the logic and thus determines the cycle time. The reset state
3.4. AWPCMOS CIRCUIT DESIGN

The preceding discussion suggests that most of the time dual-rail design is mandatory for the AWP. The primary reasons are a function not being monotone or needing inverted inputs for delay balancing as discussed in Sections 3.4.1 and 3.4.2. If that does not apply then at least the timing constraints described in the last section are not desirable.

Figure 23 shows schematics and waveforms of a XOR/XNOR dual-rail AWPCMOS gate. It can be seen that two devices in the NMOS logic part can be shared.
In a similar fashion, AND/NAND and OR/NOR dual-rail AWPCMOS gates can be constructed. Figure 24 shows the respective complementary NMOS pulldowns only.
3.4. AWPCMOS CIRCUIT DESIGN

![Logical Gates Diagram]

**Figure 24.** (N)AND and (N)OR pulldowns for dual-rail AWPCMOS logic.

**Half-Keeper vs. Cross-Coupled Keeper**

In precharged circuits the charge stored on the dynamic node is subject to leakage and charge sharing mechanisms causing unwanted discharge. Therefore PFET keepers are used to hold the charge on the precharge node. Two general configurations are commonly in use as shown in Figure 25.

![Keeper Design Options Diagram]

**Figure 25.** Keeper design options: (a) half-latch  (b) cross-coupled.

The half-latch configuration just doubles single-rail domino and controls the keeper gate by the inverted output. This has the disadvantage that a proper evaluate has to override the keeper resulting in increased delay and power consumption. Furthermore the keeper “does not know” which discharge event is due to evaluate and which is due to charge sharing. This makes it difficult to correctly size the keeper. On the one hand, it must be as small as possible because of the mentioned drawbacks; on the other hand, it must be able to counteract worst-case charge sharing. Especially in deep NMOS stacks, the difference between charge sharing — when all but the last device to ground are on — and evaluate — when there is a path to ground — is not too big which makes it difficult to maintain noise margin.

The cross-coupled configuration is better off. It derives the inverted output to control the PFET gate from the precharge node of the complementary pulldown. This eliminates the half-latch hysteresis — after precharge both keeper devices are off — at the cost of
increased precharge node capacitance due to keeper gates. In \[ \text{[BCD+98, 3.4.2, p. 113]} \] it is shown, however, that cross-coupled keepers yield a net performance gain over half-latch keepers. Charge sharing is mitigated already with smaller keepers. If one path properly evaluates and the complementary path suffers from charge sharing, the evaluate pulldown will win the race and fully activates the keeper for the complementary path.

However input delay variations can cause problems if the keeper is activated too late. Consider the NAND tree in Figure 24. If \( \bar{b} \) rises well before \( \bar{\alpha} \) then charge sharing caused by the rising \( \bar{b} \) can be desastrous because the counteracting of the keeper activated by \( \bar{\alpha} \) may be too late.

There are two further differences between the cross-coupled and the half keeper configurations. The first one can materialize even in synchronous systems while the second is more subtle and specific to the AWP.

- As the cross-coupled keepers are off after precharge has completed they do not replace charge lost by the dynamic nodes due to leakage. Depending on the clocking scheme the precharge PFETs may be already disabled effectively leaving the dynamic nodes floating. These nodes are then vulnerable to coupled noise and subject to leakage. Leakage lets the voltages drop on both precharge nodes with the rate being dependent on the node capacitance. The node with smaller capacitance will reach \( V_{DD} - V_{th,P\text{MOS}} \) first thus activating the other side’s keeper while continuously leaking and eventually triggering the output inverter, rf. Figure 26. Altogether this leads to logic failure. Often in synchronous clocking schemes there is a strict toggle between precharge and evaluate so that the dynamic node is never left floating and leakage is not an issue. If this is not the case then standby time is limited by charge retention. In an AWP environment the feedback network disables the precharge PFET after precharge which limits the possible standby time. The voltage drop in this case must be small as it causes delay variation: pulldown delay is shortened if the voltage has been reduced by leakage in the first place.

- In an AWP even in burst-mode the odds may be that always the same side in the complementary NMOS pulldown network evaluates, rf. node 1 in Figure 27. Thus the other side, node 2 in Figure 27 will never precharge; the precharge node, however, is refreshed by the keeper activated with every pulldown of node 2. So at first everything looks alright. What really happens, though, is that the falling edge of node 1 couples into node 2 via the gate-drain overlap capacitance of the keeper connected to node 2. The lost charge is only partially replaced as the keeper has not enough time to act. After a few cycles node 2 toggles between two intermediate voltages less than \( V_{DD} \). As above, this shortens the pulldown time compared to starting from \( V_{DD} \) and leads to delay variation.

While the cross-coupled keeper is the preferred configuration in synchronous design due to reduced latency and power it is not attractive for AWPs due to the two mentioned problems.

The fundamental difference between a clocked domino circuit and an AWP is that the precharge is not refreshed by the clock in every cycle. This changes the job description for the keeper: while in the first case it has to work inside of the clock cycle only in the AWP there is no concept of clock cycle. Therefore the keeper must be able to act an arbitrary number of times as the next precharge event depends on input data.
3.4. AWPCMOS CIRCUIT DESIGN

Figure 26. Leaking of cross-coupled keeper.

Figure 27. Dynamic behaviour of precharge nodes with cross-coupled keepers.
CHAPTER 3. PRINCIPLES OF ASYNCHRONOUS WAVE PIPELINES

Latchin... Dual-Rail AWPs

The general philosophy of the AWP is to reduce as much as possible global clock infrastructure leading to increased power consumption and decreased performance. We have seen that dual-rail circuits often will be the only choice for AWP and according to 3.4.2 those are quite power-hungry. The high switching activity of a dual-rail datapath can, however, be advantageously used to realize an efficient latching scheme without any external clocks.

Remember that in asynchronous design dual-rail circuits are popular because they allow for a real completion detection in a delay-insensitive manner without relying on delay matching. The cost of this approach are the latency and area of the completion detection tree.

In an AWP the problem of completion detection is not the primary issue. As the AWP is targeted at high-frequency fine-grained pipelines the spread of individual bits of a data word is bounded by a cycle time in the sub-nanosecond region. Even more, as we have seen in the discussion of the single-rail AWP system, the pulsed operation necessitates very well balanced paths at the pulse catcher for correct latching, i.e. less than a fourth of the cycle time. Put another way, in the AWP the paths are necessarily tightly controlled so that there’s nothing left to detect for a completion detector. An arbitrary bit of the datapath with some small added margin would well serve as ready signal.

The real problem in high-speed circuits using wide data paths lies somewhere else, namely to have a latch enable signal not only with the right timing but sufficient drive as well. In asynchronous circuits the latch enable signal is not fed by a strong global clock but must be generated somehow locally. Latching wide datapaths like 64 bits needs considerably drive strength that cannot be built up instantly but needs a pre-amplifier or even a buffer chain which will need some time for their operation. Then it becomes difficult again to keep data and control signals coherent as the delays of logic circuitry using modest device sizes and tapered buffers using big devices do not track well. If one tries to solve this problem by including the latch enable amplifier in the controller cycle one pays with performance loss.

In a dual-rail AWP there’s an elegant fashion to derive a ready signal with correct timing and drive proportional to the width of the datapath. If the datapath is \( n \) logic bits wide the combinational output has \( 2 \cdot n \) signals. We know that in every of the \( n \) dual-rail bit pairs exactly one bit is switching every cycle, that is in total \( n \) bits are switching. We combine these \( n \) bits into one coherent signal. The resulting \( \text{reqout} \) inherits the timing of the \( n \) delay-balanced bits and their drive strengths are added resulting in a drive proportional to \( n \). This process of “combining” the \( 2n \) signals must of course be very fast. This is accomplished by an AWPCMOS wide OR gate.

The concept of a complete dual-rail AWP with static interface is shown in Figure 28. Analogous to Figure 21 we assume static data that is converted to pulses using a local request signal and launched into the AWP combinational logic. The dual-rail version of the level-to-pulse converter uses an inverter and is called dual-rail pulser. Note that only here we can use a static inverter in connection with pulsed logic as the entering data is static, i.e. level-based. Only then is the problem of Figure 19 not present. We will meet the dual-rail pulser again in the Crypto chip application in 5.2.3.

For storing the results we assume an SRAM cell-based output register. Unlike the single-rail case not even a local clock is needed for writing the register. In idle mode both output bits stay low and the SRAM cell shown in the middle inset remains unchanged. When the pipeline delivers data at the output either one of \textit{pulse} or \textit{pulse} will go high and flip the SRAM cell accordingly.
3.4. AWPCMOS CIRCUIT DESIGN

All pulse and pulse signals are further coupled to the gate of a NMOS pulldown. The drains of all these 2n pulldowns are connected together to a common node that physically stretches over the width of the datapath. This node is the precharge node of the wide OR shown in the right inset and is pulled down by n of the 2n pulldowns connected to it. The total effective width of the n active pulldowns triggers a big output inverter that fires off the output request. Thereafter the wide OR is reset in the usual manner.
Figure 28. Latching in dual-rail AWPCMOS.
3.4. AWPCMOS Circuit Design

Single-Rail to Dual-Rail Conversion

Sometimes it is necessary to convert a single-rail signal into dual-rail. A Brent-Kung carry lookahead adder like the one described in chapter 4 is such an example: the carry lookahead tree is monotonic and thus single-rail to save power and area. The final XOR for sum generation however needs dual-rail inputs. The conversion can be performed as shown in Figure 29.

![Figure 29. Single-rail to dual-rail conversion in AWPCMOS.](image)

The converter is basically a combination of the single-rail pulse catcher of Figure 21 and the dual-rail pulser shown in Figure 28. Single-rail data \( s \) accompanied by request \( \text{req} \) enters the converter. In the low phase of \( \text{req} \) internal node \( P \) is precharged to high; foot devices \( M_1 \) and \( M_2 \) are both off. When \( \text{req} \) rises \( M_1 \) and, after the inverter pair delay of \( I_{\text{inv}_1} \) and \( I_{\text{inv}_2} \), \( M_2 \) are turned on. If \( s = 0 \) the \( P \) stays high which results in a pulse being fired off \( \overline{d} \) while \( d \) remains quiet. Input \( s = 1 \) causes \( P \) to fall. The inverters allow for \( P \) to fall before \( M_2 \) turns on. Subsequently a pulse is fired off \( d \) while this time \( \overline{d} \) stays low.

3.4.5 Low Power Design

The AWP concept saves a lot on power as it eliminates distribution of a high speed clock signal and sequential elements like latches and flipflops in the pipeline. This will be more quantitatively analyzed in section 3.6. However the dual-rail AWPCMOS circuit style is not in itself low-power. Thus it is desirable to lower the energy requirements while not impairing the AWP characteristics. Figure 30 shows how this can be achieved by lowering all swings in AWPCMOS logic. The basic idea is just to reduce \( V_{DD} \) by using a cascode NMOS device to \( V_{DD^*} = V_{DD} - V_{th,NMOS} \).

When lowering voltages it is important to check that the electrical situation is still healthy. Do we have sufficient gate overdrive where we want to switch on devices fast? Are devices safely switched off without causing cross current?

Consider first the standby position when the gate is precharged and waiting for data. Then the precharge node is at \( V_{DD^*} \). This is also the positive supply of the output inverter; therefore the PMOS \( V_{gs} \) is zero and the device is fully off. For the inverter NMOS we have overdrive of \( V_{gs} - V_{th,NMOS} = V_{DD^*} - V_{th,NMOS} = V_{DD} - 2 \cdot V_{th,NMOS} \). The NMOS pulldown should be fully on. To ensure enough overdrive this requires \( V_{DD} \gg 2 \cdot V_{th,NMOS} \). The gate output is thus zero. This is important as the switching threshold for the successor gate is only \( V_{th,NMOS} \). The keeper PMOS is correctly activated having
overdrive of $V_{gs} - V_{th,PMOS} = -V_{DD} + V_{th,NMOS} - V_{th,PMOS}$ \(^2\). Finally, the precharge PMOS is shut off.

During evaluation the precharge node is pulled down to zero causing the output inverter to rise to $V_{DD^*}$. With the same reasoning as above this is sufficient to trigger downstream logic gates. The keeper is disabled by $V_{gs} = 0$. After the feedback delay the precharge PMOS gets activated by having $V_{gs} = -V_{DD^*}$.

Comparison of the waveforms of Figure 23 and Figure 30 shows that in the low power version the pulse width is increased due to the additional feedback inverter. However, latency and cycle time are not impacted while saving nearly 50% of the energy.

\(^2\)Note that PMOS threshold $V_{th,PMOS}$ and overdrive are negative.
3.4. AWPCMOS CIRCUIT DESIGN

Power = 0.157μA/MHz/gate@ 3.3V

Figure 30. AWPCMOS low power XOR gate simulation.
3.4.6 Pulse-Width Control

There exists an asymmetry in AWPCMOS between the low and high phases of a pulse. While the duration of the former is data-dependent the latter is fixed by the feedback delay of the AWPCMOS gate. This is a consequence of the autonomous operation of the logic with the pulse not being gated by a clock or handshake signal and sometimes called embedded timing. The disadvantage of this property is that in case of larger than anticipated delay variation lowering the frequency to allow more overlap between pulses will not help. This is in contrast to both synchronous and handshake-style asynchronous systems where a setup violation can be salvaged by reducing frequency. Even if the system doesn’t quite meet its spec sacrificing a few Megahertz may be better than a chip not running at all! The latter is symptomatic for hold violations and chip designer’s nightmare as it will result in total failure. This contributes to the aggressive all-or-nothing mentality of the AWP; every path is critical and must be balanced according to the fixed pulse width determined by the highest possible frequency.

It is the author’s belief that every other scheme that can vary pulse width according to data rate has inevitably higher cycle time and latency. Two possible ways for increased robustness, quite different in nature, are briefly discussed in the following.

Clock-Data Precharging and Pseudo-Clocked Domino

The first approach is to let somehow the data itself help in precharging or resetting the gate. There are two techniques in the literature to improve conventional domino in this direction, primarily to take load from the clock and speed up the gate. The first is the clock-and-data precharged dynamic (CDPD) technique [YSL93] as illustrated in Figure 31.

![Diagram of CDPD and Pseudo-Clocked Domino](image)

**Figure 31.** Example from [YSL93]:
(a) Conventional domino AND   (b) CDPD version with High/Low stage.

A conventional domino AND is shown at the left side. Inputs labelled PH and PL are meant to be precharged high and low, respectively. The H/L stage on the right replaces the contents of the box. The H/L stages takes precharged high inputs and has its output precharged low by the NMOS device. Only if both inputs to the AND are true — equivalent to the PMOS gates being zero — will the pullup be activated giving one. The clock load, area and latency are reduced as a result of this transformation representing a step back to static logic. It is known that any inverting static gate after the precharge stage will result in a well-formed domino structure. While the commonly used inverter is the simplest possible inverting static, the H/L stage is just a more complex inverting static gate, almost a complementary NOR. Other possible combinations are described in the paper. However, if one takes this too far, one will start losing the advantages of dynamic logic. After all,
the H/L stage does logic with slow series PMOS. This is only a net performance gain when
two inverters can be eliminated, but will turn into a loss when there are more static stages
used sequentially.

The second idea in this direction is pseudo-clocked domino [BCD+98, pp. 115–116].

Figure 32 (a) shows a dual-rail AND/NAND gate.

![Figure 32. AND/NAND gate in pseudo-clocked domino logic: (a) dual-rail (b) single-rail.](image)

As the inputs are all precharged to low this turns on the four PMOS devices precharging
both output rails to zero. When data gets asserted thereafter precharge ends and not both
a and ̅a can be simultaneously on. This disables both pullups and allows pulldown in the
asserted rail to take place.

This technique is more familiar to AWPCMOS than CDPD. Yet it confirms the above
statement that trying to control the pulse width by the data inevitably slows down the
pipeline. This is because the reset capacitance is present in the forward path. It is evident
in Figure 32 that the PMOS structure loads down the preceding gate. This is exacerbated
by the fact that this series structure must be fast enough to complete precharge in time
resulting in big PMOS sizes. Furthermore, if a = b = 1 and a is lagging b this leads to cross
current.

Figure 32 (b) shows a single-rail version of the AND gate. It is of not much help,
however, because it is the only single-rail gate possible in this way. Only in the AND gate
a single PMOS works correct.

**Controlling Pulse Width by Analog Bias**

A quite different technique applies an analog bias voltage to modulate the pulse width.
This bias may be tuned and set at testing time on individual chip basis. Figure 33 shows
the concept. Let’s recall here for convenience the AWPCMOS cycle time relation equation
(3.10):

\[
t_{\text{cycle}} = t_{\text{pulldown,X}} + t_{\text{rise, inv}_\text{out}(\text{output_load})} + t_{\text{fall, inv}_\text{feedback}} +
\]

\[
t_{\text{pulldown,X}} + t_{\text{fall, inv}_\text{out}(\text{output_load})} + t_{\text{rise, inv}_\text{feedback}}
\]

The equation shows that in order to reduce the throughput without adversely affecting
latency both the rising or falling transitions of the feedback inverter could be delayed. By
doing this we will not run into problems with cross current as, fortunately, the AWP stage condition equation (3.14) is independent from the feedback inverter. In Figure 33 only the falling transition is delayed by inserting a NMOS in the path to ground with the gate set to an analog bias voltage.

![Diagram of a circuit](image)

**Figure 33.** Increasing pulse width by lowering bias.

We assume the AWP was designed for 1.5GHz operation. This is supported by the narrow pulses in the first trace on the right side. Suppose further that the design is not well enough balanced to support 1.5GHz. If the pulse width were fixed than the system would probably not run at all, even at lower frequencies. Only if both the frequency is decreased and the pulse width is increased sufficiently will the pipeline become functional. The pulses shown are already spaced for reduced rate operation at 1GHz. Now by gradually lowering \( V_{pw} \) the falling inverter transition gets delayed (second trace) resulting in the desired behaviour, namely wider output pulses (third trace).

The disadvantage of this technique is the necessity to globally distribute the bias. This analog signal must be low-impedance and shielded to prevent coupling into the bias to influence the pulse width. Finally, the bias must be precisely generated and the influence on the pulse width is quite nonlinear.

### 3.5 Analysis of AWPCMOS Delay

It was pointed out in section 3.1 and manifested in equation (3.5) that an extremely accurate balancing of logic path delays is imperative for achieving short cycle times. In the following the detractors from ideal delay equalization are categorized and analysed. Sources of delay variation in gates and interconnect can be attributed to either data-dependent delays, PVT variation or various noise mechanisms.
3.5. ANALYSIS OF AWPCMOS DELAY

It is important to note that the gate and path delay balancing needs to take place in space and time. Balancing in space refers to controlling delays when a logic wave propagates through the spatial circuit, i.e. balancing gate delays and interconnect capacitances. Balancing in time means that gates must not possess history, i.e. the same input pattern yields different delays depending on previous patterns.

Of particular importance is the concept of accumulation of delay variation. If accumulation occurs in time the wave pipeline quickly becomes infeasible whereas accumulation in space sets an upper limit on the logic depth.

Let's assume we have some combinational circuit realized in AWPCMOS and data waves are launched into the pipeline synchronously, i.e. all data bits are perfectly aligned. Which mechanisms are introducing delay variations during propagation through the logic?

The perfectly aligned data wave arrives at transistor gates. If the load presented to the driving signal by the gate capacitance are not matched delay variation occurs. Data-dependent gate capacitance introduces delay variation even when the nominal gate capacitances are perfectly matched. This effect is due to the capacitance presented by a MOSFET gate depending on the source and drain voltages of the device.

The gates themselves exhibit propagation delays that are not exactly the same for all possible input combinations. What matters is the transition from one input combination to another which reveals dependence on input history.

The results are then travelling from the gate outputs over the interconnect to the subsequent gate inputs. The interconnect capacitance must be nominally matched for balanced delays and transition times. Cross-coupling between metal lines introduces delay variation. However, this issue is relaxed for AWPCMOS due to the dual-rail precharged operation: only one wire in a dual-rail pair will switch and the monotonic property ensures that all of them are switching in the same direction. This avoids the worst-case cross-coupling event occurring in level-base logic when a rising transition fights a falling one.

The foregoing detractors can be countered by careful device sizing and layout. In addition to these deterministic mechanisms statistic PVT variations are a major source of delay variation. Temperature and voltage fluctuations are hard to predict but it appears reasonable to assume more or less homogenous profiles in the pipeline. If all devices are impacted in the same way cancellation will take place in the delay variation.

The following analysis of delay in AWPCMOS circuits emphasizes the two major contributors to delay variation: data-dependent gate delay and device process variation.

3.5.1 Data-Dependent Gate Delay

Let's assume for a moment that we have ideal conditions: constant temperature and voltage and a perfect process resulting in uniform device and interconnect characteristics. Let's further assume that all logic paths are balanced to have the same depth and their load capacitances matched. The remaining delay variation is then due to data dependency. The focus on data-dependent delay alone serves to yield an upper bound to the minimum achievable delay variation. The nonidealties of any real system will cause any wave pipelined system to operate with higher delay variation.

Data-dependent delay is one of the main obstacles for wave pipelining. It refers to small variations in the delay of a gate when presented with different inputs. This variation may be small for a single gate but by accumulating is quickly in the order of the cycle time and makes wave pipelining infeasible. For static CMOS it is shown in [Klase94, p. 111] that the NAND2 delay with both inputs rising simultaneously is nearly twice the delay when
both inputs fall.
In the following a comparison is made between AWPCMOS and seven other logic families that were proposed for wave pipelining. The test setup consists of an AND/NAND gate driving two similar gates with the inputs coming from buffered spice sources. Eye diagrams are simulated by stimulating the two inputs \( a, b \) of the AND/NAND with three consecutive patterns in all possible combinations: 

\[
a^t b^t \rightarrow a^{t+1} b^{t+1} \rightarrow a^{t+2} b^{t+2}.
\]

The simulations were performed with Spectre on schematics in a 130nm low-Vt CMOS process under nominal conditions. The input pattern cycle time is 140ps corresponding to a frequency of 7.14GHz. Three traces are shown in each of Figure 34 till Figure 41: the upper one is the input eye which is the overlay of all \( 2^{2 \times 3} = 64 \) input patterns at the inputs of the gate (output of buffers). AWPCMOS is the only dynamic logic family in this benchmark and needs pulses as inputs whereas the others accept levels.

The middle trace is the overlay of the 64 AND/NAND gate outputs. The lower trace is the overlay of the 128 AND/NAND gate outputs driven by the first one. The deltas corresponding to delay variation are printed in small letters at the bottom line of the traces and can be found as \( \Delta_0, \Delta_1, \Delta_2 \) in Table 1. Given as well are individual gate delays and device counts. The average current refers to the whole test circuit comprising \( 3 \times 64 = 192 \) AND/NAND gates and \( 4 \times 64 = 256 \) buffers (due to dual-rail inputs).

<table>
<thead>
<tr>
<th>Logic style</th>
<th>( \Delta_0 ) [( \mu \text{s} )]</th>
<th>( \Delta_1 ) [( \mu \text{s} )]</th>
<th>( \Delta_2 ) [( \mu \text{s} )]</th>
<th>Latency [( \mu \text{s} )]</th>
<th>( I_{\text{avg}} ) [mA]</th>
<th># Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWPCMOS</td>
<td>0.99</td>
<td>8.47</td>
<td>9.43</td>
<td>37.23</td>
<td>23.67</td>
<td>17</td>
</tr>
<tr>
<td>static CMOS</td>
<td>9.53</td>
<td>43.60</td>
<td>45.81</td>
<td>56.05</td>
<td>4.20</td>
<td>6</td>
</tr>
<tr>
<td>biased CMOS</td>
<td>6.47</td>
<td>33.05</td>
<td>46.45</td>
<td>42.86</td>
<td>10.69</td>
<td>7</td>
</tr>
<tr>
<td>balanced CMOS</td>
<td>9.75</td>
<td>49.52</td>
<td>30.60</td>
<td>64.65</td>
<td>4.47</td>
<td>7</td>
</tr>
<tr>
<td>DRSCMOS</td>
<td>3.98</td>
<td>55.07</td>
<td>117.95</td>
<td>67.79</td>
<td>5.09</td>
<td>16</td>
</tr>
<tr>
<td>CPL</td>
<td>9.36</td>
<td>32.29</td>
<td>72.44</td>
<td>36.02</td>
<td>8.41</td>
<td>8</td>
</tr>
<tr>
<td>DPL</td>
<td>55.86</td>
<td>69.38</td>
<td>103.85</td>
<td>50.95</td>
<td>5.58</td>
<td>8</td>
</tr>
<tr>
<td>WTGL</td>
<td>19.64</td>
<td>36.65</td>
<td>65.88</td>
<td>45.23</td>
<td>7.69</td>
<td>10</td>
</tr>
</tbody>
</table>

**Table 1.** Comparison of logic styles proposed for wave pipelining.

As is obvious from Table 1 AWPCMOS has the best control over gate delays and the smallest latency. This comes at the price of increased power consumption and device count. The traces show that all static logic families have difficulty with such a small cycle time. The pass transistor examples show heavily distorted input eyes, mainly due to sneak paths "fighting" against the input.

The remaining delay differences in AWPCMOS are due to charge history on the stacked nodes and could be reduced further by precharging these nodes.