Mälardalen University Press Dissertations
No. 23

# Implementation of Digit-Serial LDI/LDD Allpass Filters

**Krister Landernäs**

January 2006

MÄLARDALENS HÖGSKOLA

Department of Computer Science and Electronics
Mälardalen University
Västerås, Sweden

Copyright ©Krister Landernäs, 2006
ISSN:1651–4238
ISBN:91–85485–07–1
Printed by Arkitektkopia, Västerås, Sweden
Distribution: Mälardalen University Press

# Abstract

In this thesis, digit-serial implementation of recursive digital filters is considered. The theories presented can be applied to any recursive digital filter, and in this thesis we study the lossless discrete integrator (LDI) allpass filter. A brief introduction regarding suppression of limit cycles at finite wordlength conditions is given, and an extended stability region, where the second-order LDI allpass filter is free from quantization limit cycles, is presented.

The realization of digit-serial processing elements, i.e., digit-serial adders and multipliers, is studied. A new digit-serial hybrid adder (DSHA) is presented. The adder can be pipelined to the bit level with a short arithmetic critical path, which makes it well suited when implementing high-throughput recursive digital filters.

Two digit-serial multipliers which can be pipelined to the bit level are considered. It is concluded that a digit-serial/parallel multiplier based on shift-accumulation (DSAAM) is a good candidate when implementing recursive digital systems, mainly due to low latency. Furthermore, our study shows that low latency will lead to higher throughput and lower power consumption.

Scheduling of recursive digit-serial algorithms is studied. It is concluded that implementation issues such as latency and arithmetic critical path are usually required before scheduling considerations can be made. Cyclic scheduling using digit-serial arithmetics is also considered. It is shown that digit-serial cyclic scheduling is very attractive for high-throughput implementations.

# Acknowledgments

First of all, I would like to thank Dr. Johnny Holmberg, who always has taken the time to discuss my work. Our daily discussions on various topics, many of them having nothing to do with research, have made all the difference. I am also grateful to my supervisor Prof. Lennart Harnefors for giving me the opportunity to do this work and for supporting me in my research. I also like to thank my colleagues at the Department of Computer Science and Electronics for their support and daily discussions. I would also like to express my gratitude to Prof. Mark Vesterbacka, Linköping University, for his enthusiasm and interest in my research.

Finally, I would like to thank my family for their support and encouragement.

Västerås a clear day in November 2005

Krister Landernäs

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

In the last decade there has been a significant increase in the usage of battery-powered portable devices. Today, mobile telephones, MP3 players and laptop computers are common products. Many of these products also have an increasing number of functions, thus, requiring higher complexity. As a result, power consumption has become an important aspect when implementing digital systems intended for battery-powered applications. Low power consumption is also of interest in many products that are not battery powered. The reason for this is that a high power dissipation will lead to increased chip temperature. This heat will shorten the circuit life time and increase the risk of malfunction. Much time and effort is spent on integrating cooling devices in electronic systems to get rid of excess heat.

Digital signal processing is common in many of the devices described above, for example mobile telephones. It is, therefore, important to study how low-power implementation of digital signal processing algorithms can be achieved. Careful considerations, concerning for example arithmetics must be made when realizing systems in order to minimize power dissipation. A common rule of thumb is that low hardware complexity is likely to render a system with lower power consumption than it's more complex counterpart, since the capacitive load is reduced. Finding a relation between hardware complexity and power consumption is a non-trivial task. The switching activity of the circuit is an important parameter to consider when studying power dissipation. Unfortunately the relationship between switching activity and hardware complexity is difficult to study without implementing the circuit and performing simulations.

The throughput requirement of the signal processing depends on the application. An audio signal will typically require much lower processing rates than a video signal. In most signal processing cases, however, there is no advantage in performing the computation faster than required. This will only cause the processing elements to wait until further processing is required. To this end, an efficient implementation must meet throughput requirements while exhibiting low power consumption. Naturally, a small hardware solution is preferable since it reduces the manufacturing cost for the system.

In this thesis we study implementation of high-speed and low-power digital filters. We particularly study power/speed characteristics for digit-serial filter implementations. Digit-serial computation offers a higher throughput than its corresponding bit-serial realization without the overhead obtained in a bit-parallel solution. This makes digit-serial implementation interesting in moderate-speed low-power applications. The main motivation for using digit-serial arithmetics in low-power designs is that it requires fewer wires and less complex processing elements compared to the corresponding bit-parallel implementation. A digit-serial design approach allows the designer to find a trade off between area, speed, and power for the application under consideration.

## 1.2 Digital Filters

There are several reasons why digital filters have become more common in electronic systems over the years. Like many digital systems today, digital filters are often implemented in a computer using a high-level programming language. This results in a short development time and makes them flexible and highly adaptable, since changing the filter characteristics simply implies changing some variables in the code. Analog filters on the other hand are implemented using analog components, such as inductors and capacitors, which must be carefully tuned. This makes analog filters harder to develop and modify. Another advantage with digital design is that the characteristics of digital components do not change over time. Digital systems are also unaffected by temperature variations. Advances in CMOS processes have resulted in higher packing density and lower threshold voltages, leading to a considerable decrease in power consumption, which further explains the increased interest in digital filters.

Today, frequency-selective digital filters are important and common components in modern communication systems. Like their analog counterparts, digital filters are used to suppress unwanted frequency components. A linear, time-invariant and causal filter can be described by a [illegible]

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{l=0}^{M} b_l u(n-l), \quad N \geq M. \tag{1.1}$$

By transforming (1.1) with the $z$-transform [39] we can express it as a [illegible]

$$\frac{Y(z)}{U(z)} = \frac{\sum_{l=0}^{M} b_l z^{-l}}{1 + \sum_{k=1}^{N} a_k z^{-k}} = \frac{B(z)}{A(z)} = H(z). \tag{1.2}$$

The frequency function can be obtained by substituting $z = e^{j\Omega}$ in (1.2), where $\Omega$ is the normalized frequency

$$\Omega = 2\pi \frac{f}{f_\mathrm{s}}, \tag{1.3}$$

and where $f_\mathrm{s}$ is the sample frequency. The frequency specification of a filter is often described using cut-off frequency $\Omega_p$, maximum allowed passband ripple $r_\mathrm{p}$ and maximum allowed stopband ripple $r_\mathrm{s}$.

Digital filters can also be described using a [illegible] [52]

$$\begin{aligned} x(n+1) &= Ax(n) + Bu(n) && (1.4)\\ y(n) &= Cx(n) + Du(n), && (1.5) \end{aligned}$$

where $x(n)$ is the $N$-dimensional state-vector and $A$,$B$,$C$, and $D$ are referred to as the [illegible]. In this thesis only single-variable signals are considered, which implies that $B$, $C$, and $D$ are of dimensions $N \times 1$, $1 \times N$, and $1 \times 1$, respectively.

We can derive a transfer function from the state-space expression as

$$H(z) = C(zI - A)^{-1}B + D. \tag{1.6}$$

The transfer function is a mathematical description of a digital filter. However, it does not give any information of how the filter can be implemented. In fact, for a given transfer function there exists an infinite number of possible digital filter structures. When visualizing a filter structure, a [illegible] (SFG) is commonly used [52]. The SFG consists of nodes and branches.

The function described by (1.1) is a recursive function, since the computation requires the value of former output samples. Since the impulse response of the filter described in (1.1) is infinite, these filters are known as [illegible] (IIR) filters. A well-known IIR filter structure is the direct-form filter structure. In Fig. 1.1, the SFG for an $N$th order IIR filter is shown.

In the case where $a_k = 0$ for $1 \leq k \leq N$ the function described by (1.1) is a [illegible] (FIR) filter. FIR filter structures are, although exceptions exist, non-recursive [39]. The SFG for a typical FIR digital filter structure is shown in Fig. 1.2.

The recursive nature of the IIR filter can cause these filters to become unstable. It is therefore necessary to perform stability analyses when designing IIR digital filters, especially at finite wordlength conditions, see Section 1.4. This is not the case for FIR filter: they cannot become unstable. FIR filters can also be designed with exact linear phase. The main drawback of FIR filters is that they require higher filter orders than IIR filters to achieve a certain filter specification. The higher filter order makes FIR filters larger to implement in hardware than the corresponding IIR filters. Take for example the case where a filter with $\Omega_\mathrm{p} = 0.01$, $r_\mathrm{p} = 0.3$ dB, and $r_\mathrm{s} = 40$ dB, is to be designed. In the FIR filter case the required

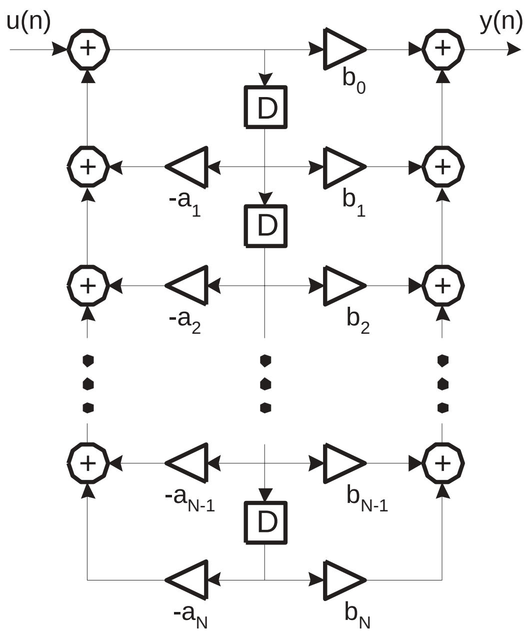

Figure 1.1: $N$th order digital IIR filter structure.

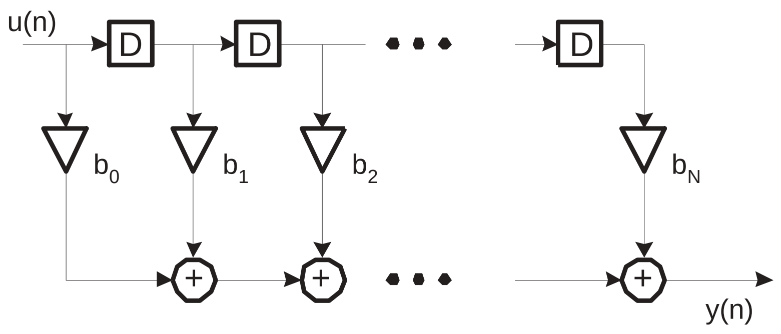

Figure 1.2: $N$th order digital FIR filter structure.

filter order is 112, if the Remez algorithm [27] is used. The corresponding IIR filter order is 7 for a Butterworth filter [27] and even lower for Chebyshev and elliptic filters.

## 1.2.1 Digital Lattice Filters

IIR transfer functions can be realized using two parallel-connected allpass filters [39], provided that conditions, considered below are met. These filters are commonly known as [illegible] *fi*[illegible] [39]. When designing digital lattice filters the separation into two allpass filters is made according to [15]. The transfer function for a digital lattice filter can be expressed as

$$H(z) = \frac{B(z)}{A(z)} = \frac{1}{2}\left[H_1(z) \pm H_2(z)\right], \tag{1.7}$$

where $H_1(z)$ and $H_2(z)$ are allpass filter transfer functions. We can re-express (1.7) as

$$H(z) = \frac{1}{2}\left(\frac{A_0(z^{-1})}{A_0(z)}z^M \pm \frac{A_1(z^{-1})}{A_1(z)}z^P\right), \tag{1.8}$$

where $M$ and $P$ are the filter orders for the allpass filters. A necessary condition for (1.7) is that $B(z)$ must be either a symmetric or antisymmetric function [39]. This implies that digital lattice filters can realize odd-order elliptic, Butterworth, or Chebyshev lowpass/highpass frequency functions, and two times odd-order $(6, 10, 14, ...)$ bandpass and bandstop filters. A typical digital lattice filter structure is shown in Fig. 1.3.

Digital lattice filters have several properties which make them well suited for implementation. First, digital lattice filters exhibit the power-complementary property [39]. This implies that

$$\left|\frac{1}{2}\left[H_1\left(e^{j\Omega}\right) + H_2\left(e^{j\Omega}\right)\right]\right|^2 + \left|\frac{1}{2}\left[H_1\left(e^{j\Omega}\right) - H_2\left(e^{j\Omega}\right)\right]\right|^2 = 1. \tag{1.9}$$

The power-complementary property of digital lattice filters yields that they typically will have low passband sensitivity [52]. As a result, they can be implemented with fewer bits in the filter coefficients, reducing the latency, see Section 1.3, and the power consumption of the filter [52]. Another advantage when implementing digital lattice filters is that they can be realized with a canonical number of multipliers and delay elements. An $N$th order digital lattice filter can therefore be realized with $N$ multipliers, whereas a direct-form IIR filter requires $2N + 1$ multipliers.

There exist two allpass filter structures (known to the author) that can be implemented with a low minimal sample period and a canonical number of multipliers. These are the lossless discrete integrator/differentiator (LDI/LDD) allpass filter [27] and the wave digital (WD) lattice filter [13]. It has been shown that the LDI/LDD allpass filter can be implemented with less hardware resources compared

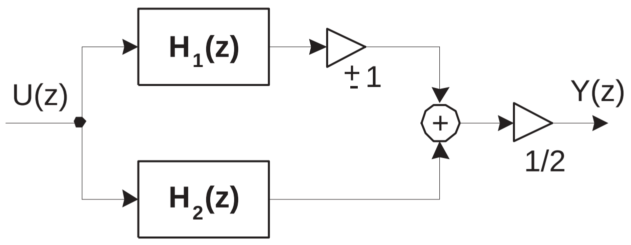

Figure 1.3: Digital lattice filter.

to the corresponding WD implementation case when considering low- and higpass filters [27]. Furthermore, the former filter structure exhibits a lower amount of quantization noise compared to the latter case [27]. Therefore, in this thesis, the LDI allpass filter structure is considered.

Wave digital filters have good filter properties under finite wordlength conditions, provided that magnitude truncation and saturation arithmetics are used and placed at the delay elements [14]. WD filters are low-sensitive filter structures and are good candidates when designing digital lattice filters. The WD filter consists of first- and second-order cascade connected WD allpass sections. These allpass sections are also known as adaptors. In Fig. 1.4, a three-port series adaptor and a second-order Richards' adaptor are shown.

## 1.2.2 LDI/LDD Allpass Filters

Analog LC filters are passive and have low sensitivity to component variations. These properties are also desirable when designing digital filters. It is, therefore, no surprise that analog LC filters were used as prototypes, not only for WD filters, but also when Bruton [6] began his work on the lossless discrete integrator/differentiator filter (LDI/LDD). Bruton introduced several analog-to-digital transformations, so-called LDI transformations, which can be used to transform the analog prototype filter to a corresponding digital filter structure. Over the years, Bruton and others [6], [27], [49], have studied the LDI/LDD filter and improved upon the original work.

In this thesis, we will mainly consider the lossless discrete integrator (LDI) allpass filter presented in [25]. Over the years several LDI allpass filters have been presented [27]. It has been shown that the LDI allpass filter structure exhibits good filter properties when the poles are placed around $z = 1$, even better than the corresponding WD case. If the poles are placed around $z = -1$ the transformation $z \to -z$ should be used. We then get the corresponding lossless discrete differentiator (LDD) allpass filter structure. In Fig. 1.5, the general-order LDI/LDD allpass filter structure is shown, where the plus and minus signs correspond to LDI and LDD, respectively. The reason for our interest in the LDI/LDD allpass filter is

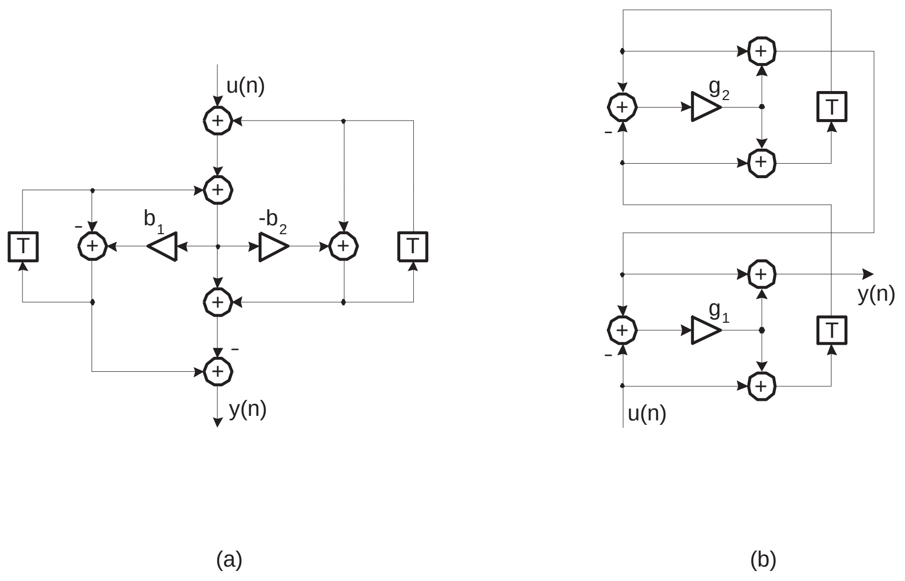

Figure 1.4: WD allpass filters. (a) Three-port series adaptor. (b) Second-order Richards' adaptor.

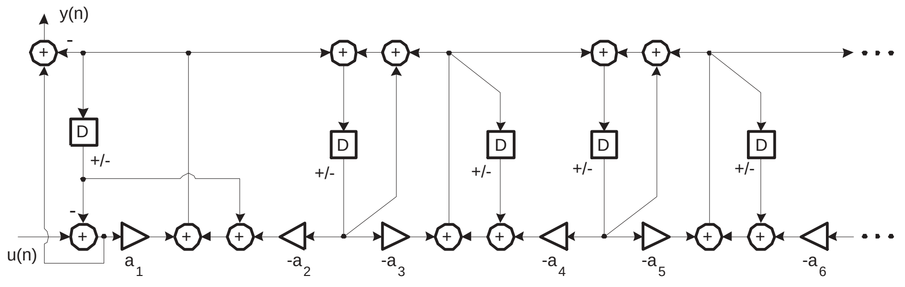

Figure 1.5: General-order LDI/LDD allpass filter structure.

that it, like the WD filter, is a [illegible] *fi*[illegible] [27]. Thus, the length of the filter coefficients can be kept small while maintaining an adequate filter frequency function. This is very advantageous for hardware implementations, since low-sensitive filter structures have good power, area, and throughput characteristics.

### LDI Lattice Filter Design Example

Design formulas for the general-order LDI allpass filter was given in [27]. Let us use these formulas to design an 11th-order LDI lattice filter with the following specification

$$\begin{aligned} \Omega_p &= 0.3\pi & (1.10)\\ r_{\mathrm{p}} &= 0.5\ \mathrm{dB} & (1.11)\\ r_{\mathrm{s}} &= 110\ \mathrm{dB}. & (1.12) \end{aligned}$$

The filter is shown in Fig. 1.6.

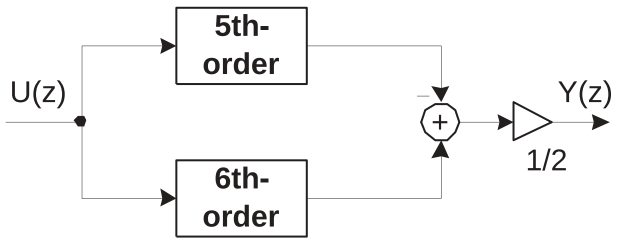

Figure 1.6: 11th-order digital lattice filter.

As presented in [27], the state-space description can be modified in order to simplify the calculation of the filter coefficients. Modifying (1.4) and (1.5) by applying the $z$-transform and introducing a ”differentiator variable”, $\xi = z - 1$, gives us

$$\begin{aligned} \xi X(z) &= A'X(z) + BU(z) & (1.13)\\ Y(z) &= CX(z) + DU(z), & (1.14) \end{aligned}$$

where $A' = A - I$. For the 6th-order LDI allpass filter the characteristic polynominal of $A'$ can be expressed as

$$\begin{aligned} p(\xi) &= \det(\xi I - A') = \xi^6 + c_1\xi^5 + c_2\xi^4 + c_3\xi^3 + c_4\xi^2 + c_5\xi^1 + c_6 \\ &= (\xi + 1 - p_1)(\xi + 1 - p_2)\cdots(\xi + 1 - p_6), & (1.15) \end{aligned}$$

where $p_1, \ldots, p_6$ are the poles of the filter in the $z$-plane. The coefficients can be calculated as

$$\begin{aligned}
\alpha_1 &= c_1 - c_2 + c_3 - c_4 + c_5 - c_6 \\
\alpha_2 &= c_1 - \alpha_1 + \frac{1}{\alpha_1}(-c_3 + 2c_4 - 3c_5 + 4c_6) \\
\alpha_3 &= c_1 - \alpha_1 - \alpha_2 + \frac{1}{\alpha_2}\left(-c_4 + 2c_5 - 3c_6 + \frac{1}{\alpha_1}(c_{-5} - 3c_{-6})\right) \\
\alpha_4 &= c_1 - \alpha_1 - \alpha_2 - \alpha_3 + \frac{1}{\alpha_3}\left(-\frac{1}{\alpha_1}(c_5 - 3c_6) + \frac{1}{\alpha_2}c_6\right) \\
\alpha_5 &= c_1 - \alpha_1 - \alpha_2 - \alpha_3 - \alpha_4 + \frac{1}{\alpha_4}\left(-\frac{1}{\alpha_2}c_6\right) \\
\alpha_6 &= c_1 - \alpha_1 - \alpha_2 - \alpha_3 - \alpha_4 - \alpha_5.
\end{aligned} \tag{1.16}$$

Using MATLAB the values of $c_1, \ldots, c_6$ in (1.15) can be calculated as

$$\begin{aligned}
p(\xi) &= \xi^6 + 2.009672\xi^5 + 2.851608\xi^4 + 2.203817\xi^3 + \ldots \\
&+ 1.247697\xi^2 + 0.366784\xi^1 + 0.067207.
\end{aligned} \tag{1.17}$$

From (1.18) the coefficients for the 6th-order LDI allpass filter can be derived

$$\begin{aligned}
\alpha_1^{(1)} &= 0.413761 \\
\alpha_2^{(1)} &= 0.290938 \\
\alpha_3^{(1)} &= 0.216846 \\
\alpha_4^{(1)} &= 0.312596 \\
\alpha_5^{(1)} &= 0.036551 \\
\alpha_6^{(1)} &= 0.738979.
\end{aligned} \tag{1.18}$$

The coefficients for the 5th-order LDI allpass filter can be calculated in a similar manner, rendering

$$\begin{aligned}
\alpha_1^{(2)} &= 0.414030 \\
\alpha_2^{(2)} &= 0.286741 \\
\alpha_3^{(2)} &= 0.252460 \\
\alpha_4^{(2)} &= 0.137919 \\
\alpha_5^{(2)} &= 0.457750.
\end{aligned} \tag{1.19}$$

The frequency function for the 11th-order LDI digital lattice filter using the coefficients derived above is shown in Fig. 1.7.

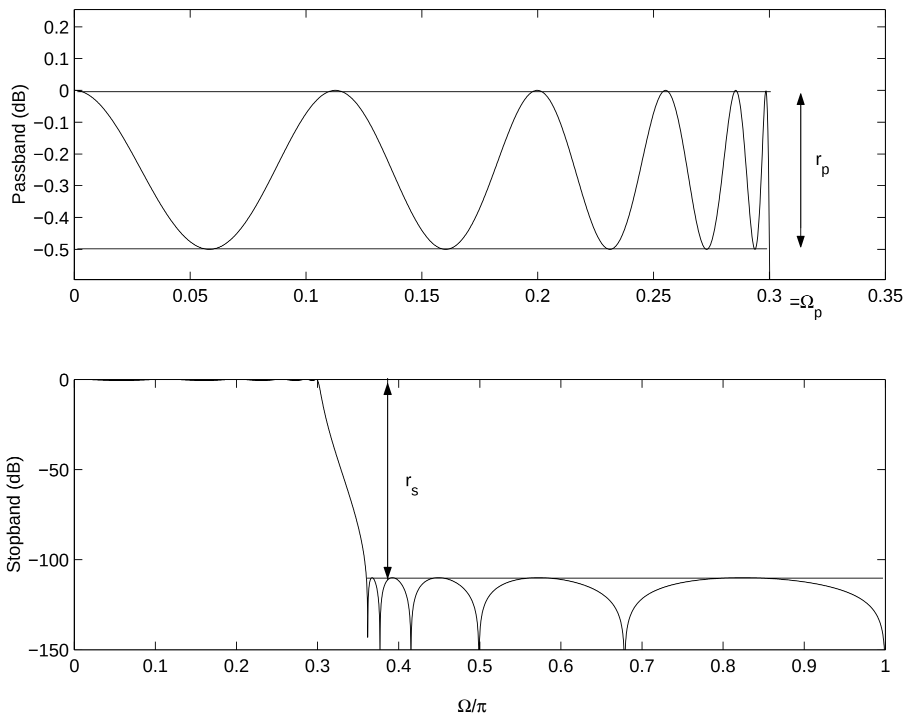

Figure 1.7: Filter specification and frequency function for the 11th-order LDI lattice filter.

## 1.3 Computational Properties of Digital Filter Algorithms

By studying the computational properties of a digital filter algorithm, the performance of the filter can be determined. At this, so-called, [illegible], no information about the realization of the processing elements is known. Still, many issues like parallelism and minimum execution time can be considered at this level.

### 1.3.1 Precedence Graph

When considering computational properties of a digital filter the SFG description is mapped to a [illegible] (PG) [52]. The PG is a graphical description of the state-space representation and describes in which order the processing elements can be executed. By studying the PG it can be derived which processing elements that can operate in parallel and which can execute sequentially. In Fig. 1.8, an

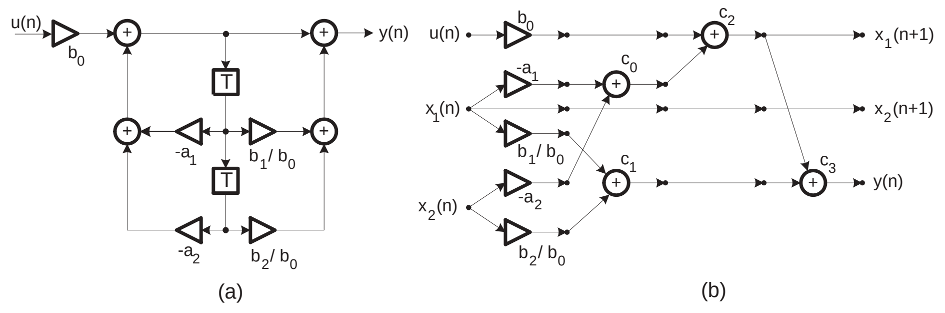

Figure 1.8: a) SFG of second-order DF filter. b) Precedence graph of second-order DF filter.

SFG and a precedence graph of a DF filter is shown.

### 1.3.2 Latency and Throughput at the Algorithmic Level

To describe the computational properties of an algorithm, two expressions, [illegible] and [illegible], are used [46]. Assume that a data flow is applied to a general digital algorithm. Latency at the algorithmic level is the time it takes for the applied data flow to reach the output. Throughput is a measurement of how frequently new input data can be applied to the system. The relationship between latency and throughput depends on the characteristics of the data flow. More on this in Section 1.4.

### 1.3.3 Critical Path and Minimal Sample Period

The throughput of an algorithm is determined by the longest directed path in the precedence graph. This path is known as the [illegible] ($T_{\mathrm{cp_a}}$) of the algorithm [46]. The algorithmic critical path is an upper bound of the throughput. The throughput cannot be increased higher than $1/T_{\mathrm{cp_a}}$ without rearranging the algorithm. For the DF filter, shown in Fig. 1.8, the algorithmic critical path is one multiplier and three adders.

Decreasing the algorithmic critical path is often desirable in high-performance digital filter design. Equivalent transformations, such as associative and distributive methods or re-timing, can sometimes be used [52]. Pipelining is another method which sometimes can be used to decrease $T_{\mathrm{cp_a}}$. By introducing delay elements between the processing elements, a long sequential chain of processing elements can be divided into smaller chains, allowing some processing elements to execute in parallel. Note, however, that in recursive algorithms the sample period is restricted by the recursive loops of the structure. Pipelining the loops will not increase the throughput of the algorithm. Another method that can increase the throughput is unfolding [46]. Unfolding does not decrease the algorithmic critical path of the algorithm, but unfolding increases the parallelism of the algorithm, resulting in a higher throughput.

In recursive algorithms, the recursive loops impose a theoretical bound on the sample period. This bound, also known as the [illegible], is given by

$$T_{\mathrm{min}} = \max_i \left\{ \frac{T_{\mathrm{opt}}}{N_i} \right\}, \tag{1.20}$$

where $T_{\mathrm{opt}}$ and $N_i$ are the total latency of the arithmetic operations and the number of delay elements in the directed loop $i$, respectively [48]. When the algorithmic critical path of an algorithm is longer than $T_{\mathrm{min}}$, transformations can be made so that $T_{\mathrm{cp_a}} = T_{\mathrm{min}}$. More on this in Section 4.4.

## 1.4 System Design

Several implementation issues must be considered in order to realize a logical description of a filter algorithm described by a precedence graph. These system design issues will be discussed in the next sections.

### 1.4.1 Number Representation

A digital system can be implemented using either fixed-point arithmetics or floating-point arithmetics. The former is more common when designing low-power systems, since floating-point processing elements are more complex, resulting in a higher power consumption [52]. One drawback when using fixed-point arithmetics is that the number range is quite limited in comparison to floating-point.

In this thesis, fixed-point [illegible] number representation is considered, unless otherwise noted. The magnitude of the signal is assumed to be less or equal to one, i.e, $|X| \leq 1$. Furthermore, for LDI lowpass filters and LDD highpass filters, coefficient magnitudes less or equal to one are usually obtained (when the poles are placed around $z = 1$ and $z = -1$ for LDI and LDD, respectively), thus, $|\alpha_{\mathrm{n}}| \leq 1$, where $\alpha_{\mathrm{n}}$ is an arbitrary filter coefficient. These two conditions are sufficient to prevent overflow after multiplication. An $f_x + 1$ bit number is represented as (given that $|X| \leq 1$):

$$X = x_0.x_{-1} \cdots x_{-f_{x+1}} x_{-f_x} \tag{1.21}$$

where the bits $x_{-i}$, $i = 0, \ldots, f_{\mathrm{x}}$, are either 0 or 1 and the most significant bit $x_0$ is the sign bit, with the value 0 denoting positive numbers and 1 denoting negative numbers. The value of a two's complement number is given by:

$$X = -x_0 + \sum_{i=1}^{f_x} x_{-i} 2^{-i}. \tag{1.22}$$

The number range is $-1 \leq X \leq 1 - Q$, where $Q = 2^{-f_{\mathrm{x}}}$.

### 1.4.2 Signal Quantization

In finite-precision digital filters, the increase in wordlength after multiplication must be handled. Multiplying two numbers $x$ and $y$, where $f_{\mathrm{x}} + 1$ and $f_{\mathrm{y}} + 1$ are the wordlengths of them, respectively, will result in a product of length $f_{\mathrm{x}} + f_{\mathrm{y}} + 1$ bits. The removal of the least significant bits is commonly known as [illegible]. Quantization will introduce an error that can be modeled as noise. This quantization noise will affect the [illegible] (SNR) at the output of the filter and should, therefore, be kept as small as possible. Different filter structures will have different noise properties [39]. It has been shown in [27] that the LDI allpass filter has good noise properties, even better than the corresponding WD realization, provided that the poles are placed around $z = 1$.

Quantization can be performed in several ways, as seen in Fig. 1.9. The easiest quantization scheme is value truncation, where the unwanted bits are simply dropped. The error introduced by quantization is different depending on the type of quantization used. Round-off will introduce a smaller error than both magnitude truncation and value truncation [27]. The round-off function is, however, more complex to implement in hardware.

Regardless of quantization method, they can all give rise to unwanted oscillations, so-called [illegible] [27], due to the fact that they are nonlinear functions. A stable linear system may, therefore, become unstable when introducing these nonlinearities. Quantization limit cycles are usually small in amplitude, a couple of least significant bits. The effect of the limit cycles can be reduced by increasing the wordlength of the signal. Magnitude truncation has been shown to have the best properties for suppressing quantization limit cycles of the ones presented here [27]. Different filter structures will have different quantization behaviour and this must be studied for each structure. Quantization limit cycles in LDI/LDD allpass filters are studied in Chapter 2 and Publication VII.

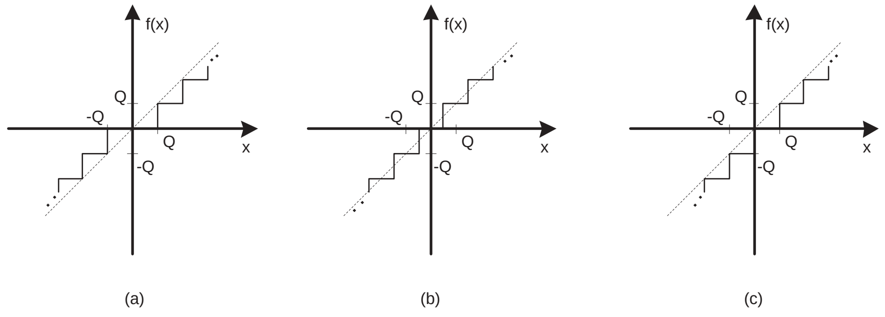

Figure 1.9: Quantization. (a) Magnitude truncation. (b) Round-off. (c) Value truncation.

### 1.4.3 Overflow

Overflow occurs when the sum of two numbers exceeds the range of the number representation. When considering two's complement numbers, overflow is easily detected. If the sum of two positive numbers becomes negative or, vice versa, an overflow has occured. This overflow characteristic, also known as wrapping, is shown in Fig. 1.10(a).

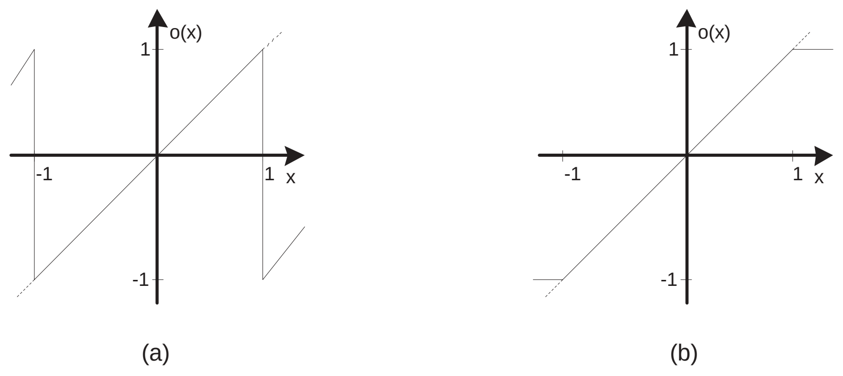

Figure 1.10: Overflow characteristics. (a) Wrapping (two's complement). (b) Saturation.

To prevent wrapping when overflow occurs, a saturation nonlinearity can be used. Saturation will set the overflowed signal to the largest or smallest value of the number representation, depending on the sign of the overflowed signal. A typical saturation characteristic is shown in Fig. 1.10(b). Overflow is undesirable when designing digital filter, since it can cause [illegible]*fl*[illegible]. Overflow limit cycles are large in amplitude and are sustained even when the input signal is zero. To suppress overflow limit cycles, saturation arithmetics can usually be used [27]. More on limit cycles in Chapter 2 and in Publication VII.

The probability of overflow can be reduced by using scaling. Critical nodes in the filter structure, where the risk of overflow is high, can be modified by inserting scaling multipliers. These multipliers will lower the gain of the critical node and, thus, prevent overflow [52].

### 1.4.4 Digit-Serial Arithmetics

Bit-serial and bit-parallel arithmetics are common implementation styles (also known as data-flow architectures) in digital processing systems. The logical values of a signal are referred to as a word. In bit-serial implementation each word is processed one bit at a time on a single wire, usually with the least significant bit (LSB) first. Computation is generally carried out on each bit as they are received by the processing elements. This implementation style usually leads to small processing elements and small overhead due to wiring, since only one wire is used to transmit the signal.

In bit-parallel implementation, the bits in each word are transmitted and processed simultaneously. Clearly, this will require $f_x + 1$ number of wires and more complex processing elements, where $f_x$ is the number of fractional bits in each word. A system using parallel arithmetics can process a word in one clock cycle, whereas a corresponding bit-serial system will require at least $f_\mathrm{x} + 1$ clock cycles. The arithmetic critical path (see Section 1.4.7) in the bit-serial case is, however, usually much smaller. In Fig. 1.11, a bit-parallel and a bit-serial data-flow architecture is shown.

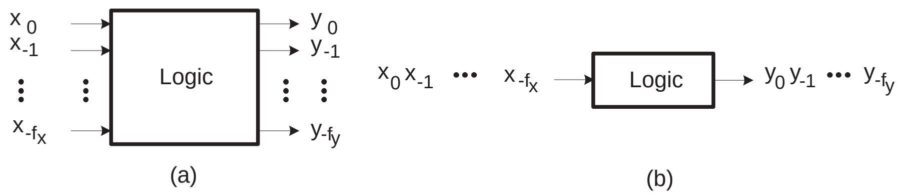
$x_0$
$x_{-1}$
$x_{-f_x}$
Logic
$y_0$
$y_{-1}$
$y_{-f_y}$
(a)
$x_0 \, x_{-1} \cdots x_{-f_x}$
Logic
$y_0 \, y_{-1} \cdots y_{-f_y}$
(b)

Figure 1.11: Data-flow architectures. a) Bit-parallel. b) Bit-serial.

In digit-serial computation, the bits in each word are divided into groups called digits, $d$, and each digit is processed one at a time. Thus, the minimal number of clock cycles to compute a whole word is $(f_x + 1)/d$. Digit-serial computation can therefore be considered as a compromise between bit-serial $d = 1$ and bit-parallel computation $d = f_x + 1$. In order to keep the digits aligned and simplify timing issues, it is required that $(f_x + 1)$ is an integer multiple of $d$, which is assumed throughout this thesis. Furthermore, least significant digit first (LSD) computation is assumed unless otherwise noted. In Fig. 1.12, a digit-serial data-flow architecture is shown.

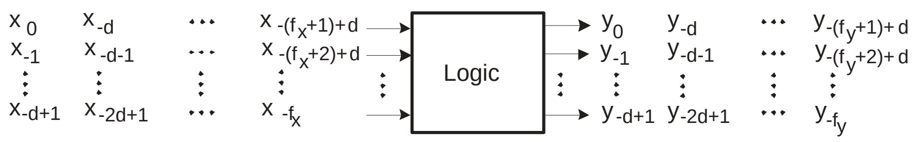
$x_0$ $x_{-d}$ ⋯ $x_{-(f_x+1)+d}$
$x_{-1}$ $x_{-d-1}$ ⋯ $x_{-(f_x+2)+d}$
$x_{-d+1}$ $x_{-2d+1}$ ⋯ $x_{-f_x}$
Logic
$y_0$ $y_{-d}$ ⋯ $y_{-(f_y+1)+d}$
$y_{-1}$ $y_{-d-1}$ ⋯ $y_{-(f_y+2)+d}$
$y_{-d+1}$ $y_{-2d+1}$ ⋯ $y_{-f_y}$

Figure 1.12: Digit-serial data-flow architecture.

The interest in digit-serial processing techniques originates from the late 1970's. Hartley and Corbett studied automatic chip layout tools, also known as silicon compilers, using digit-serial processing elements [20], [21]. Their theories on designing digit-serial layout cells were summarized in [22] where it was concluded that their approach renders faster development time compared to full custom layout. Irwin and Owens also studied this topic [30]. In [44], a systematic approach to generate digit-serial architectures from bit-serial architectures using unfolding was described. Over the years digit-serial computation has been applied to several research areas, including ADSL systems, where the FFT processor architecture can be implemented using digit-serial arithmetics [47] and MPEG video decoding, where the inverse discrete cosine transform can be implemented using digit-serial arithmetics [29]. Digit-serial arithmetics have also been considered when implementing digital filters [1], [32], [38].

The advantage of digit-serial computation is that it allows a trade-off between area, power and throughput. Recently, digit-serial architectures which allow a high degree of pipelining have been presented [8]. These systems tolerate a throughput comparable to parallel systems, but with smaller chip area. It has also been shown in [8] that digit-serial systems are attractive in low-power applications, such as battery-powered systems.

### 1.4.5 Computation Graph

The precedence graph, described in Section 1.3.1, may be extended to a computation graph which also comprises timing information. This is possible at the arithmetic level, when detailed information about the processing elements are known. The throughput of the algorithm and timing of control signals can then be obtained using the computation graph.

### 1.4.6 Latency and Throughput at the Arithmetic Level

Latency and throughput are two common expressions when considering arithmetic operations. The former is the time it takes to produce an output value for a given input value. In digit-serial arithmetics, latency is the time it takes for a digit with a certain significance level to propagate. When studying digital algorithms a distinction between [illegible] (see Section 1.3.2) and [illegible] is usually made. However, in this thesis algorithmic transformations will not be considered. Thus, throughout this thesis latency implies the arithmetic latency ($L$), unless otherwise noted. Throughput is a measure of the sample rate (samples/second). These two concepts are illustrated in Fig. 1.13. In Fig. 1.13 the [illegible] ($T_s = 1/\text{throughput}$) is shown.

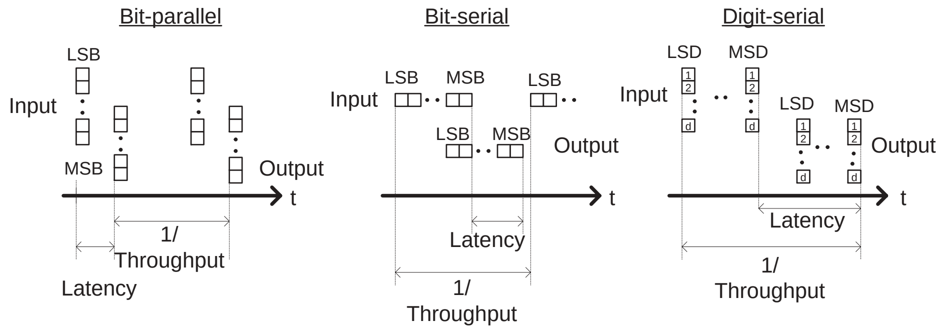

Figure 1.13: Arithmetic latency for bit-parallel, bit-serial and digit-serial systems.

The throughput of a digital filter is best visualized using a computation graph. Let us consider the DF filter shown in Fig. 1.8. In this example we assume that multipliers and adders have a latency of three and one (normal values for digit-serial arithmetics), respectively. The corresponding computation graph is shown in Fig. 1.14, where $T_{\text{clk}}$ is the period time of the clock.

### 1.4.7 Pipelining

Pipelining considerations can be made at both the algorithmic level and the arithmetic level. It is important to distinguish between the two. At the algorithmic level, pipelining is used to exploit inherent parallelism of the algorithm and, hence, increase the throughput of the algorithm, as described in Section 1.3.3.

At the arithmetic level, the longest path, register-output → register-input, in the system determines the clock period. This path is referred to as the arithmetic critical path, $T_{\text{cp}}$. In general the clock period can be expressed as

$$T_{\text{clk}} \geq T_{\text{cp}} \text{ [ns]}, \tag{1.23}$$

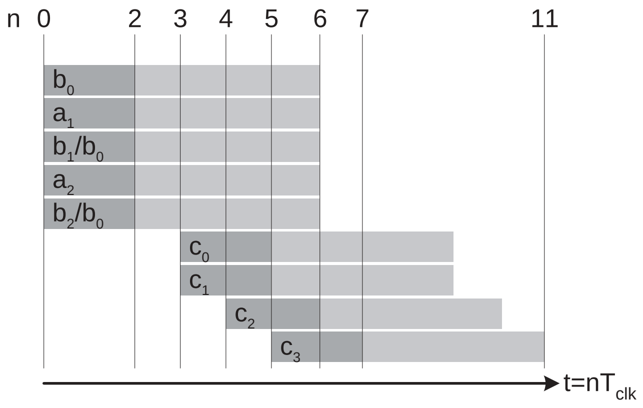

Figure 1.14: Computation graph for a second-order DF filter.

where $T_{\text{clk}}$ is the period time of the clock. In this thesis, however, we use in the theoretical studies that $T_{\text{clk}} = T_{\text{cp}}$.

Pipelining at the arithmetic level corresponds to inserting a number of registers into the structure, hence, shortening the arithmetic critical path. Since recursive loops cannot be pipelined, the arithmetic critical path, i.e., the longest arithmetic loop, will limit the clock period. In the case were the architecture contains no recursive loops, no theoretical lower bound exists on the clock period.

Pipelining at the arithmetic level is not limited to inserting registers between the processing elements. Large processing elements may benefit from introducing pipelining in the structure. We refer to this as internal pipelining. The degree, or level, of pipelining corresponds to the number of register stages inserted into the architecture. In a non-recursive system, pipelining will increase the throughput. We illustrate this by an example. Let us study the system in Fig. 1.15(a).

The sample period of this system can be written as

$$T_{\text{s}} = N_{\text{d}} T_{\text{cp}} \text{ [ns]}, \tag{1.24}$$

where $N_{\text{d}}$ is the number of digits applied to the system and $T_{\text{cp}}$ is the arithmetic critical path. Inserting a pipelining level will result in Fig. 1.15(b). The sample period for this system becomes

$$T_{\text{s}} = (N_{\text{d}} + 1) \max \{T_{\text{cp1}}, T_{\text{cp2}}\}. \tag{1.25}$$

Assuming that

$$T_{\text{cp1}} = T_{\text{cp2}} = \frac{T_{\text{cp}}}{2}, \tag{1.26}$$

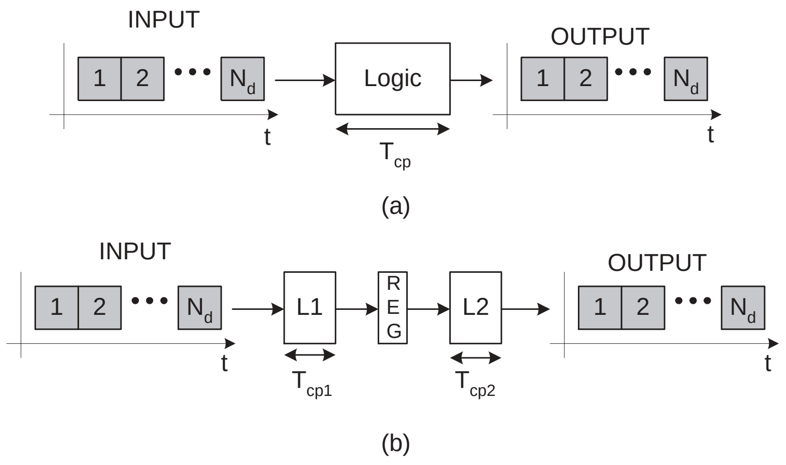

Figure 1.15: Logic system with a) No pipelining and b) one level of pipelining.

the sample period of (1.25) can be expressed as

$$T_\mathrm{s} = (N_\mathrm{d} + 1)\,\frac{T_\mathrm{cp}}{2}. \tag{1.27}$$

It is easy to show that

$$(N_\mathrm{d} + 1)\,\frac{T_\mathrm{cp}}{2} \leq N_\mathrm{d} T_\mathrm{cp}, \tag{1.28}$$

for $N_\mathrm{d} \geq 1$. For an arbitrary degree of pipelining, (1.28) can be extended

$$(N_\mathrm{d} + P - 1)\,\frac{T_\mathrm{cp}}{P} \leq N_\mathrm{d} T_\mathrm{cp}, \tag{1.29}$$

where $P$ is the number of pipelining levels. Condition (1.29) also holds for $N_\mathrm{d} \geq 1$. Clearly, increasing the level of pipelining will result in lower sample period. This analysis is only true in theory, since the delays of the registers are not considered in the analysis above. When the arithmetic critical path is dominated by the delay of the register, further pipelining will not lead to decreased sample period.

### 1.4.8 Power Consumption

As discussed earlier, power consumption is an important implementation aspect in many modern digital systems. The main source of power consumption in CMOS circuits is the dynamic power dissipation caused by switching in the circuit. A model for the dynamic power consumption in a CMOS inverter (approximately true for general CMOS gates) is given by,

$$P_{\mathrm{dyn}} = \alpha f_{\mathrm{clk}} C_{\mathrm{L}} V_{\mathrm{dd}}^2, \tag{1.30}$$

where $\alpha$ is the activity factor, $f_{\mathrm{clk}}$ is the clock frequency and $C_{\mathrm{L}}$ is the capacitive load being switched. The activity factor is the average number of transitions on the output of the gate during one clock cycle. Typically, $\alpha \leq 1$, but due to glitches some systems may experience an activity factor larger than one [7].

It should be quite clear from (1.30) that reducing the supply voltage will have a large impact on the power consumption. Voltage scaling is a popular method to reduce the power consumption of CMOS circuits [37]. Reducing the supply voltage will, however, also affect the speed of the circuit. A first-order approximation of the delay in a CMOS gate was given in [7] and can be expressed as

$$T_{\mathrm{d}} = \frac{C_{\mathrm{L}} V_{\mathrm{dd}}}{\frac{\mu C_{\mathrm{ox}}}{2} (W/L) (V_{\mathrm{dd}} - V_{\mathrm{T}})^m}, \tag{1.31}$$

where $1 \leq m \leq 2$ for short-channel devices.

The reduction in speed caused by voltage scaling may be compensated by scaling of threshold voltages. A technology-independent solution is to increase parallelism in the system to compensate for the speed degradation. As technology improvements lead to smaller voltage supply and smaller threshold voltages, voltage scaling becomes increasingly difficult due to increased leakage currents and noise.

In this thesis, voltage scaling will not be considered. The standard cell design approach used in this thesis is not well suited for voltage scaling due to several reasons. Threshold scaling is not possible when using standard cells. Furthermore, the behavior of the standard cells are only guaranteed using certain voltage ranges. In the case of the UMC 0.18 $\mu$m process, used in this thesis, the functionality of the cells is only guaranteed down to $V_{\mathrm{DD}} = 1.62$ V (under normal operating conditions $V_{\mathrm{DD}} = 1.8$ V).

### 1.4.9 Implementation Considerations

General-purpose processors are not efficient when implementing signal processing algorithms, whereas a more specialized architecture like a programmable digital signal processor will result in a better performance. A dedicated hardware solution (i.e. ASIC) is, however, the most effective implementation method in terms of area and power. The sequential execution of the software processors does not take advantage of inherent parallelism in the algorithms, and as a result, lowering power consumption by increasing parallelism is not possible [37]. The power consumption for a dedicated hardware implementation can be two to three orders of magnitude lower than the corresponding programmable solution [10].

### 1.4.10 Design Flow

A design flow describes the process of a chip design from concept to production. Several design flows exist and they differ mainly on the level of detail. For more on design flows topologies see [53].

This section will briefly describe all steps in the filter design flow shown in Fig. 1.16. The design flow begins with a filter specification. This specification is the solution to a filter problem. It contains information about the amplitude function, cut-off frequency, allowed attenuation etc. In addition, it may also include implementation constraints, such as minimal throughput and maximum power consumption.

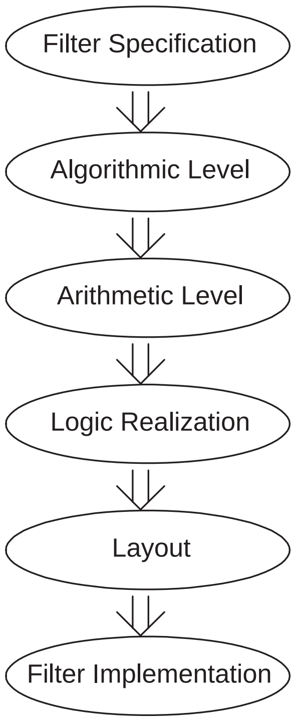

Figure 1.16: Digital filter design flow.

At the algorithmic level, the appropriate digital filter algorithm is chosen and scheduled. This step also includes resource allocation and mapping. Next, at the arithmetic level, arithmetic issues such as number representation and data-flow architectures are determined. Furthermore, it must also be decided how to realize the processing elements. An adder may for example be implemented in several ways depending on performance constraints. In the layout step, chip planning issues like floorplanning and routing are considered. When all steps have been carried out, the filter can be implemented in a integrated circuit like an ASIC or an FPGA.

The flow shown in Fig. 1.16 is a simplification; it does not contain all details comprised in hardware implementation. Several iterations and verifications must usually be performed at each step of the flow. These are left out in order to simplify the design flow graph.

### 1.4.11 Design Tools

All implementations in this thesis were performed using the same methodology. The chosen technology was the UMC 0.18 $\mu$m standard cell technology unless otherwise noted [50]. The main reason for using standard cells instead of a full-custom design approach is that the implementation time becomes much shorter in the former case. The chosen standard cell technology allows up to six metal layers and the recommended supply voltage is 1.8 V. The implementation design flow used is shown in Fig. 1.17. The implementation was described using VHDL and the correctness of

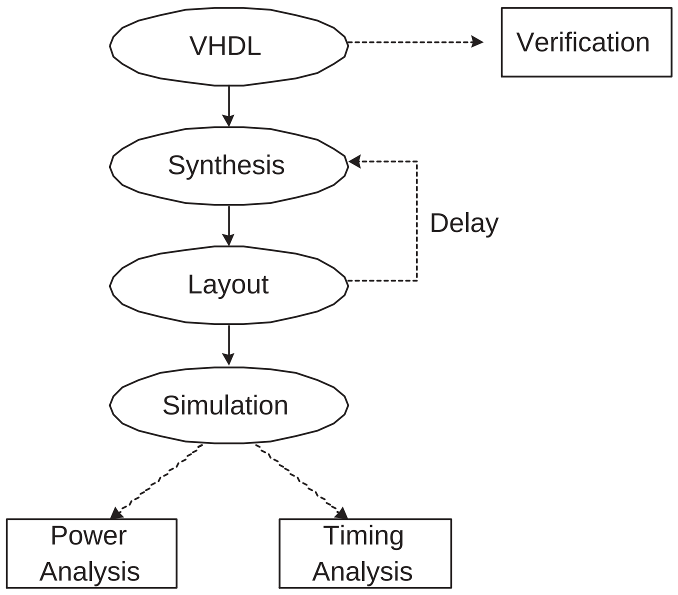

Figure 1.17: Implementation design flow.

the VHDL code was verified using Mentor Graphics Modelsim [40]. Synthesis was performed using Synopsys Design analyzer [11], and circuit layout was generated using Cadence Silicon Ensemble [12]. The layout was generated using four metal layers. A clock tree generation was performed in order to minimize clock skew. Since the performance of the filter logic was studied, no pads were included in the layout. Delay information due to wiring was back-annotated to Design Analyzer, where a static timing analysis was performed. The post-layout netlist with wire RC-delay was then simulated using Spice descriptions of the standard cells in Synopsys Nanosim [42]. From Nanosim, information about current consumption and timing was studied.

## 1.5 Scientific Contributions

This thesis is based on the following publications (Note: The page layout of some papers may have been changed to improve readability. The contents of the papers are, however, unchanged):

**Publication I:** [illegible] K. Landernäs, J. Holmberg, L. Harnefors, and M. Vesterbacka, in [illegible], ISCAS 2002 Vol. 2, pp. II-684–II-687, Phoenix, USA, May 2002.

- In this work, a second-order LDI allpass filter was implemented using digit-serial arithmetics. The performance was compared to a corresponding WD filter. Some theories on maximally fast implementation was also given.

**Publication II:** [illegible] K. Landernäs, J. Holmberg, and M. Vesterbacka, in [illegible], ISCAS 2004, Vol. 3, pp. 23–26, Vancouver, Canada, May 2004.

- In this paper, a new digit-serial adder architecture was presented. The adder was based on both CLAA and conditional sum techniques. It was shown that the proposed adder architecture was well suited for implementation in high-speed recursive filters.

**Publication III:** [illegible] K. Landernäs, J. Holmberg, and O. Gustafsson, in [illegible], NORSIG'2004, pp. 125–128, Espoo, Finland, June 2004

- In this work, two digit-serial multipliers that can be pipelined to the bit-level were implemented and compared to each other. It was shown that a multiplier based on shift-accumulation had a higher throughput as well as lower current consumption.

**Publication IV:** [illegible] K. Landernäs and J. Holmberg, in [illegible], ECCTD'2005.

– In this paper, a 6th-order LDI allpass filter was implemented. Two cases were studied. In the first case unfolding was used to realise the processing elements. Arbitrary pipelining was used in the second case. It was shown that arbitrary pipelining will result in a higher throughput. This is, however, at the expense of much higher current consumption.

**Publication V:** [illegible] K. Landernäs, J. Holmberg and M. Vesterbacka, accepted at [illegible], ICECS'2006.

– Retiming was studied as a method to decrease the power consumption in recursive digital filters. It was shown that retiming can reduce the power consumption with about 20% for small digit-sizes without affecting the throughput of the filter. It was also shown that introducing a large number of registers in the filter structure will increase the current consumption. This trade-off, between reducing the amount of glitches and the increase in the number of registers, was also considered in this work.

**Publication VI:** [illegible] K. Landernäs and J. Holmberg, submitted to [illegible], ISCAS'2006.

– In this work, scheduling considerations for a digit-serial second-order LDI allpass filter was considered. A method known as cyclic scheduling was studied. The filters were implemented in a $0.18\mu m$ process and it was shown that the second-order LDI allpass filter can be realized with 40% less area using cyclic scheduling compared to single interval scheduling. Furthermore, it was also shown that for small digit sizes cyclic scheduling results in a 20% better power-throughput characteristic.

**Publication VII:** [illegible] J. Holmberg, L. Harnefors, K. Landernäs, and S. Signell, submitted to [illegible]

– A new modified lossless discrete integrator/differentiator (LDI/LDD) was presented. The filter structure was analyzed concerning parasitic oscillations, coefficient quantization, quantization noise, and implementation properties. The contribution in this publication is the development of the resulting LDI/LDD structure which has a minimal sample period of $3T_{\text{add}} + T_{\text{mult}}$.

## Other Publications

- [illegible] J. Holmberg, L. Harnefors, K. Landerns, and S. Signell, in [illegible]

[illegible], Vol. 2, pp. 685–688, Sydney, Australia, May 2001.

- [illegible] J. Holmberg, K. Landerns, and M. Vesterbacka in [illegible], Vol. 1, pp. 237–240, Espoo, Finland, August 2001.
- [illegible] J. Holmberg, L. Harnefors, K. Landernäs, and S. Signell, in [illegible], NORSIG'2002, Hurtigruten, Norway, October 2002

# Chapter 2

# Stability Results for the LDI Allpass Filter

It is a well known fact that recursive digital filters can sustain parasitic oscillations, so-called limit cycles, due to finite wordlength [9]. The increase in wordlength after arithmetic operations require signal quantization. Quantization may be carried out in several ways, which was shown in Section 1.4.2. They are all nonlinear operations which may give rise to limit cycles. Naturally, limit cycles are unwanted effects, since they alter the expected behavior of the filter. It is, therefore, important to derive conditions where the filter suppress these phenomena.

Limit cycles can also arise due to signal overflow in the filter [27]. Overflow limit cycles are more serious than quantization limit cycles, since the amplitude of the former is much greater. In contrast to quantization limit cycles, which affect the SNR of the filter, overflow limit cycles will ruin the filter performance. The necessity of studying conditions under which overflow limit cycles do not arise should be apparent. In Fig. 2.1 typical behavior of quantization and overflow limit cycles are shown.

Due to theoretical difficulties, a stability analysis must often include some more or less unrealistic assumptions. One common assumption is to limit the input to a constant or even zero value, with the state $x(n) \neq 0$. The latter is known as a zero-input, autonomous, or unforced system. In this chapter we will limit the analysis to zero-input conditions, which is usually sufficient [27].

The aim of this chapter is to extend the stability region for the second-order LDI allpass filter structure presented in [27]. We will adopt a method presented in [9] and apply the theories to the mentioned filter.

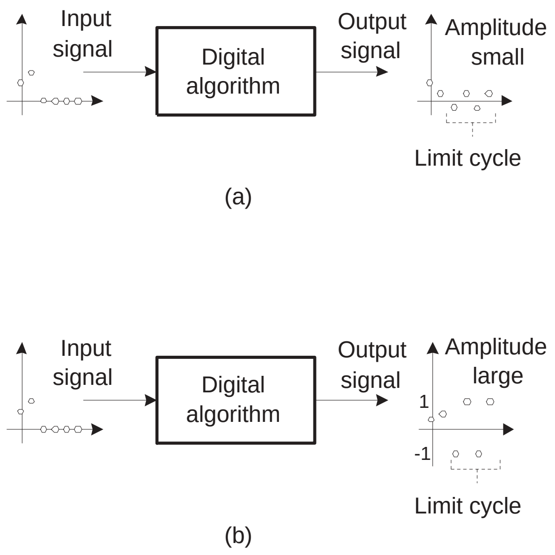

Figure 2.1: Limit cycle behavior. a) Typical quantization limit cycle. b) Typical overflow limit cycle.

## 2.1 Previously Published Results

In this section, we will present previously published stability results for the second-order LDI allpass filter. Over the years several publications have considered this topic [19], [24]. In [27], a stability region for the second-order LDI/LDD allpass filter, which is depicted in Fig. 2.2, was presented.

The stability analysis was performed using Lyapunov theory [27]. With using this approach, a Lyapunov function is introduced. It can be used to describe the "energy" of the linear system, and must meet the following criteria:

$$\begin{aligned} V(x(n)) &> 0,\ x(n) \neq 0,\ V(0) = 0 && (2.1) \\ \Delta V(x(n)) &= V(x(n+1)) - V(x(n)) \leq 0. && (2.2) \end{aligned}$$

This implies that if the "energy" of the system is bounded, it is stable. Furthermore, if the energy is decreasing the system is asymptotically stable. A Lyapunov function can be expressed as the quadratic form

$$V(x(n)) = x^T(n)Px(n), \tag{2.3}$$

where $P$ is a symmetric $N \times N$ matrix. Criterion (2.1) is met only if $P$ is positive

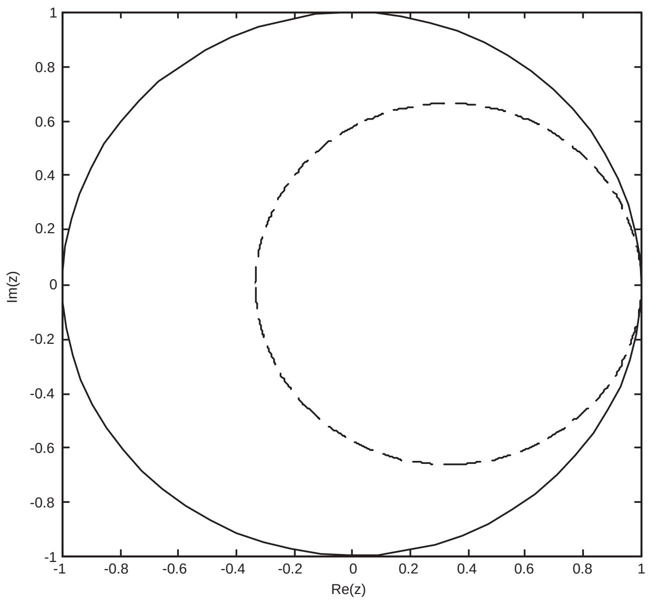

Figure 2.2: Previously published stability region.

definite. For a linear system, we can re-express criterion (2.2) as

$$\begin{aligned}\Delta V(x(n)) &= x^T(n)\underbrace{\left(P^TAP-P\right)}_{-Q}x(n) \leq 0 \\ &\Rightarrow x^T(n)Qx(n) \geq 0. \end{aligned} \tag{2.4}$$

Thus, $Q$ must be positive semidefinite for criterion (2.2) to hold. A Lyapunov function for a linear system can, therefore, be expressed as (2.3), provided that $P$ is positive definite and $Q$ is positive semidefinite.

When studying the nonlinear system, the "trick" is to first find a Lyapunov function with matrices $P$ and $Q$ for the linear system. This Lyapunov function can then be used also for the nonlinear system, as we shall see momentarily. Recall that if the "energy" of a system is always decreasing, the system is asymptotically stable. This can, intuitively, be applied for the nonlinear system. If the "energy" of the nonlinear system is always equal to or smaller than the "energy" of the asymptotically stable linear system, i.e.,

$$g^T(x(n))Pg(x(n)) \leq x^T(n)Px(n), \tag{2.5}$$

where $g(\cdot)$ is the combined overflow and quantization nonlinearities, then the nonlinear system is asymptotically stable. This can be shown strictly as follows. Let us for the moment restrict the study to a diagonal matrix $P = \mathrm{diag}(p_1, p_2, \ldots, p_N)$, since then the calculation of (2.5) will be quite simple:

$$\begin{aligned}\Delta V(x(n)) &= V(x(n+1)) - V(x(n)) = V(g(Ax(n))) - V(x(n)) \\ &\leq V(Ax(n)) - V(x(n)) = -x^T(n)Qx(n) \leq 0 \\ &\Leftarrow p_1g^2(x_1) + p_2g^2(x_2) + \ldots + p_Ng^2(x_N) \\ &\leq p_1x_1^2 + p_2x_2^2 + \ldots + p_Nx_N^2. \end{aligned} \tag{2.6}$$

Thus, it is necessary that

$$\sum_{i=1}^{N} p_i\left(g^2(x_i) - x_i^2\right) \leq 0, \tag{2.7}$$

which is fulfilled if

$$|g(x_i| \leq |x_i|, \quad i = 1, 2, \ldots, N. \tag{2.8}$$

Note that using restriction (2.5), quantization can only be performed immediately before the delay elements. Furthermore, overflow may only occur at the delay elements. To prevent overflow in other nodes in the filter structure, scaling must be used.

In [27], it was shown that (2.5) holds for the second-order LDI allpass filter when the overflow and quantization nonlinearities are placed at the delay elements.

The following matrices $P$ and $Q$ were chosen for the second-order LDI allpass filter

$$V(x(n)) = x^T(n) \overbrace{\begin{bmatrix} 1-\alpha_1 & 0 \\ 0 & \alpha_2 \end{bmatrix}}^{P} x(n), \tag{2.9}$$

$$Q = \begin{bmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{bmatrix} \tag{2.10}$$

where $\alpha_1$ and $\alpha_2$ are the multiplier coefficients and

$$\begin{aligned} q_{11} &= 2\alpha_1 - 3\alpha_1^2 + \alpha_1^3 - \alpha_2 + 2\alpha_2\alpha_1 - \alpha_2\alpha_1^2 \\ q_{12} &= -\alpha_2\alpha_1 + \alpha_2\alpha_1^2 + \alpha_2^2 - \alpha_2^2\alpha_1 \\ q_{22} &= \alpha_2^2 + \alpha_2^2\alpha_1 - \alpha_2^3. \end{aligned} \tag{2.11}$$

The stability region (i.e. where $P$ and $Q$ both are positive definite) for the second-order LDI allpass filter is shown in Fig. 2.2 [27]. In the next section we will adopt a stability analysis presented in [9]. The aim of the analysis is to extend the stability region for the second-order LDI allpass filter presented in [27].

Furthermore, in [27] it was shown that the $N$th-order LDI allpass filter is free from overflow limit cycles when

$$N \quad even: \alpha_{\mathrm{i}} + \alpha_{\mathrm{i}+1} \leq 2, \quad i = 1,3,\ldots,N-1 \tag{2.12}$$

$$N \quad odd: \begin{cases} \alpha_{\mathrm{i}} + \alpha_{\mathrm{i}+1} \leq 2, & i = 1,3,\ldots,N-2 \\ \alpha_N \leq 2, \end{cases} \tag{2.13}$$

where $\alpha_{\mathrm{i}} \geq 0$, $i = 0,1,2,\ldots,N$, provided that saturation arithmetics are used and the nonlinearities are placed at the delay elements.

## 2.2 Stability Analysis for Systems with One Nonlinearity

In [9], Claasen et al. derived a criterion for the absence of zero-input limit cycles of a specific length $N$ in discrete-time systems with one nonlinearity. An arbitrary stable system with one nonlinearity can be divided into a linear part

$$W(z) = \frac{R(z)}{X(z)} \tag{2.14}$$

and a nonlinear part $Q(\cdot)$, as shown in Fig. 2.3. Claasen puts the following restrictions on the non-linearity

$$\begin{aligned} Q(0) &= 0 && (2.15) \\ 0 \leq \frac{Q(x(n))}{x(n)} &\leq k, \ \forall x(n) \neq 0 && (2.16) \\ [Q(x(n)+h) - Q(x(n))] \cdot h &\geq 0, \ \forall x(n), h && (2.17) \end{aligned}$$

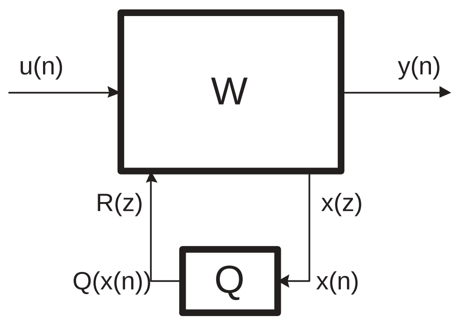

Figure 2.3: Nonlinear system divided into a linear part $W$ and a nonlinear part $Q$.

This implies that the characteristic of the nonlinearity lies in the region shown in Fig. 2.4.

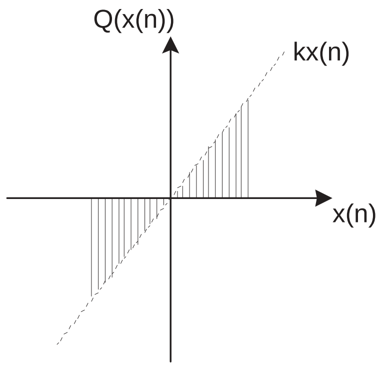

Figure 2.4: Region where the characteristic of the nonlinearity must lie.

In [9], it was shown that limit cycles of length $N$ are absent in the discrete-time system shown in Fig. 2.3, if $W(z)$ is finite for $|z| = 1$ and

$$\Re\{W(z_l)\} - \frac{1}{k} < 0, \tag{2.18}$$

where $z_l = e^{(j2\pi/N)l}$ and $l = 0, 1, \ldots, N/2$. By introducing condition (2.17), (2.18) can be extended and, thus, allowing a larger stability region to be found. This implies that limit cycles of length $N$ are absent in the discrete-time system shown in Fig. 2.3, if $W(z)$ is finite for $|z| = 1$ and the system includes one nonlinearity satisfying (2.15), (2.16), and (2.17), if $\exists\gamma_p \geq 0$ for $l = 0, 1 \ldots N/2$ when

$$\Re\left\{W(z_l)\left(1 + \sum_{p=1}^{N-1}\gamma_p\left(1 - z_l^p\right)\right)\right\} - \frac{1}{k} < 0. \tag{2.19}$$

In most higher-order digital filters, including the LDI/LDD allpass filter, several nonlinearities are required. Thus, the theory presented before must be extended to include the general case. In [9], it was shown that in the general case the nonlinear system can be divided into a linear part $W$ and several nonlinearities $Q_m$ as shown in Fig. 2.5. Each nonlinearity must satisfy the conditions given in (2.15)–(2.17) and

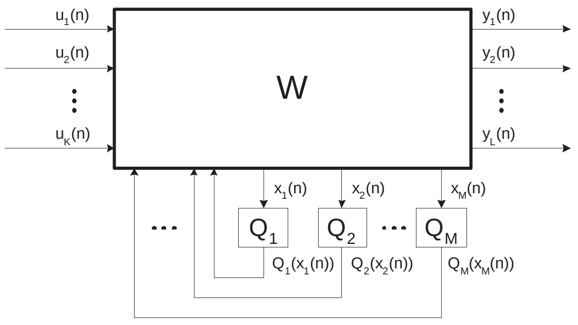

Figure 2.5: Nonlinear system divided into a linear part $W$ and several nonlinearities $Q_m$.

thus

$$\begin{aligned} Q_m(0) &= 0 && \qquad (2.20)\\ 0 \leq \frac{Q_m(x_m(n))}{x_m(n)} &\leq k_m &&, \forall x_m(n) \neq 0 \qquad (2.21)\\ \left[Q_m(x_m(n)+h) - Q_m(x_m(n))\right]\cdot h &\geq 0 &&, \forall x_m(n), h \qquad (2.22)\end{aligned}$$

for $m = 1, 2, \ldots, M$, where $M$ is the number of nonlinearities. The linear part $W$ is in the general case described by a transmission matrix $\overline{W}(z)$ with size $M \times M$. In [9], it was shown that limit cycles of length $N$ are absent from the discrete-time system in Fig. 2.5, where each nonlinearity $Q_m$ satisfies (2.20) and (2.21) when the Hermitian part of

$$\overline{W}\left(z_l\right) - \operatorname{diag}\left(\frac{1}{k_m}\right), \tag{2.23}$$

is negative definite for $l = 0, 1, \ldots, N/2$. Note that $\overline{W}(z)$ must be finite for $|z| = 1$.

The Hermitian part of a matrix $A$ is defined as

$$H(A) \equiv \frac{A + A^*}{2}, \tag{2.24}$$

where $A^*$ is the complex conjugate transpose of $A$.

## 2.3 Stability Analysis for the Second-Order LDI/LDD Allpass Filter

In this section we will apply the theory presented in [9] to the second-order LDI/LDD allpass filter. We consider the case where two nonlinearities placed at the delay elements are used. The SFG of the filter studied is shown in Fig. 2.6. The nonlin-

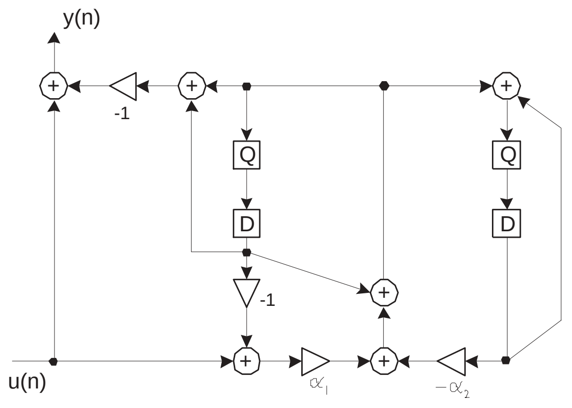

Figure 2.6: Second-order LDI/LDD allpass filter structure with two nonlinearities.

earities are assumed to be magnitude truncation, i.e.,

$$k_m = 1, \tag{2.25}$$

for $m = 1, 2$. The characteristics of the nonlinearities are shown in Fig. 2.7.

The linear transmission matrix for the second-order LDI/LDD filter becomes

$$\overline{W}(z) = \begin{bmatrix} 1 - \alpha_1 z^{-1} & -\alpha_2 z^{-1} \\ 1 - \alpha_1 z^{-1} & 1 - \alpha_2 z^{-1} \end{bmatrix}. \tag{2.26}$$

Combining (2.23) and (2.26) we can show that the second-order LDI/LDD allpass filter, in Fig. 2.6, is free from limit cycles of length $N$ when the Hermitian part of

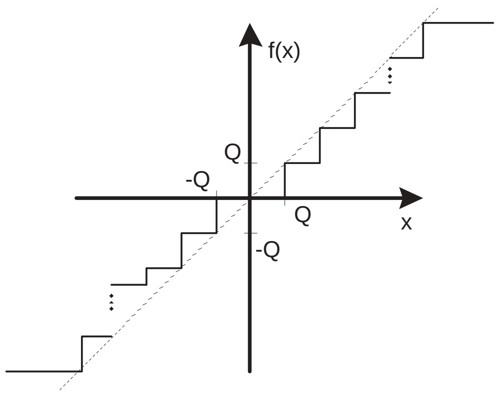

Figure 2.7: Combined saturation and magnitude truncation nonlinearity.

$$\overline{W}(z_l) - \mathrm{diag}\left(\frac{1}{k_m}\right) =$$

$$\frac{1}{2}\begin{bmatrix} (1-\alpha_1)\, z_l^{-1} + (1-\alpha_1)\, z_l - 2 & -\alpha_2 z_l^{-1} + (1-\alpha_1)\, z_l \\ (1-\alpha_1)\, z_l^{-1} - \alpha_2 z_l & (1-\alpha_2)\, z_l^{-1} + (1-\alpha_2)\, z_l - 2 \end{bmatrix} \tag{2.27}$$

is negative definite, where $z_l = e^{(j2\pi/N)l}$ for $l = 0, 1, \ldots, N/2$.

Solving (2.27) numerically for different quotients $l/N$ and different values for $\alpha_1$ and $\alpha_2$ resulted in the region shown in Fig. 2.8. The region in Fig. 2.8 is an extension of the region shown in Fig. 2.2.

## 2.4 Summary

In this chapter a stability analysis for the second-order LDI/LDD allpass filter was made. Applying the theory presented in [4] on the second-order LDI/LDD allpass filter a new stability criterion was found. It was shown that the second-order LDI/LDD allpass filter is free from magnitude-truncation limit cycles in the region presented in Fig. 2.8. This is an extension of the region presented in [27].

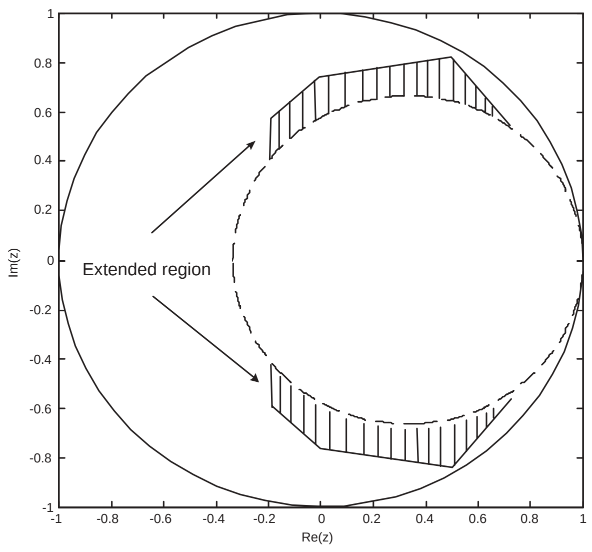

Figure 2.8: Region where the second-order LDI/LDD allpass filter is free from quantization limit cycles (new region in dashed area), provided that the nonlinearity in Fig. 2.7 is placed immediately before the delay elements.

# Chapter 3

# Digit-Serial Processing Elements

## 3.1 Introduction

Adders and multipliers are the main building blocks in a digital filter. They can be realized in several ways and the choice of topology is not trivial. The processing elements will have a significant impact on the system performance and, thus, careful consideration must be made in order to achieve the desired properties. In this chapter, we give an introduction to the processing elements used in Publications I–VI.

## 3.2 Adders

Adders are common components in digital signal processing systems. The design of the adder is important since it is commonly used when designing other arithmetic operations. The computation time of an adder depends to a great extent on how the carry bits are propagated. A carry bit is generated whenever the sum of two bits for any given significance level cannot be represented with a single bit. This carry bit must then propagate, and be added, to the next higher significance level. In the following sections we will give an overview of the most common adders, for the sake of completeness. A study on digit-serial adders can be found in Publication II.

### 3.2.1 Linear-Time Adders

A well known family of adders are [illegible]. The throughput for these adders increases linearly with the size of the input operands and, thus, the theoretical bound of their speed is $O(n)$. The main advantage of linear-time adders are that they are usually small in hardware implementations compared to other types of adders.

The most straightforward method when adding two digits of size $d$ is to utilize $d$ [illegible]. A full adder has two operand bits $x_i$, $y_i$ and an incoming carry $c_i$ as inputs. It computes a sum bit and an outgoing carry bit as

$$s_i = x_i \oplus y_i \oplus c_i \tag{3.1}$$

$$c_{i+1} = x_i \cdot y_i + c_i \cdot (x_i + y_i). \tag{3.2}$$

### Ripple-Carry Adder

By connecting $d$ full adders, two $d$-size numbers can be added. This is shown in Fig. 3.1. With this adder the carry must propagate through all $d$ full adders before the sum is obtained. The adder is, therefore, known as a [illegible] (RCA) [34]. Since the carry bit must propagate through all full adders, the throughput of the RCA is said to be $O(d)$ and the adder is known as a linear-time adder.

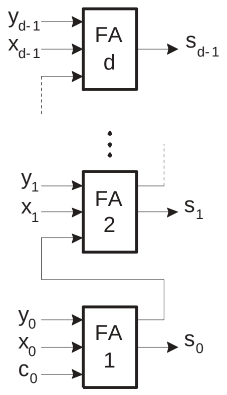

Figure 3.1: Ripple-carry adder.

### Manchester Adder

Another type of linear-time adder is the [illegible] (MCA) [41]. Its carry propagation time is potentially shorter than the corresponding propagation time for the RCA. The MCA utilizes carry-propagate, -generate and -kill functions, defined as

$$p_i = x_i \oplus y_i \tag{3.3}$$

$$g_i = x_i \cdot y_i \tag{3.4}$$

$$k_i = \overline{x_i} \cdot \overline{y_i}, \tag{3.5}$$

respectively. Once a carry is generated it is quickly propagated through a chain of switches controlled by the above functions. The implementation properties of these switches are crucial to the speed of the MCA. Switches are hard to implement with a semi-custom design approach and, thus, MCAs are best suited for full-custom designs. A feasible solution is to use [illegible] to realize the switches [41]. In Fig. 3.2, an MCA module is shown. To realize a $d$-bit MCA, $d$ modules are connected in series. The worst-case propagation delay occurs when the carry

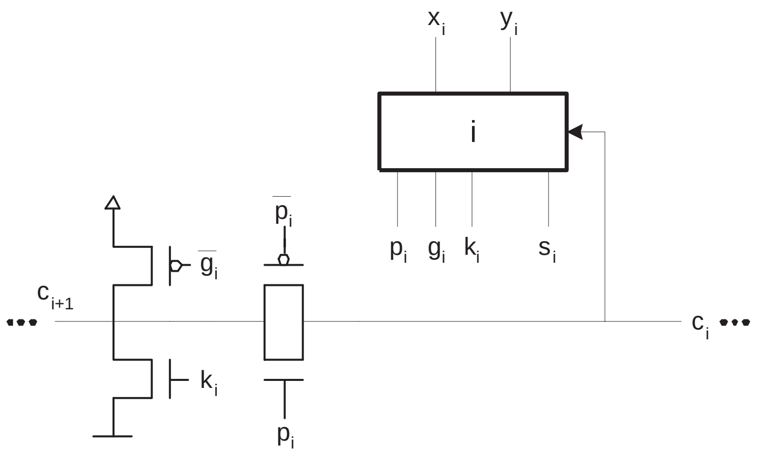

Figure 3.2: Manchester adder module.

must propagate through $d$ switches. In that case, the MCA will be faster than the RCA, provided that the latency for a switch is shorter than the latency for a carry propagation through a full adder.

## 3.2.2 Logarithmic-Time Adders

It is quite evident that the sequential computation of the carry bits is not favorable when considering high throughput, especially for large digit sizes. Due to this fact, adders with parallel carry computation have been studied extensively over the years [34], [41]. It has been shown in [34], that a theoretical bound on the speed of these adders is $O(\log(n))$. An overview of the most common [illegible] is given here.

### Carry-Look-Ahead Adders

Generating all carry bits in parallel, and thereby avoiding the long propagation time of a single carry, is possible since the generation and propagation of the carry bits depend only on the input operands to the adder. Consider Fig. 3.3: a carry is generated ($c_{\mathrm{i}+1} = 1$) at a significance level $i$ if $x_i$ and $y_i$ are both 1. When $x_{\mathrm{i}}$ is 1 and $y_{\mathrm{i}}$ is 0, or vice versa, a carry will propagate, i.e., $c_{\mathrm{i}+1} = c_{\mathrm{i}}$.

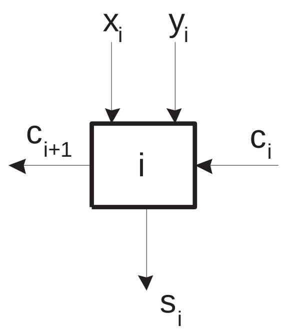

Figure 3.3: Carry schematic.

Let $p_i$ and $g_i$ denote the carry propagation and generation at stage $i$, respectively. The computation of $p_i$ and $g_i$ can then be expressed as shown in (3.3) and (3.4). A carry, $c_{i+1}$, and sum $s_i$ from each stage can then be calculated as

$$s_i = p_i \oplus c_i \tag{3.6}$$
$$c_{i+1} = g_i + c_i p_i. \tag{3.7}$$

This approach when calculating the carry is commonly known as a carry-look-ahead scheme and, hence, the adder as a [illegible] (CLAA) [34]. The CLAA consists of three parts. First, the propagator and generator functions (3.3)–(3.4) are generated. After that the individual carry bits must be computed and a final sum is generated. The three steps are shown in Fig. 3.4. The crucial part of the adder is the carry-generation stage, as it can be implemented in numerous ways. A very straightforward method is to implement the carry-generation using two-level logic [34]. This is, however, only feasible when the operands are small, since the fan-in increases as $(d + 1)$. This problem could be lessen by connecting several CLAA groups in a ripple-carry manner, resulting in an adder with faster computational speed than the RCA.

The computational speed of the CLAA can be increased further, compared to connecting several CLAAs in series, by applying a group carry-look-ahead technique. This will require two new functions, the group-generate $G_{i:j}$ and the group-propagate $P_{i:j}$

$$G_{i:j} = g_j + \sum_{k=i}^{j-1} \left( \prod_{h=k+1}^{j} p_h \right) g_k \tag{3.8}$$

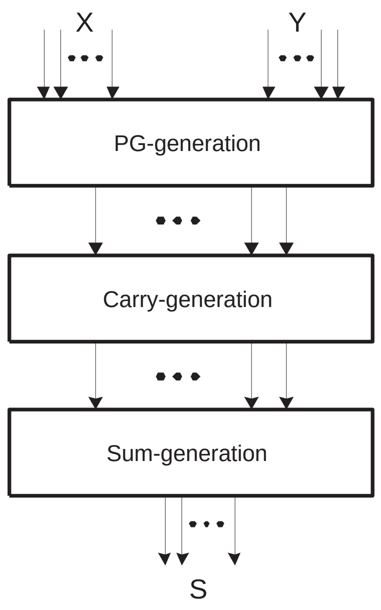

Figure 3.4: Block diagram of a carry-look-ahead adder.

$$P_{i:j} \quad = \quad \prod_{k=i}^{j} p_k \ , \tag{3.9}$$

where $i : j$ indicates bit $i$ to $j$, i.e., the group size. The carry out from each group can then be calculated as

$$C_{j+1} = G_{i:j} + C_i \cdot P_{i:j}. \tag{3.10}$$

Note that we are using capital letters to distinguish group carry bits from carry bits. Also note that $C_0 = c_0$. By calculating the group carry bits first, more carries can be calculated in parallel resulting in a faster computation. For large digit sizes, additional higher levels of carry-look-ahead may be introduced in order to further increase the speed of the adder.

### Parallel Prefix Adders

It should be clear from (3.7) that the carry-retaining logic can be realized in many ways. An obvious question is, what approach yields the fastest result? Let us consider a general logic function with associative operators (in this case addition):

$$f = x_0 + x_1 + \cdots + x_{\mathrm{m}}. \tag{3.11}$$

Assuming that we use operators with the same number of inputs, a binary tree would result in a minimal logic depth of $O(\log(n))$, where $n$ is the number of operands to be added [34]. This is true when the operators considered are associative, i.e., the ordering of the operators can be arbitrary.

Equation (3.7) has been shown to be associative [5]. Thus, a tree realization of the carry generation would result in the minimal logic depth network. The maximum theoretical speed speed of the CLAA is therefore said to be $O(\log(n))$ and the CLAA is referred to as a logarithmic-time adder.

Note that this is a theoretic bound and the actual speed of the CLAA will depend to a great extent on the chosen implementation approach and technology. Ultimately, implementation issues like fan-out, wire delay and number of logic levels will determine the speed of the adder. Adders realized using a tree approach are known as *parallel prefix adders* [28].

Over the years several parallel prefix adders have been proposed. They differ regarding the number of logic levels required, fan-out and overhead due to wiring. Well known parallel prefix adders are the Brent-Kung adder [5], Kogge-Stone adder [33] and the Ladner-Fisher adder [35]. The Ladner-Fisher adder can be realized with a minimal number of levels in the carry retaining tree structure [34]. Furthermore, it has been shown in [28] that the Ladner-Fisher adder is well suited for deep-submicron implementation. The wire overhead for the Ladner-Fisher adder is smaller than for the other two parallel prefix adders mentioned above. It requires extra buffers, but since wire decoupling effects dominate overall delay in deep-submicron designs [28], the Ladner-Fisher adder becomes a good candidate when designing deep-submicron adders. In Fig. 3.5, a functional graph of a Ladner-Fisher prefix adder for $d = 8$ is shown.

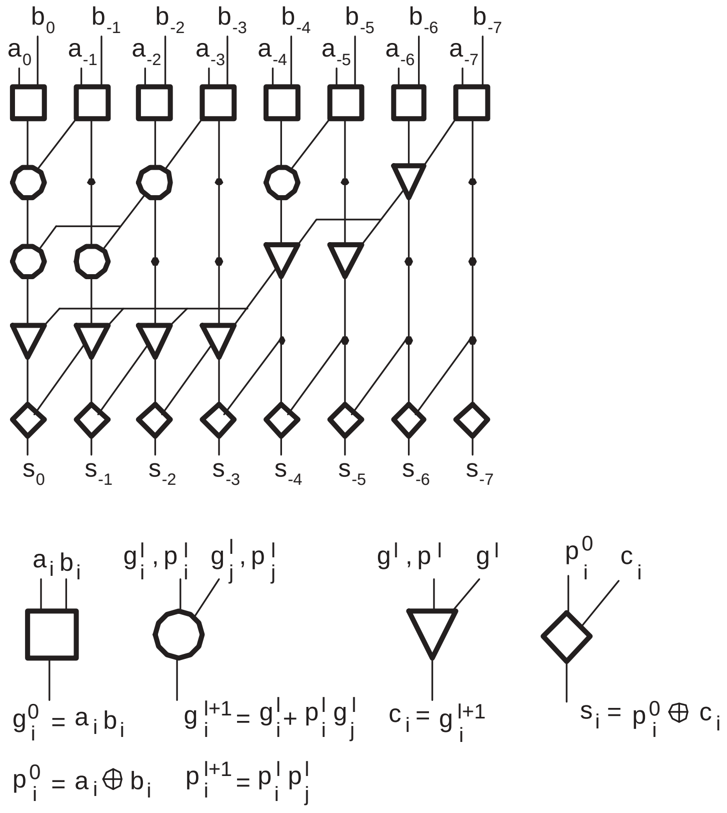
$b_0$ $b_{-1}$ $b_{-2}$ $b_{-3}$ $b_{-4}$ $b_{-5}$ $b_{-6}$ $b_{-7}$
$a_0$ $a_{-1}$ $a_{-2}$ $a_{-3}$ $a_{-4}$ $a_{-5}$ $a_{-6}$ $a_{-7}$
$s_0$ $s_{-1}$ $s_{-2}$ $s_{-3}$ $s_{-4}$ $s_{-5}$ $s_{-6}$ $s_{-7}$
$a_i\, b_i$
$g_i^l, p_i^l$ $g_j^l, p_j^l$
$g^l, p^l$ $g^l$
$p_i^0$ $c_i$
$g_i^0 = a_i b_i$
$p_i^0 = a_i \oplus b_i$
$g_i^{l+1} = g_i^l + p_i^l g_j^l$
$p_i^{l+1} = p_i^l p_j^l$
$c_i = g_i^{l+1}$
$s_i = p_i^0 \oplus c_i$

Figure 3.5: Functional graph of a Ladner-Fisher prefix adder.

## Conditional Sum Adder

Another logarithmic-time adder is the [illegible] [34]. It will generate two sets of outputs. Each set will include a sum and a carry-out. One set will generate an output assuming that the incoming carry bit is zero. The other set is calculated using a carry-in equal to one. When the correct carry-in is known we can simply choose which output to use. A functional diagram of a conditional sum adder is shown in Fig. 3.5.

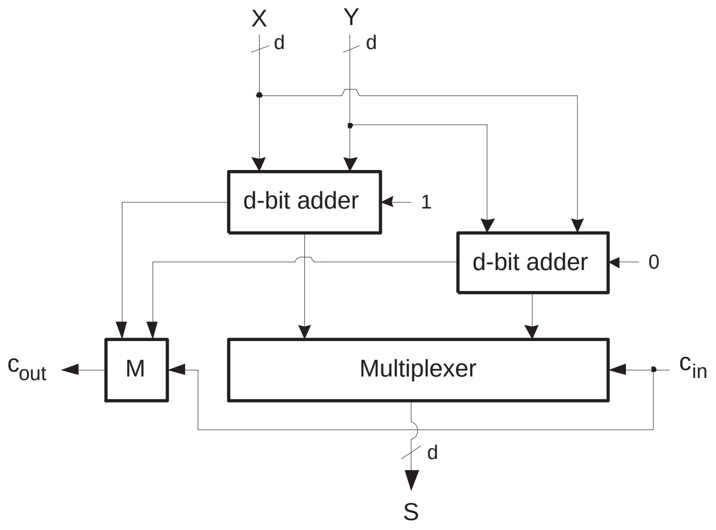

Figure 3.6: Functional graph of a conditional sum adder.

### 3.2.3 Digit-Serial Adders

A digit-serial adder will add two incoming digits of size $d$. The sum, also of size $d$, must be generated and the most significant carry bit added to the next succeeding digit. This can be performed using the general digit-serial adder structure shown in Fig. 3.7. The latency of the carry loop will become a lower bound on the arithmetic critical path of the adder, since loops cannot be pipelined.

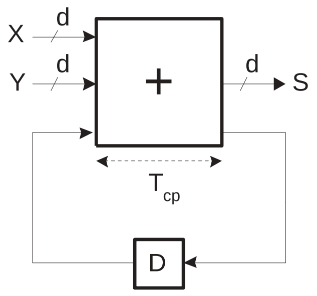

Figure 3.7: General digit-serial adder.

### Digit-Serial Ripple-Carry Adder

A digit-serial RCA (DSRCA) can be realized by delaying the most significant carry bit one clock cycle and adding it to the least significant full adder (as the least significant carry bit). This is shown in Fig. 3.8. Clearly, the latency for the DSRCA

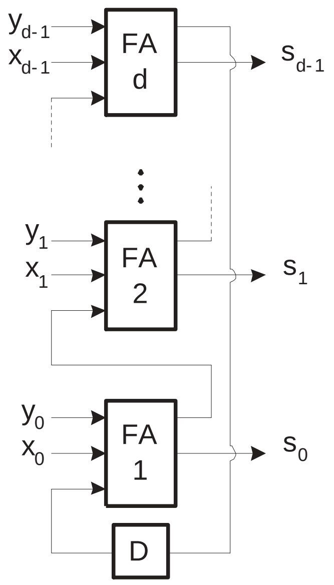

Figure 3.8: Digit-serial RCA.

is proportional to the input digit size and can be expressed as

$$L_{\mathrm{RCA}} = (d+1)T_{\mathrm{fa}}. \tag{3.12}$$

Note, that we assume that the delay of the D flip-flop is roughly the same as the delay of the full adder. This assumption is, e.g., the case in the UMC $0.18\,\mu$m technology where the latency for the full adder is $L_{\mathrm{fa}} \approx 0.23$ ns and the latency for the D flip-flop is $L_{\mathrm{ff}} \approx 0.19$ ns [50].

The main drawback of the DSRCA is that the arithmetic critical path becomes the carry retaining loop. The throughput of the DSRCA can therefore not be increased by introducing pipelining.

### Digit-Serial Carry-Look-Ahead Adder

To realize a digit-serial CLAA the most significant carry in the carry-generation must be delayed one clock cycle and fed back as the least significant carry for the next computation. This is shown in Fig. 3.9. As discussed in Section 1.4.7,

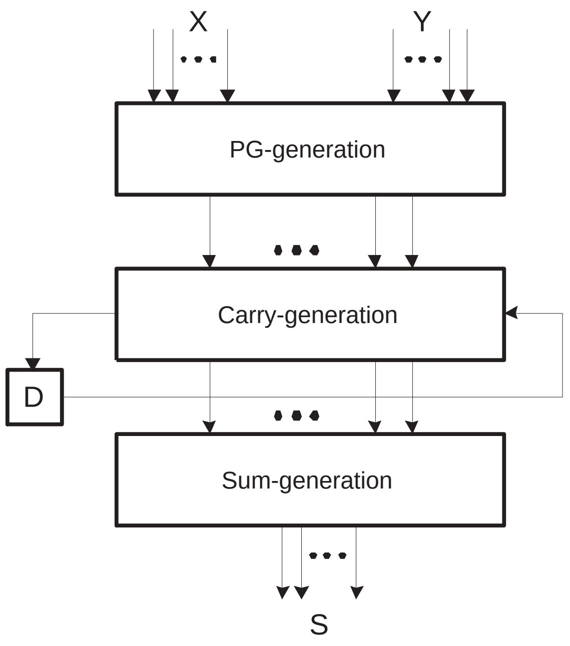

Figure 3.9: Digit-serial CLAA.

pipelining can be used to increase the throughput of a digital system by shortening $T_{\mathrm{cp}}$. The carry loop, shown in Fig. 3.9, cannot be pipelined in order to increase the throughput of the adder. It is, therefore, important that the latency in this loop is kept short. A system that can be pipelined to an arbitrary degree is often referred to as a [illegible] pipelined system [3]. For a bit-level pipelined system it is assumed that $T_{\mathrm{cp}} \leq T_{\mathrm{fa}}$, where $T_{\mathrm{fa}}$ is the latency of a full adder. Bit-level pipelined digit-serial adders are common when designing fast digit-serial systems. Several digit-serial multipliers have been proposed that require a fast bit-level pipelined digit-serial adder [8], [43].

In [3], a bit-level pipelined digit-serial CLAA (DSCLA) was presented. By dividing the propagate and generate function into groups, as described in Section 3.2.2, only the logic required to generate the group carry bits is contained by the carry loop. For $d \leq 24$ and an appropriate group size, this propagation delay can be kept less than or equal to the latency one full adder, thus, allowing bit-level pipelining of the adder. The DSCLA will have two additional stages compared to the standard CLAA, as seen in Fig. 3.10.

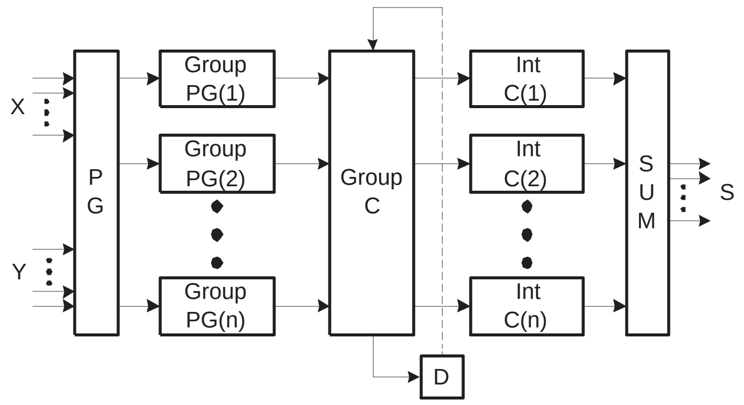

Figure 3.10: Digit-serial CLAA (DSCLA).

In Section 3.2.2, it was concluded that parallel prefix adders are well suited for high-speed applications. By modifying the Ladner-Fisher parallel prefix adder, shown in Fig. 3.5, a digit-serial Ladner-Fisher adder (DSLF) can be obtained. The modification is necessary, since the former adder assumes that the incoming carry is zero. A DSLF for $d = 8$ is shown in Fig. 3.11. For a functional description of the elements in the DSLF see Fig. 3.5.

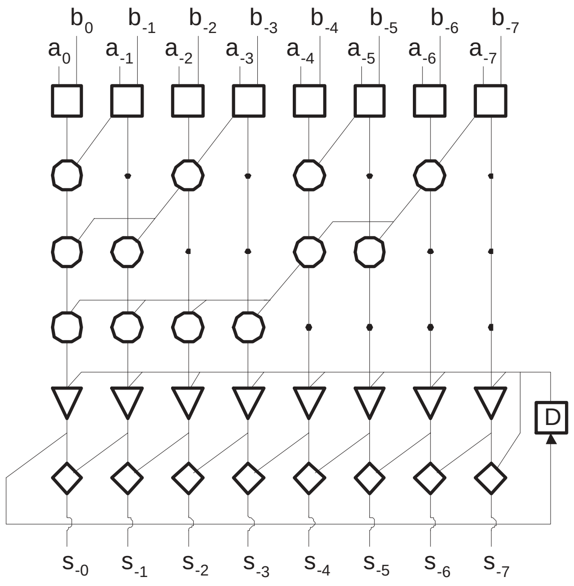
$b_0$ $b_{-1}$ $b_{-2}$ $b_{-3}$ $b_{-4}$ $b_{-5}$ $b_{-6}$ $b_{-7}$
$a_0$ $a_{-1}$ $a_{-2}$ $a_{-3}$ $a_{-4}$ $a_{-5}$ $a_{-6}$ $a_{-7}$
D
$s_{-0}$ $s_{-1}$ $s_{-2}$ $s_{-3}$ $s_{-4}$ $s_{-5}$ $s_{-6}$ $s_{-7}$

Figure 3.11: Digit-serial Ladner-Fisher adder with $d = 8$.

## 3.3 A New Digit-Serial Hybrid Adder

Hybrid adders combine two or more of the common adder methods to perform addition [51]. In Publication II, a digit-serial adder based on carry-look-ahead and conditional sum techniques is presented. We will denote this adder DSHA. The DSHA can be pipelined to the bit level and has low latency. The adder executes two functions in parallel: one generates the group carry bits while the other generates all possible sums. When the correct group carry is known we simply choose the appropriate sum, thus, the adder is based on both carry-look-ahead and conditional sum techniques. The DSHA is shown in Fig. 3.12. Obviously, the DSHA will become larger than the DSCLA. The main advantage of the DSHA compared to the DSCLA is that the former has shorter logic depth. This makes the DSHA well suited for systems requiring low latency, e.g., recursive digital filters.

## 3.4 Digit-Serial Shifting

Shifting, i.e., multiplication by $2^{\pm n}, n = 1, 2, ...$, is a common operation in many multipliers [52]. So also in digit-serial multipliers. In order to better visualize digit-serial arithmetic operations, such as shifting and multiplying, we will utilize

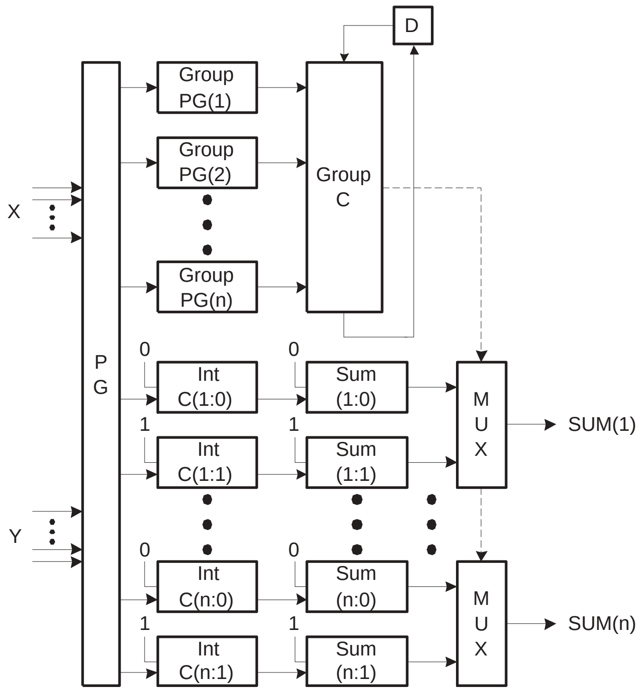

Figure 3.12: Digit-serial hybrid adder (DSHA).

a [illegible]. A number

$$X = -x_0 \cdot 2^0 + x_{-1} \cdot 2^{-1} + \cdots + x_{-f_x} \cdot 2^{-f_x}, \tag{3.13}$$

can be expressed, using the matrix notation, as

$$X = \begin{bmatrix} -x_0 & x_{-1} & \cdots & x_{-f_x} \end{bmatrix} \cdot \begin{bmatrix} 2^0 \\ 2^{-1} \\ \vdots \\ 2^{-f_x} \end{bmatrix} \tag{3.14}$$

Dividing (3.14) into digits of size $d$ yields

$$X = 2^* \left( \underbrace{\begin{bmatrix} -x_0 \\ x_{-1} \\ \vdots \\ x_{-d+1} \end{bmatrix}}_{X_0} + \underbrace{\begin{bmatrix} x_{-d} \\ x_{-d-1} \\ \vdots \\ x_{-2d+1} \end{bmatrix}}_{X_{-1}} \cdot 2^{-d} + \underbrace{\begin{bmatrix} x_{-2d} \\ x_{-2d-1} \\ \vdots \\ x_{-3d+1} \end{bmatrix}}_{X_{-2}} \cdot 2^{-2d} + \cdots \right.$$

$$\left. + \underbrace{\begin{bmatrix} x_{-f_x+d-1} \\ x_{-f_x+d-2} \\ \vdots \\ x_{-f_x} \end{bmatrix}}_{X_{-\frac{f_x+1}{d}} - 1} \cdot 2^{-f_x+d-1} \right), \tag{3.15}$$

with

$$2^* = \begin{bmatrix} 2^0 & 2^{-1} & \cdots & 2^{-d+1} \end{bmatrix}. \tag{3.16}$$

Denoting the number of digits as

$$m = \frac{f_x + 1}{d}, \tag{3.17}$$

we can write a number with arbitrary digit size:

$$X = 2^* \cdot \left( X_0 + X_{-1} \cdot 2^{-d} + \cdots + X_{-m+1} \cdot 2^{(-m+1)d} \right). \tag{3.18}$$

Note that the notation visualizes the computation order of the digits. When considering least significant digit first, or LSD-first computation, $X_{-m+1}$ is received first by the processing element, then $X_{-m+2}$ and so on. Let us study a $2^1$ shift for the number shown in (3.19):

$$X = \underbrace{\begin{bmatrix} 2^0 & 2^{-1} & 2^{-2} & 2^{-3} \end{bmatrix}}_{2^*} \cdot \left( \begin{bmatrix} -x_0 \\ x_{-1} \\ x_{-2} \\ x_{-3} \end{bmatrix} \cdot 2^0 + \begin{bmatrix} x_{-4} \\ x_{-5} \\ x_{-6} \\ x_{-7} \end{bmatrix} \cdot 2^{-4} \right). \tag{3.19}$$

A $2^1$ shift would result in

$$X \cdot 2^1 = 2^* \cdot \left( \begin{bmatrix} 0 \\ 0 \\ 0 \\ -x_0 \end{bmatrix} \cdot 2^4 + \begin{bmatrix} x_{-1} \\ x_{-2} \\ x_{-3} \\ x_{-4} \end{bmatrix} \cdot 2^0 + \begin{bmatrix} x_{-5} \\ x_{-6} \\ x_{-7} \\ 0 \end{bmatrix} \cdot 2^{-4} \right). \tag{3.20}$$

Thus, a $2^1$ shift will move up each bit in the arrays one step. Intuitively, a $2^n$ shift moves up each bit $n$ steps. The logic to realize a $2^n$ shift is shown in Fig. 3.13. The

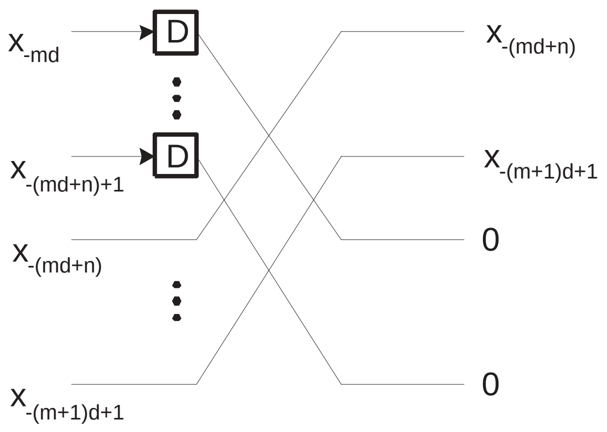

Figure 3.13: $2^n$ shift realization.

latency of the shift operator can be derived by studying the latency of the $2^d$ shift, shown in Fig. 3.14, which can be expressed as

$$L_{2^d} = T_{\text{clk}}. \tag{3.21}$$

A $2^1$ shift will have the following latency

$$L_{2^1} = \left(\frac{1}{d}\right) T_{\text{clk}}, \tag{3.22}$$

or more general

$$L_{2^i} = \left(\frac{i}{d}\right) T_{\text{clk}}, \tag{3.23}$$

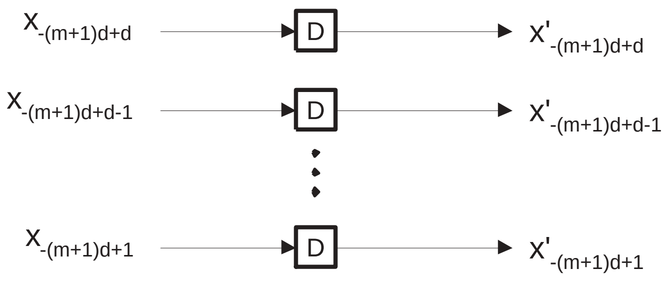

Figure 3.14: $2^d$ shift.

where $i$ is the number of 2 shifts. Usually, it is desired that the latency should be an integer number. This will keep the digits aligned, which simplifies timing issues in the digital system.

A $2^{-1}$ shift is a non-realizable operation, since bits will move across into preceding digits. This type of shifting cannot be implemented unless pipelining is also performed. In Fig. 3.15, the logic for a pipelined $2^{-1}$ shift is shown for $d = 4$. The latency for a pipelined $2^{-1}$ shift is

$$L_{2^{-1}} = \left(\frac{d-1}{d}\right) T_{\text{clk}}. \tag{3.24}$$

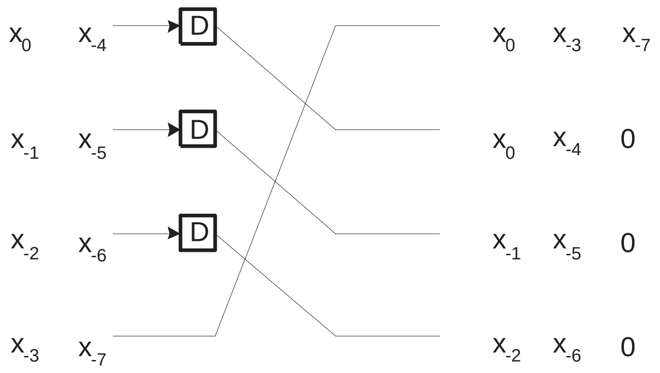

Figure 3.15: Pipelined $2^{-1}$ shift realization.

For $i$ number of $2^{-1}$ shifts the latency becomes

$$L_{2^{-i}} = \left(\frac{d-i}{d}\right) T_{\text{clk}}. \tag{3.25}$$

## 3.5 Digit-Serial Multipliers

Multiplication is performed in two steps; generation and accumulation of partial products. When multiplying two numbers, $X$ and $Y$, a total of $f_{\mathrm{y}}+1$ partial products of length $f_{\mathrm{x}}+1$ is generated, where $f_{\mathrm{x}}+1$ and $f_{\mathrm{y}}+1$ are the wordlengths of $X$ and $Y$, respectively. The generation of the partial products is usually straightforward. A common hardware solution it to use AND gates to obtain the partial products. The accumulation of the partial products can, however, be realized in several ways.

In this thesis, digit-serial multipliers with a fixed digit size will be studied. That is, the input data to the multiplier and the output data from the multiplier are of size $d$ as shown in Fig. 3.17. This allows us to implement the multiplier in any digit-serial system without altering the data flow to or from the multiplier. Furthermore, we assume that the multiplicand is constant. This is the case when implementing, e.g., digital filters with fixed coefficients. Multipliers where one operand is constant can usually be significantly simplified [54].

The multipliers studied in this chapter can only handle positive numbers. In order to operate correctly when the input is a negative number the sign bit must be extended. The logic which retains the sign bit is called sign extension [52]. The sign extension unit can easily be implemented using muxes as shown in Fig. 3.16. A signal, denoted **Ctrl** in Fig. 3.16, controls the behavior of the muxes. The logic used to generate the **Ctrl** signal in Fig. 3.16 allows the **Sel** signal to be one clock period. This simplifies the implementation of the control unit.

### 3.5.1 Digit-Serial/Parallel Multiplier

Serial/parallel (S/P) multipliers are attractive when one of the operands is constant, since the hardware complexity can be significantly reduced [54]. This is the case in, e.g., digital filter implementations with fixed filter coefficients. In bit-serial implementations the S/P multiplier is very efficient since the shift elements provide natural pipelining, i.e., any additional pipelining is superfluous. Let us, therefore, study digit-serial multiplication with fixed digit size and a parallel multiplicand as shown in Fig. 3.17. Let us further assume LSD first computation and positive numbers. Using the notation from (3.18) multiplication can be expressed as

$$\begin{aligned} X \cdot Y &= 2^{*} \cdot \big(X_0 \cdot Y + X_{-1} \cdot Y \cdot 2^{-d} + X_{-2} \cdot Y \cdot 2^{-2d} + \cdots \\ &\quad + X_{-m+1} \cdot Y \cdot 2^{(-m+1)d}\big) . \end{aligned} \tag{3.26}$$

Let us consider one partial product and assume that $Y$ is a positive number:

$$X_{-2} \cdot Y = \begin{bmatrix} x_{-2d} \\ x_{-2d-1} \\ \vdots \\ x_{-3d+1} \end{bmatrix} \cdot \begin{bmatrix} 0 & y_{-1} & \cdots & y_{-f_y} \end{bmatrix} \cdot \begin{bmatrix} 2^{0} \\ 2^{-1} \\ \vdots \\ 2^{-f_y} \end{bmatrix}$$

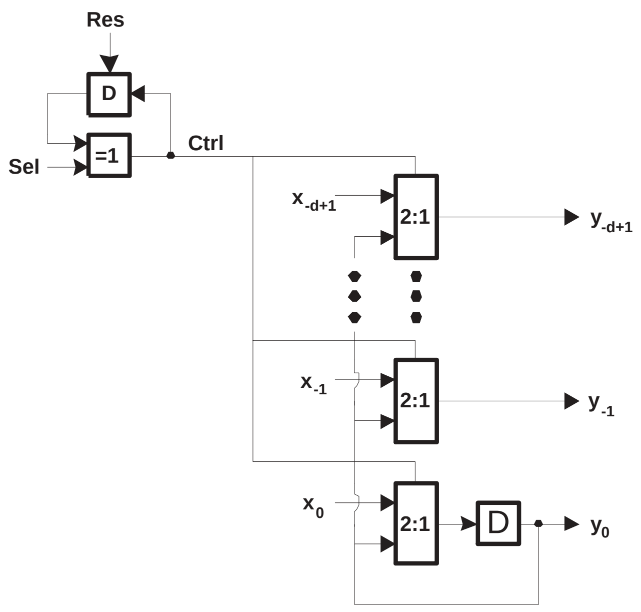

Figure 3.16: Digit-serial sign extension unit.

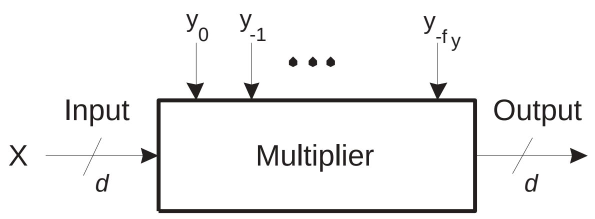

Figure 3.17: Fixed digit size multiplication.

$$= \begin{bmatrix} 0 & x_{-2d} \cdot y_{-1} & x_{-2d} \cdot y_{-2} & \cdots & x_{-2d} \cdot y_{-f_y} \\ 0 & x_{-2d-1} \cdot y_{-1} & x_{-2d-1} \cdot y_{-2} & \cdots & x_{-2d-1} \cdot y_{-f_y} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \underbrace{x_{-3d+1} \cdot y_{-1}}_{X_{-2} \cdot y_{-1}} & \underbrace{x_{-3d+1} \cdot y_{-2}}_{X_{-2} \cdot y_{-2}} & \cdots & \underbrace{x_{-3d+1} \cdot y_{-f_y}}_{X_{-2} \cdot y_{-f_y}} \end{bmatrix} \cdot \begin{bmatrix} 2^0 \\ 2^{-1} \\ \vdots \\ 2^{-f_y} \end{bmatrix} \tag{3.27}$$

$$= \begin{bmatrix} 0 & X_{-2} \cdot y_{-1} & \cdots & X_{-2} \cdot y_{-f_y} \end{bmatrix} \cdot \begin{bmatrix} 2^0 \\ 2^{-1} \\ \vdots \\ 2^{-f_y} \end{bmatrix} \tag{3.28}$$

This multiplication can be realized by the S/P multiplier shown in Fig. 3.18, denoted S/P$_\text{I}$. The function of the multiplier can be shown by rewriting (3.28) as

$$X_{-2} \cdot y_{-1} \cdot 2^{-1} + X_{-2} \cdot y_{-2} \cdot 2^{-2} + \cdots + X_{-2} \cdot y_{-f_y} \cdot 2^{-f_y}, \tag{3.29}$$

and extracting $2^{-f_y}$:

$$2^{-f_y} \left( X_{-2} \cdot y_{-fy} + 2 \left( X_{-2} \cdot y_{-f_y+1} + 2 \left( \cdots + 2 \left( X_{-2} \cdot y_{-2} + 2 \left( X_{-2} \cdot y_{-1} \right) \right) \ldots \right) \right) \right). \tag{3.30}$$

The latency of the S/P$_\text{I}$ multiplier will differ depending on the adder architecture.

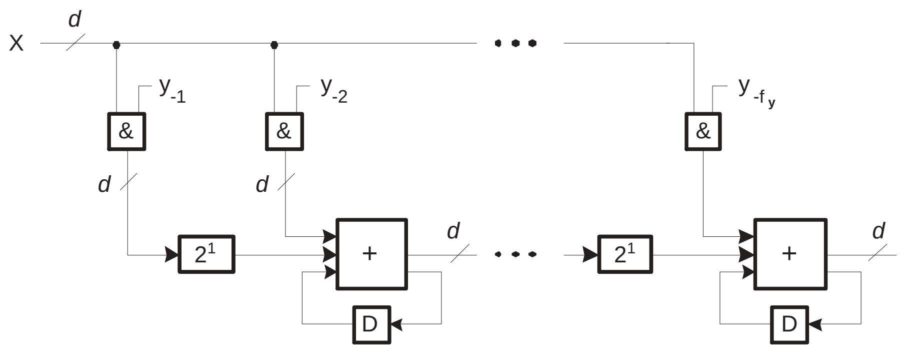

Figure 3.18: S/P$_\text{I}$ multiplier.

Assuming that the adder has no internal pipelining, the latency becomes

$$L = \left( \frac{f_y}{d} \right) T_{\text{cp}}. \tag{3.31}$$

In order to keep the digits aligned, the fractional bits of the parallel operand must be an integer multiple of the digit size. This will simplify timing considerations for the multiplier. When this is not the case, a number of zeros can be added to the most significant end of the coefficient. This will result in an additional number of shifts at the multiplier output.

The same multiplication can be performed using $2^{-1}$ shifts instead. This multiplier, denoted S/P$_{\rm II}$, is shown in Fig. 3.19. The S/P$_{\rm II}$ is easily derived by rewriting

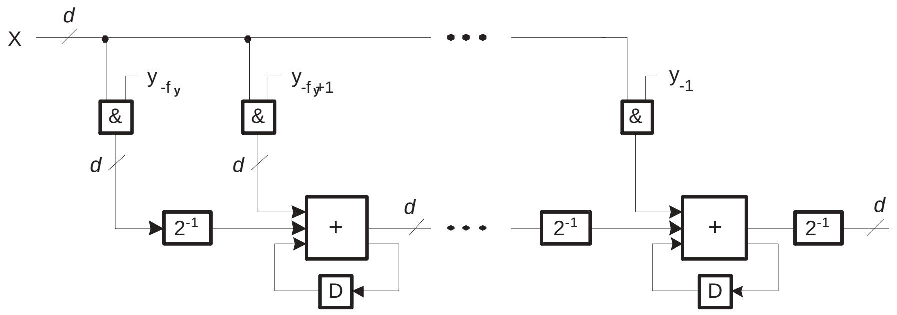

Figure 3.19: Non-causal S/P$_{\rm II}$ multiplier.

(3.29) as

$$\left(\left(\dots\left(X_{-2}\cdot y_{-f_y}\cdot 2^{-1}+X_{-2}\cdot y_{-f_y+1}\right)\cdot 2^{-1}+\dots+X_{-2}\cdot y_{-2}\right)\cdot 2^{-1}+X_{-2}\cdot y_{-1}\right)\cdot 2^{-1}. \tag{3.32}$$

Since the $2^{-1}$ shift cannot be performed the S/P$_{\rm II}$ multiplier can only be realized by inserting pipelining levels as shown in Fig. 3.20. The resulting multiplier structure

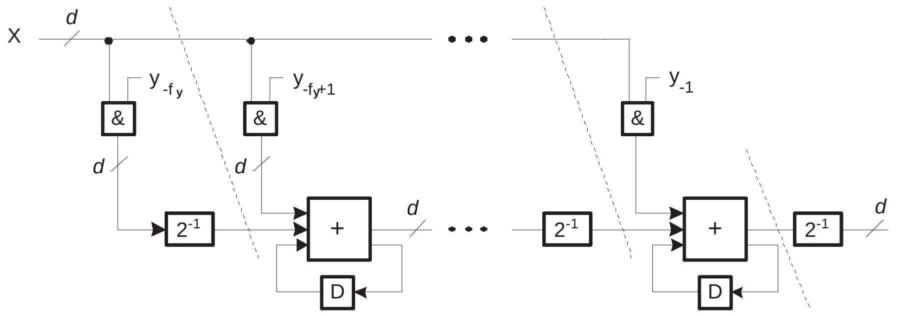

Figure 3.20: Pipelined S/P$_{\rm II}$ multiplier.

is shown in Fig. 3.21. The latency of the S/P$_{\rm II}$ multiplier, when considering adders with no internal pipelining, becomes

$$L=\left(\frac{d-1}{d}f_{\rm y}\right)T_{\rm cp}. \tag{3.33}$$

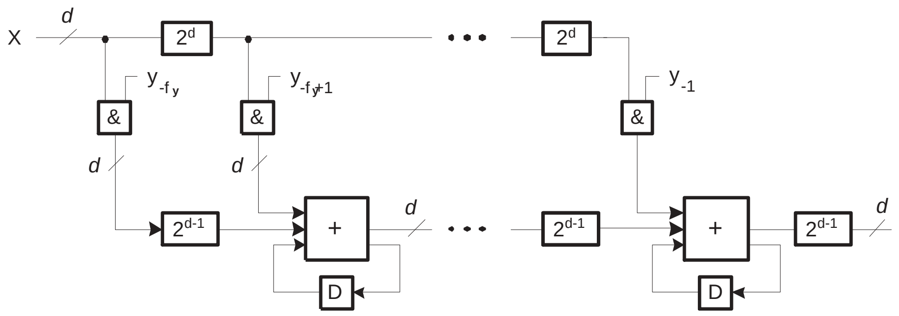

Figure 3.21: $S/P_{\rm II}$ multiplier.

### 3.5.2 Digit-Serial/Parallel Multiplier Based on Shift-Accumulation

A well known parallel multiplier is the shift-and-add multiplier [52]. In general, the shift-and-add multiplier generates the partial products sequentially and accumulates them successively as they are generated. This approach can be used to implement a digit-serial multiplier [17]. We will denote the digit-serial shift-and-add multiplier DSAAM.

The DSAAM basically consists of three parts. A partial product generator, which is implemented using AND-gates. An accumulator that adds the partial products which have the same significance level with carry signals from the preceding accumulation. The accumulator is implemented with [illegible] (CSAs). Depending on how the accumulator is implemented a CSA structure with several levels is used to reduce the number of operands. Finally, a carry-propagating adder is needed. In [17], it is shown that the total number of full adders, for any given of size of operands, is constant. The DSAAM structure is shown in Fig. 3.22. Let us consider an example, where $f_{\rm y} + 1 = 4$ and $d = 2$, in order to illustrate how the multiplier is implemented. The multiplier operands can then be expressed as

$$Y = y_0 \cdot 2^0 + y_{-1} \cdot 2^{-1} + y_{-2} \cdot 2^{-2} + y_{-3} \cdot 2^{-3}, \tag{3.34}$$

and

$$X = \begin{bmatrix} 2^0 & 2^{-1} \end{bmatrix} \cdot \left( \begin{bmatrix} x_0 \\ x_{-1} \end{bmatrix} + \begin{bmatrix} x_{-2} \\ x_{-3} \end{bmatrix} \cdot 2^{-2} + \begin{bmatrix} x_{-4} \\ x_{-5} \end{bmatrix} \cdot 2^{-4} \right). \tag{3.35}$$

After partial product generation the accumulation step can be expressed as

$$X \cdot Y = \begin{bmatrix} 2^0 & 2^{-1} \end{bmatrix} \cdot \left( \begin{bmatrix} x_0 y_0 & x_0 y_{-1} & x_0 y_{-2} & x_0 y_{-3} & 0 & 0 & 0 & 0 \\ x_{-1} y_0 & x_{-1} y_{-1} & x_{-1} y_{-2} & x_{-1} y_{-3} & 0 & 0 & 0 & 0 \end{bmatrix} + \right.$$

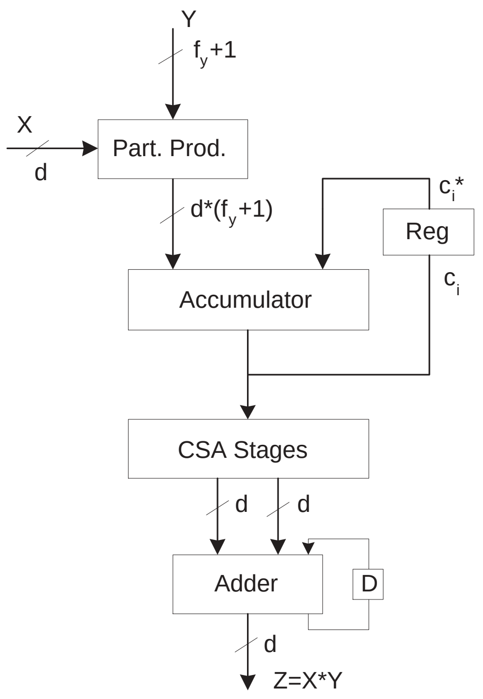

Figure 3.22: Digit-serial shift-and-add multiplier.

$$\begin{bmatrix} 0 & 0 & x_{-2}y_0 & x_{-2}y_{-1} & x_{-2}y_{-2} & x_{-2}y_{-3} & 0 & 0 \\ 0 & 0 & x_{-3}y_0 & x_{-3}y_{-1} & x_{-3}y_{-2} & x_{-3}y_{-3} & 0 & 0 \end{bmatrix} +$$

$$\left.\begin{bmatrix} 0 & 0 & 0 & 0 & x_{-4}y_0 & x_{-4}y_{-1} & x_{-4}y_{-2} & x_{-4}y_{-3} \\ 0 & 0 & 0 & 0 & x_{-5}y_0 & x_{-5}y_{-1} & x_{-5}y_{-2} & x_{-5}y_{-3} \end{bmatrix}\right) \cdot \begin{bmatrix} 2^0 \\ 2^{-1} \\ \vdots \\ 2^{-7} \end{bmatrix}. \qquad (3.36)$$

The partial product generator will generate $d \cdot (f_{\mathrm{y}} + 1)$ partial products each clock cycle. In this example three digits are studied resulting in three partial products matrices as seen in (3.36). The accumulator will add all products with the same significance level with incoming carries from the preceding accumulation. Note that the $d$ least significant bits are always fed forward. The accumulator can be implemented using a one-level CSA structure as seen in Fig. 3.23. The notation $c_i^*$ is explained in Fig. 3.22. A CSA structure is required in order to reduce the number

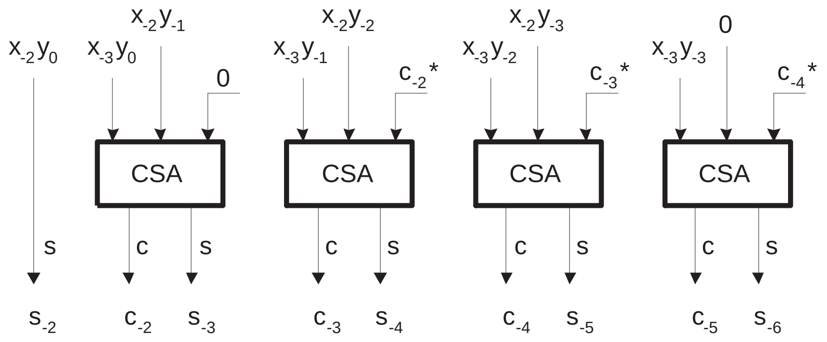

Figure 3.23: Accumulator.

of inputs to the final adder to $2d$. In this example, the CSA structure becomes quite simple as seen in Fig. 3.24. The resulting output from the CSA structure can now be shifted and added using any of the carry-propagating adders presented in the former chapter.

The DSAAM can easily be modified to handle two's complement numbers. By inverting the partial products containing the sign bit of the coefficient and setting the register corresponding to that significance to one, a two's complement coefficient can be used [17]. The input data to the DSAAM can be sign extended in order to get the correct behavior.

The DSAAM will contain two feedback loops: the carry recursion in the accumulator and the the carry loop in the carry propagating adder. The longest logic path of these two loops will determine $T_{\mathrm{cp}}$. In general, a good approach is

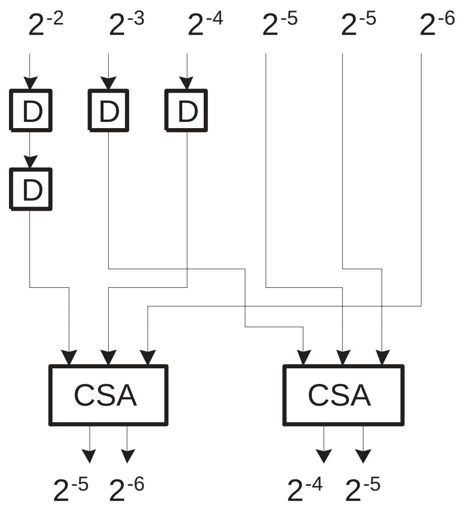

Figure 3.24: CSA structure.

to try to equalize the logic path in these two loops. Since the accumulator can be implemented with a logic path of one CSA the multiplier can be pipelined to the bit level. This can be done provided that the carry-propagating adder also can be pipelined to the bit level.

## 3.5.3 A Pipelined Digit-Serial/Parallel Multiplier

As described earlier in this chapter, the $\mathrm{S/P_I}$ multiplier cannot be pipelined to the bit level. This can be a restriction in some high-speed systems. Several attempts have been made to realize a serial-parallel multiplier that allows arbitrary pipelining. In [8], a digit-serial multiplier that can be pipelined to the bit level was presented. This multiplier requires both the multiplier and the multiplicand to be digits, making it less attractive in fixed coefficient filter design, since the coefficient cannot be used to optimize the amount of hardware [54].

In [43], a digit-serial/parallel multiplier that can be pipelined to the bit level was presented. We will denote this multiplier $\mathrm{S/P_{pipe}}$. The behavior of the multiplier is shown using an example where $f_\mathrm{x} = 15$ and $f_\mathrm{y} = 7$:

$$X = \underbrace{\begin{bmatrix} 2^0 & 2^{-1} & 2^{-2} & 2^{-3} \end{bmatrix}}_{2_x^*} \cdot \left( \begin{bmatrix} x_0 \\ x_{-1} \\ x_{-2} \\ x_{-3} \end{bmatrix} + \begin{bmatrix} x_{-4} \\ x_{-5} \\ x_{-6} \\ x_{-7} \end{bmatrix} \cdot 2^{-4} + \begin{bmatrix} x_{-8} \\ x_{-9} \\ x_{-10} \\ x_{-11} \end{bmatrix} \cdot 2^{-8} + \dots \right.$$

$$+ \begin{bmatrix} x_{-12} \\ x_{-13} \\ x_{-14} \\ x_{-15} \end{bmatrix} \cdot 2^{-12} \Bigg), \tag{3.37}$$

and

$$Y = \underbrace{\begin{bmatrix} y_0 & y_{-1} & \cdots & y_{-7} \end{bmatrix}}_{Y^*} \cdot \underbrace{\begin{bmatrix} 2^0 \\ 2^{-1} \\ \vdots \\ 2^{-7} \end{bmatrix}}_{2^*_y}. \tag{3.38}$$

Multiplying $X$ with $Y$ will generate

$$X \cdot Y = 2^*_x \cdot \left( \begin{bmatrix} x_0 Y^* \\ x_{-1} Y^* \\ x_{-2} Y^* \\ x_{-3} Y^* \end{bmatrix} + \begin{bmatrix} x_{-4} Y^* \\ x_{-5} Y^* \\ x_{-6} Y^* \\ x_{-7} Y^* \end{bmatrix} \cdot 2^{-4} + \begin{bmatrix} x_{-8} Y^* \\ x_{-9} Y^* \\ x_{-10} Y^* \\ x_{-11} Y^* \end{bmatrix} \cdot 2^{-8} + \ldots \right.$$

$$\left. + \begin{bmatrix} x_{-12} Y^* \\ x_{-13} Y^* \\ x_{-14} Y^* \\ x_{-15} Y^* \end{bmatrix} \cdot 2^{-12} \right) \cdot 2^*_y. \tag{3.39}$$

Extracting $2^{-12}$ in (3.39), we can write the expression as

$$X \cdot Y = 2^*_x \cdot \left( \left( \begin{bmatrix} x_0 Y^* & 0 \cdots 0 \\ x_{-1} Y^* & 0 \cdots 0 \\ x_{-2} Y^* & 0 \cdots 0 \\ x_{-3} Y^* & 0 \cdots 0 \end{bmatrix} + \begin{bmatrix} 0 \cdots 0 & x_{-4} Y^* & 0 \cdots 0 \\ 0 \cdots 0 & x_{-5} Y^* & 0 \cdots 0 \\ 0 \cdots 0 & x_{-6} Y^* & 0 \cdots 0 \\ 0 \cdots 0 & x_{-7} Y^* & 0 \cdots 0 \end{bmatrix} + \right. \right.$$

$$\left. \left. + \begin{bmatrix} 0 \cdots 0 & x_{-8} Y^* & 0 \cdots 0 \\ 0 \cdots 0 & x_{-9} Y^* & 0 \cdots 0 \\ 0 \cdots 0 & x_{-10} Y^* & 0 \cdots 0 \\ 0 \cdots 0 & x_{-11} Y^* & 0 \cdots 0 \end{bmatrix} + \begin{bmatrix} 0 \cdots 0 & x_{-12} Y^* \\ 0 \cdots 0 & x_{-13} Y^* \\ 0 \cdots 0 & x_{-14} Y^* \\ 0 \cdots 0 & x_{-15} Y^* \end{bmatrix} \right) \right) \cdot \underbrace{\begin{bmatrix} 2^0 \\ 2^{-1} \\ \vdots \\ 2^{-19} \end{bmatrix}}_{72^*}. \tag{3.40}$$

The zeros are introduced to keep the correct significance level for the matrices. The total number of zeros introduced in each matrix is $f_{\mathrm{x}} + 1 - d$. The length of $2^*$ is $f_{\mathrm{y}} + f_{\mathrm{x}} + 2 - d$. The S/P$_{\mathrm{pipe}}$ uses a number of pipes to add the rows of (3.40). A

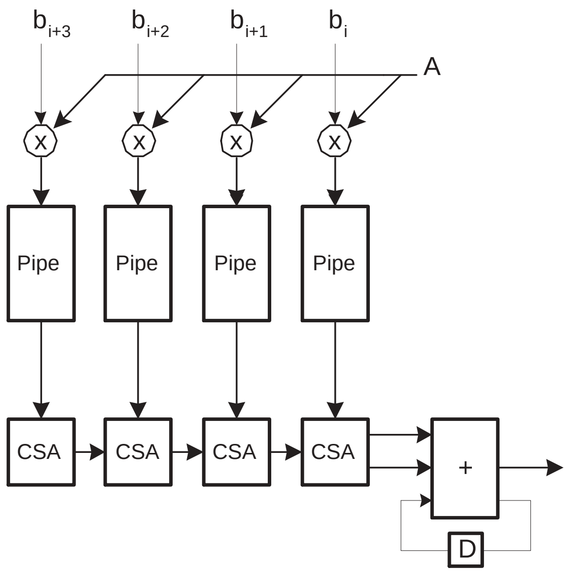

Figure 3.25: S/P$_{\text{pipe}}$ multiplier with $d = 4$.

description of the multiplier for $d = 4$ is shown in Fig. 3.25. As can be seen from Fig. 3.25, the multiplier is fed serially while the multiplicand is parallel. Each pipe is a systolic array of full adders. The architecture of the pipe depends on the size of the multiplicand. The output from each pipe is a digit in carry-save representation. These digits are then added using 2-level CSA nets. Finally an adder with carry recursion is used to obtain the correct digit in two's complement form. Naturally this adder must also be pipelined to the bit level.

# Chapter 4

# Scheduling of Digit-Serial Processing Elements

## 4.1 Introduction

The throughput of a digital filter is an important implementation constraints. There are several factors that will determine the throughput of the filter algorithm, such as the choice of processing elements, pipelining and scheduling. Different processing elements were discussed in the former chapter. In this chapter, we will consider pipelining and scheduling of algorithms using digit-serial arithmetics. That is, the data flow in the algorithm consists of digits. We assume LSD-first computation (see Section 1.4.4). Furthermore, we will only consider recursive algorithms.

The topics in this chapter are considered in the following publications: Digit-serial scheduling is considered in Publication IV. Cyclic scheduling is considered in Publication I and VI. In Publication V, retiming is studied as a method to decrease the power consumption, see Section 4.5.

In a recursive digital system, pipelining cannot be used to increase the throughput of the algorithm. This is because all pipelining levels will affect both the direct path and the recursive path of the system, forcing us to decrease the input signal rate. This is not the case in a non-recursive system, where pipelining, in theory, always will increase the throughput of the system.

In a digit-serial recursive system, the feedback loop contains $(f_{\mathrm{x}}+1)/d$ registers to ensure correct timing between the input signal and the state signals. Using [illegible], these registers can be moved into the system to decrease the arithmetic critical path, which is shown in Fig. 4.1. In some systems, the operation shown in Fig. 4.1 may also require pipelining at the output [52]. The number of registers in the feedback loop is determined by the wordlength of the input signal and the chosen digit size. Increasing the digit size will lead to fever registers and, hence,

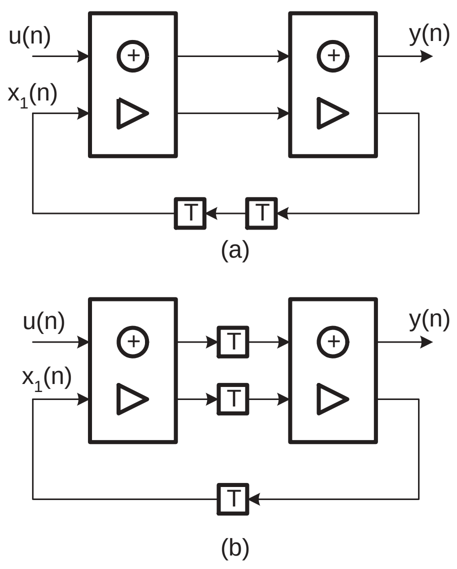

Figure 4.1: Retiming of a recursive digit-serial system. a) Before retiming and b) After retiming.

fewer flip-flops can be used. In the case where $d = f_{\mathrm{x}} + 1$, the system is bit-parallel, and only one register section can be retimed. The fact that only a limited number of registers can be used to decrease the arithmetic critical path in a recursive system further emphasizes the need for low-latency processing elements. It is important to note that retiming of the registers in the feedback loop will always increase the throughput of the algorithm, provided that the clock frequency is properly increased. In general, studying the algorithmic critical path of an algorithm is sufficient when analyzing the throughput of an algorithm. The algorithmic critical path for the general-order LDI/LDD allpass filter consist of one multiplier and three adders, i.e.,

$$T_{\mathrm{cp_a}} = T_{\mathrm{mult}} + 3T_{\mathrm{add}}. \tag{4.1}$$

This algorithmic critical path can also be obtained for other high-performance recursive digital filters, such as the WD allpass filter [54].

## 4.2 Computational Properties of Digit-Serial Processing Elements

In Section 1.4.5, the computation graph was introduced. We will use computation graphs to describe the timing behavior of the digit-serial processing elements. Let us consider a digit-serial multiplier with no pipelining, where the parallel coefficient is $f_y + 1$ bits. Assuming that we multiply $m$ digits (see (3.17)), the data flow of the multiplier can be described by Fig. 4.2. A a lower bound on the the latency is

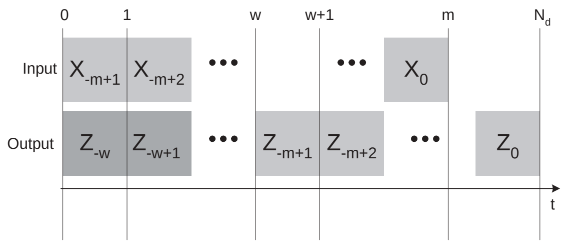

Figure 4.2: Computation graph for a general digit-serial multiplier with no pipelining.

$w$ clock cycles and $N_d$ is the minimum number of clock cycles to obtain all bits of the product. The computation time of the multiplier can be expressed as

$$T_{s_{\text{mult}}} = N_{\text{d}} \cdot T_{\text{clk}} = \left\lceil m + \underbrace{\frac{f_{\text{y}}}{d}}_{w} \right\rceil \cdot T_{\text{cp}}. \tag{4.2}$$

A general digit-serial adder can be described using the computation graph in Fig. 4.3. It is important to note, that the computation graphs in Fig. 4.2 and Fig. 4.3

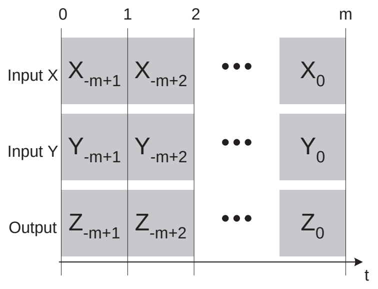

Figure 4.3: Computation graph for a general digit-serial adder with no pipelining.

assume that no pipelining is introduced. Introducing pipelining will increase the number of clock cycles required to obtain the product, but it will also, hopefully, decrease the arithmetic critical path.

## 4.3 Single Interval Scheduling

In this section, we will consider the case where a recursive filter algorithm with an algorithmic critical path corresponding to (4.1) is scheduled over one sample interval using [illegible] [27]. When using isomorphic mapping, a processing element is assigned to each operation in the algorithm [54]. The main advantage of isomorphic mapping is that it allows adaption of each processing element [54] to the computational workload.

We will use a computation graph in order to show how retiming affects the sample period of the system in (4.1). Let us first consider the case when retiming has not been performed and, thus, no registers are introduced between the processing elements. The resulting computation graph is shown in Fig. 4.4. The number of

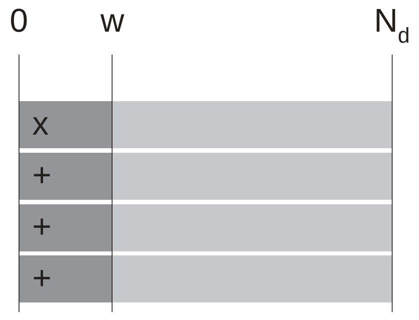

Figure 4.4: Computation graph with no retiming.

clock cycles required to compute a sample is determined by the multipliers computation time. This is due to the fact that multiplication increases the wordlength of the internal data in the digital algorithm. The sample period can, in this case, expressed as

$$T_{\mathrm{s}} = N_{\mathrm{d}} \cdot T_{\mathrm{cp}}, \tag{4.3}$$

where $N_{\mathrm{d}}$ is the number of data digits of the product and $T_{\mathrm{cp}}$ is the arithmetic critical path for the system (equal to $T_{\mathrm{clk}}$). Note that the number of internal data digits will be larger than the number of input data digits. We can write $N_{\mathrm{d}}$ as

$$N_{\mathrm{d}} = \underbrace{\frac{f_{\mathrm{x}}+1}{d}}_{m} + \underbrace{\frac{f_{\mathrm{y}}}{d}}_{w}, \tag{4.4}$$

where $m$ is the number of input digits and $w$ is the latency.

Since no registers was introduced in the former schedule, the arithmetic critical path will, most certainly, be long. To avoid this we can use retiming as described in Section 4.1.

Let us consider the case where a register section is introduced after each processing element. The modified computation graph is shown in Fig. 4.5, where two

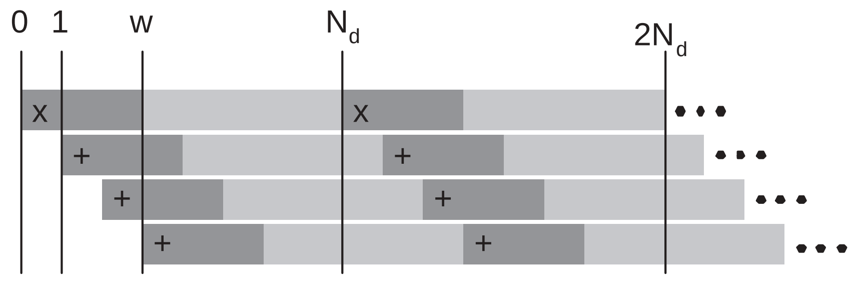

Figure 4.5: Computation graph with register sections after each processing element.

consecutive samples are displayed. Clearly, the computation of the second sample can begin before the computation of the first sample is complete. The part of the computation graph between $N_\mathrm{d}$ and $2N_\mathrm{d}$ can, therefore, be used to describe the throughput of the algorithm. To this end, the computation graphs will only contain one sample henceforth.

From Fig. 4.5 we have that a sample will take $N_\mathrm{d}$ clock cycles to produce. Note that this is the same number of clock cycles as the schedule in Fig. 4.4. The sample period in Fig. 4.5 will, however, be lower since $T_\mathrm{cp}$ is shorter.

### 4.3.1 Scheduling Using Ripple-Carry Adders

As stated earlier, it is hard to generalize any scheduling theories, since information about the processing elements is required. In the special case where DSRCAs and $\mathrm{S/P_I}$ multipliers are utilized (see Sections 3.2.3 and 3.5.1) some theories can, however, be established. From (3.12), a theoretical estimate of the latency for the DSRCA is given. We can use this estimate when studying scheduling of DSRCAs and $\mathrm{S/P_I}$ multipliers.

When connecting several processing elements, the arithmetic critical path will increase. Assuming that $k$ DSRCAs are connected in series, the arithmetic critical path becomes:

$$T_\mathrm{cp} = (k + d)T_\mathrm{fa}. \tag{4.5}$$

This is shown in Fig. 4.6, where the arithmetic critical path is highlighted.

Connecting a $\mathrm{S/P_I}$ and a DSRCA will result in $T_\mathrm{cp} = (d + 2)T_\mathrm{fa}$, as seen in Fig. 4.7. Expression (4.5) can therefore be used when connecting DSRCAs and/or $\mathrm{S/P_I}$s.

In order to limit $T_\mathrm{cp}$ to $(d+1)T_\mathrm{fa}$, which is the minimum $T_\mathrm{cp}$ when using these types of processing elements, register sections must be introduced between the

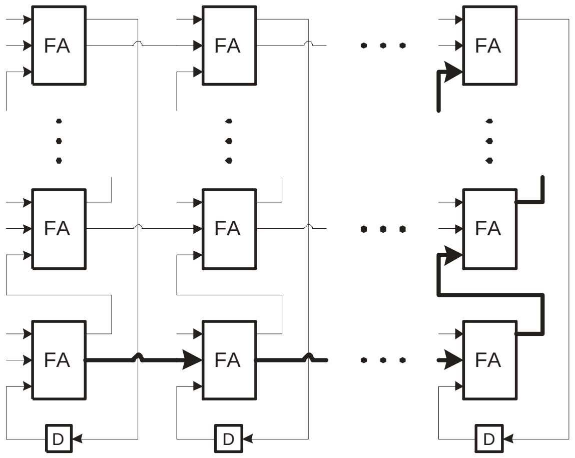

Figure 4.6: $T_{\mathrm{cp}}$ when connecting DSRCAs in series.

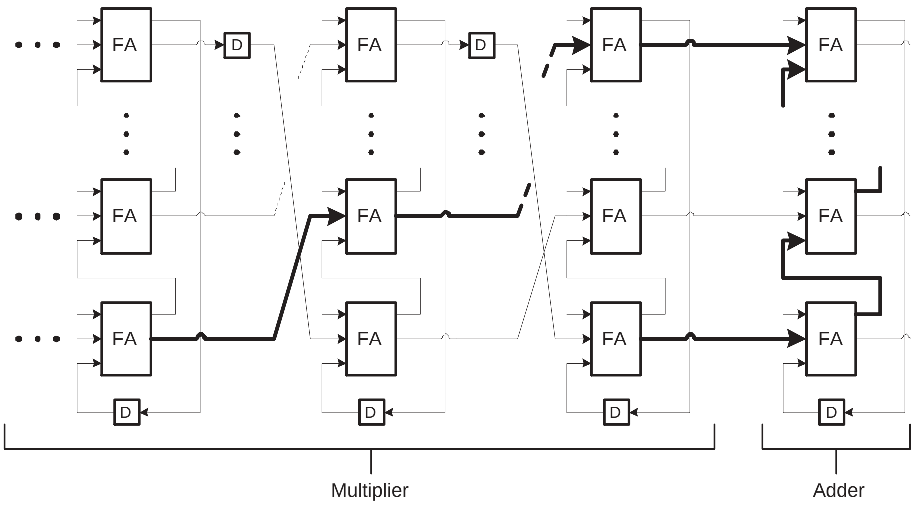

Figure 4.7: $T_{\mathrm{cp}}$ when connecting a $\mathrm{S/P_I}$ and DSRCA in series.

processing elements. As discussed in the former section, this can be achieved by utilizing retiming.

To summarize, in the special case where DSRCAs and $\mathrm{S/P_I}$ multipliers are utilized, the theoretical estimate (3.12) can be used when considering scheduling of processing elements. This is possible since there exists a simple relation between the digit size and the arithmetic critical path for these processing elements. Another reason a theoretical estimate can be found is that register sections are normally only introduced between the processing elements and a simple expression for the arithmetic critical path when connecting several processing elements can be found (4.5). When other processing elements are used, scheduling becomes harder, as we will see in the next section.

### 4.3.2 Scheduling Using Bit-Level Pipelined Processing Elements

Scheduling processing elements that can be pipelined to the bit level is similar to the approach presented in the former section. The difference is that in order to limit the arithmetic critical path, pipelining registers must usually be introduced within the processing elements. This makes it harder to derive a theoretical estimate of the final performance. The convenient relation between digit size and arithmetic critical path that exists in the ripple-carry adder case does not exist when using for example carry-look-ahead adders. Let us illustrate this by an example.

Assume that $f_y = 12$, $f_x = 23$, and the digit size is $d = 4$. Let us consider the case where the DSAAM and the DSHA are utilized. Both these processing elements can be pipelined to the bit level. In order to obtain $T_{\mathrm{cp}} = 2T_{\mathrm{fa}}$ (bit-level pipelining with $T_{\mathrm{D}} \approx T_{\mathrm{fa}}$) the DSHA must be pipelined two times and the DSAAM requires eight pipelining levels. The resulting schedule is shown in Fig. 4.8(a). Clearly,

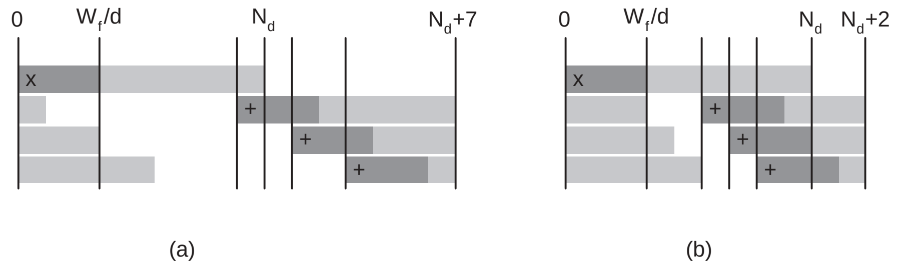

Figure 4.8: Scheduling for $d = 4$ with (a) $T_{\mathrm{cp}} \approx 2T_{\mathrm{fa}}$ or (b) $T_{\mathrm{cp}} \approx 3T_{\mathrm{fa}}$.

pipelining to the bit level will result in a schedule greater than $N_{\mathrm{d}}$ clock cycles. The sample period in this case becomes

$$T_{\mathrm{s}} \approx 32T_{\mathrm{fa}}. \tag{4.6}$$

By allowing a longer arithmetic critical path the latency of the system will decrease. An arithmetic critical path of $T_{\mathrm{cp}} \approx 3T_{\mathrm{fa}}$ would require the DSHA to be pipelined one time and the DSAAM to be pipelined five times. The resulting schedule is shown in Fig. 4.8(b). The sample period when using this schedule becomes

$$T_{\mathrm{s}} \approx 33T_{\mathrm{fa}}. \tag{4.7}$$

Nothing conclusive can be said about which schedule to use, which implies that better information about the arithmetic critical path and latency of the processing elements is required. This information may, however, require implementation of the processing elements. It will sometimes be necessary to implement the processing elements with different degrees of pipelining.

It is important to note that the pipelining degree will not only affect the throughput of the algorithm. The implementation area and power consumption will also be affected.

## 4.4 Digit-Serial Cyclic Scheduling

In the former sections, we studied single-interval scheduling. A scheduling transformation known as [illegible] or [illegible] can be used to increase the throughput of the system. Cyclic scheduling has been studied extensively over the years [54], [18], [52], [32]. In [25], cyclic scheduling of a second-order bit-serial LDI allpass filter was considered and the filter was implemented in a FPGA.

The throughput of a recursive algorithm is restricted by $T_{\min}$, according to (1.20). When the algorithmic critical path is longer than $T_{\min}$, scheduling transformations must be performed in order to obtain the minimal sample period. These transformations are also known as cyclic scheduling [52]. A cyclic schedule can be obtained by unfolding [44] the filter algorithm $m$ times. The concept is shown in Fig. 4.9, where a filter structure $F$ is unfolded $m$ times. Cyclic scheduling, therefore, requires the filter algorithm to be scheduled over $m$ sample periods [54]. The value of $m$ is given by

$$m \geq \left\lceil \frac{T_{\mathrm{s}}}{T_{\min}} \right\rceil . \tag{4.8}$$

From (4.8) it is clear that $m$ needs to be an integer and naturally a small value is desired since increasing $m$ also leads to more hardware. In addition to finding $m$, retiming of the registers in the recursive loop must also be solved. An integer value of $m$ can be found for the second-order LDI allpass filter by rewriting (4.8) using (1.20) and (4.3). Note that the latency in clock cycles for the multiplier and adder is $f_{\mathrm{y}}/d$ and 0, respectively. We introduce the variable $p$ as the number of register sections that will be introduced in each filter structure (1, 2…m), see Fig.

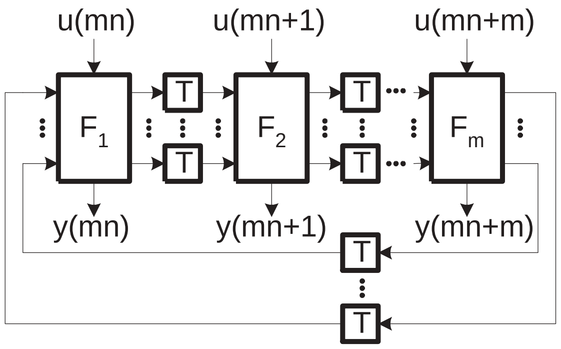

Figure 4.9: Cyclic scheduling.

4.9. This gives us

$$m \geq \left\lceil \frac{f_\mathrm{x} + f_\mathrm{y} + 1}{f_\mathrm{y} + pd} \right\rceil . \tag{4.9}$$

In the case where $f_\mathrm{x}$ and $f_\mathrm{y}$ are fixed an integer value of $m$ can be found by solving (4.9) for different values of $p$.

## 4.5 Using Retiming to Reduce Power Consumption

Retiming of the registers in the feedback loop can be used to lower the power consumption of a digital filter. It is well known that glitches can cause unnecessary switching of a logic network, which will increase the dynamic power consumption [7], [36]. A method to reduce the amount of glitches is to insert registers in the system since glitches, at the input of a register, will not propagate to the output, i.e., register outputs are practically glitch free [36]. This can be done in recursive digital filters by utilizing retiming. A study on how to reduce the number of glitches in a digit-serial second-order LDI allpass filter can be found in Publication V.

## 4.6 Control Unit

The number of reset signals required when implementing a digital filter depends on how the filter is scheduled. In the case where all processing elements execute in parallel only one reset signal is required, see Fig. 4.4. When this is not the case several reset signals must be generated (Fig. 4.5). Furthermore, sign-extension units also require additional control signals. All these signals must be generated by a control unit. Obviously, the control unit should not degrade the filter performance.

It should, therefore, be kept as small as possible with short arithmetic critical path. In [16], a control unit based on an HSLC counter was presented. The HSLC counter is a resource-efficient non-binary counter that is well suited for digital filter implementation [2]. The counter follows a pattern of $2^k - 2$ states, where $k$ is the number of counter bits. A 4-bit HSLC counter will, for example, generate a pattern of 14 values. It will generate two states fewer than a typical binary counter, however, a short arithmetic critical path compensates for that. In Fig. 4.10, a control unit

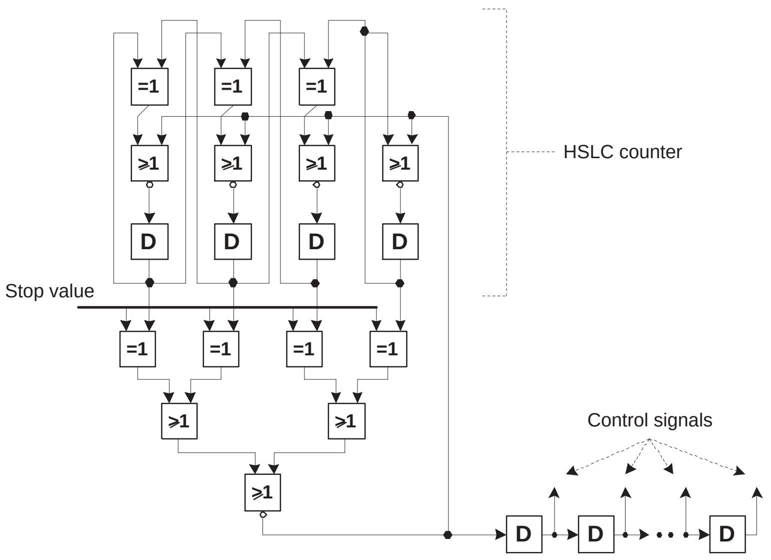

Figure 4.10: Control unit using a 4-bit HSLC counter.

with a 4-bit HSLC counter is shown. The counter will reset itself upon reaching a fixed predetermined stop value. Thus, the only input signals required by the control unit is a global reset and the clock.

# Chapter 5

# Conclusions

In this work, implementation of recursive digital filters using digit-serial arithmetics was studied. The filter considered was the LDI allpass filter structure, although the theories can be applied to other recursive systems. The LDI allpass filter has, like the WD filter, several properties which makes it well suited for implementation. It has a short minimal sample period and low coefficient sensitivity. The LDI allpass filter can, however, be implemented with less hardware resources compared to a corresponding WD filter.

Digit-serial arithmetics was considered throughout this work. Traditionally, digit-serial processing elements are derived using unfolding. However, this leads to a loop in each processing element and, hence, the filters cannot be pipelined arbitrarily. To circumvent this, different digit-serial adders and multipliers which can be pipelined to the bit-level have been studied in this thesis. The main objective in order to make bit-level pipelining possible is to minimize the latency in any recursive part of the processing element. It is also important that the processing elements have a low latency to yield a high filter throughput. A new digit-serial hybrid adder, with these properties, was presented and it was shown that the adder is an interesting candidate in high-throughput recursive systems. It was also shown that a digit-serial multiplier based on shift accumulation can be implemented with a low latency, thus, making it well suited for the systems studied in this thesis.

In general, the main drawback with using bit-level pipelined processing elements is that the current consumption becomes high. These types of processing elements should, therefore, be considered in digit-serial high throughput applications where a higher power consumption can be tolerated. Bit-level pipelined processing element could also be considered in systems where voltage scaling is utilized. The high throughput can then by traded for lower power consumption. Voltage scaling is, however, beyond the scope of this work.

Different scheduling transformations which enhances the filter performance was considered in this work. Cyclic scheduling can be used to increase the throughput of a digit-serial recursive filter and in this thesis cyclic scheduling of a second-order digit-serial LDI allpass filter was studied. Cyclic scheduling will increase the throughput of a digit-serial system significantly especially for smaller digit-sizes.

Retiming was considered as a method to decrease the amount of glitches in a second-order LDI allpass filter. It was shown that retiming, if it is properly performed, can be used to decrease the power consumption.

## 5.1 Future Work

There are several suitable topics for future research concerning the digit-serial design process. An advantage with using unfolding to derive digit-serial processing elements is that there exist a simple theoretical relation between digit-size and latency. Finding a similar relation for processing elements that can be pipelined to an arbitrary degree would simplify the design process of high-throughput digit-serial filters. The studies made in Publication V concerning retiming should be extended to also comprise processing elements with an arbitrary pipelining. These studies should also include identifying critical nodes in the filter structure where the probability of glitches is high.

# Bibliography

[1] A. Aggoun, M. K. Ibrahim, and A. Ashur, “Design methodology for subdigit pipelined digit-serial IIR filters,” in [illegible], Vol. 68, No. 1, pp. 73–86, 1998.

[2] R. Ahmed and D. Perreault, “High speed low connectivity (HSLC) counters (using cellular automata concept,” in [illegible], Vol. 1, pp. 433–436, 1992.

[3] A. S. Ashur, M. K. Ibrahim, and A. Aggoun, “Systolic digit-serial multiplier,” [illegible], Vol. 143, pp. 14–20, 1996.

[4] A. I. Barkin, “Sufficient conditions for the abscence of auto-oscillations in pulse-systems,” [illegible], Vol. 31, pp. 942–946, June 1970.

[5] R. P. Brent and H. T. Kung, “A regular layout for parallel adders,” [illegible], Vol. C-31, No. 3, pp. 260–264, March 1982.

[6] L. T. Bruton and D. A. Vaughan-Pope, “Synthesis of digital ladder filters from LC filters,” [illegible], Vol. CAS-23, No. 6, pp. 395–402, June 1976.

[7] A. P. Chandrakasan and R. W. Brodersen “Minimizing power consumption in digital CMOS circuits,” [illegible], Vol. 83, No. 4, pp. 498–523, 1995.

[8] Y.-N. Chang, J. H. Satyanarayana, and K. K. Parhi, “Systematic design of high-speed and low-power digit-serial multipliers,” [illegible], Vol. 37, No. 3, pp. 420–431, March 2002.

[9] T. Claasen, W. F. G. Mecklenbräeuker, and J. B. H. Peek, “Frequency domain criteria for the abscence of zero-input limit cycles in nonlinear discrete-time systems, with applications to digital filters,” [illegible], Vol. CAS-22, No. 3, pp. 232–239, March 1975.

[10] R. W. Davis, N. Zhang, K. Camera, D. Markovic, T. Smilkstein, J. M. Ammer, E. Yeo, S. Augsburger, B. Nikolic, and R. W. Brodersen, “A design

environment for high-throughput low-power dedicated signal processing systems," [illegible], Vol. 45, No. 12, pp. 1585–1596, December 1998.

[11] Design Compiler Reference Manual, v1999.10, Synopsys 1999.

[12] Envisia Silicon Ensemble Place-and-Route Reference, Product version 5.3, Cadence Design Systems, Inc., November 1999.

[13] A. Fettweis, H. Levin, and A. Sedlmeyer "Wave digital lattice filters," [illegible], Vol. 2, pp. 203–211, June 1974.

[14] A. Fettweis and K. Meerkötter "Suppression of parasitic oscillations in wave digital filters," [illegible], Vol. CAS – 22, pp. 239–246, March 1975.

[15] L. Gazsi, "Explicit formulas for lattice wave digital filters," [illegible], Vol. CAS–32, No. 1, pp. 68–88, January 1985.

[16] O. Gustafsson and L. Wanhammar, "Maximally fast scheduling of bit-serial lattice wave digital filters using three-port adaptor allpass sections," in [illegible], NORSIG'2000, Norrköping, Sweden, June 2000.

[17] O. Gustafsson and L. Wanhammar, "Bit-level pipelinable general and fixed coefficient digit-serial/parallel multipliers based on shift-accumulation," in [illegible] 2002, Vol. 2, pp. 493–496, 15-18 September 2002.

[18] O. Gustafsson, "Contributions to low-complexity digital filters," [illegible], Dissertation No. 837, Linköping University, Linköping, Sweden, 2003.

[19] L. Harnefors, J. Holmberg, and S. Signell, "Suppression of overflow limit cycles in LDI allpass/lattice filters," [illegible], 1999, Vol. 47, No. 1, pp. 594–598, April 2000.

[20] R. I. Hartley and P. F. Corbett, "A digit-serial silicon compiler," in [illegible], pp. 646–649, 12-15 June 1988.

[21] R. Hartley and P. Corbett, "A digit-serial compiler operator library," in [illegible], Vol. 1, pp. 641–646, May 1989.

[22] R. Hartley and P. Corbett, "Digit-serial processing techniques," [illegible], Vol. 37, pp. 707–719, June 1990.

[23] R. I. Hartley and K. K. Parhi, [illegible] Kluwer Academic Publishers, 1995.

[24] J. Holmberg, L. Harnefors, and S. Signell, “Stability analysis of the second-order lossless digital integrator allpass filter,” in [illegible] 2000, Vol. 1, pp. I-367–I-370, Geneva, Switzerland, May 1999.

[25] J. Holmberg, L. Harnefors, K. Landernäs, and S. Signell, “Computational properties of LDI/LDD lattice filters,” in [illegible], Vol. 2, pp. 685–688, Sydney, Australia, May 2001.

[26] J. Holmberg, K. Landernäs, and M. Vesterbacka, “Implementation aspects of second-order LDI/LDD allpass filters,” in [illegible], Vol. 1, pp. 237–240, Espoo, Finland, August 2001.

[27] J. Holmberg, “On design, analysis and implementation of LDI/LDD lattice filters,” [illegible], Dissertation No. 1, Mälardalen University, Västerås, Sweden, 2002.

[28] Z. Huang and M. D. Ercegovac, “Effect of wire delay on the design of prefix adders in deep-submicron technology,” in [illegible], 2000, Vol. 2, pp. 1713–1717, 2000.

[29] C.-Y. Hung and P. Landman, “Compact inverse discrete cosine transform circuit for MPEG video decoding,” in [illegible], pp. 364–373, November 1997.

[30] M. J. Irwin and R. M. Owens, “Design issues in digit serial signal processors,” in [illegible], Vol. 1, pp. 441–444, May 1989.

[31] P. Israsena and S. Summerfield, “Bit-level retiming of high-speed digital recursive filters,” in [illegible], Vol. 2, pp. 19–24, October 2002.

[32] M. Karlsson, “Implementation of Digit-Serial Filters,” [illegible], Dissertations No. 952, Linköping University, Linköping, Sweden, 2005.

[33] P. M. Kogge and H. S. Stone, “A parallel algorithm for the efficient solution of a general class of recurrence equations,” [illegible], Vol. C-22, No. 8, pp. 786–793, Aug. 1973.

[34] I. Koren, [illegible] A. K. Peters, Ltd, 2002.

[35] R. E. Ladner and M. J. Fischer, “Parallel prefix computation,” [illegible], Vol. 27, No. 4, pp. 831–838, Oct. 1980.

[36] J. Leijten, J. van Meerbergen, and J. Jess, “Analysis and reduction of gltiches in synchronous networks,” in [illegible] pp. 398–403, 1995.

[37] D. Liu and C. Svensson, “Trading speed for low power by choice of supply and threshold voltages,” [illegible], Vol. 28, No. 1, pp. 10–17, January 1993.

[38] H. Ming, O. Vainio, and M. Renfors, “Digit-serial design of a wave digital filter,” in [illegible] Vol. 1, pp. 542–545, 1999.

[39] S. K. Mitra, [illegible] McGraw-Hill, 2001.

[40] Modelsim Quick Guide, v5.2, Model Technology Inc., 1998.

[41] C. Nagendra, M. J. Irwin, and R. M. Owens, “Area-time-power tradeoffs in parallel adders,” [illegible], Vol. 43, No. 10, pp. 689–702, October 1996.

[42] Nanosim User Guide, Release 2002.03, Synopsys 2002.

[43] O. Nibouche, A. Bouridane, M. Nibouche, and D. Crookes, “A new pipelined digit serial-parallel multiplier,” in [illegible], Vol. 1, pp. 12–15, Geneva, Switzerland, May 2000.

[44] K. K. Parhi, “A systematic approach for design of digit-serial signal processing architectures,” [illegible], Vol. 38, No. 4, pp. 358–375, April 1991.

[45] K. K. Parhi, “Low-energy CSMT carry-generators and binary adders,” [illegible], pp. 450–462, December 1999.

[46] K. K. Parhi, [illegible] Wiley-Interscience, 1999.

[47] K. K. Parhi, “Approaches to low-power implementations of DSP systems,” [illegible], Vol. 48, No. 10, pp. 1214–1224, October 2001.

[48] M. Renfors and Y. Neuvo, “The maximal sample rate of digital filters under hardware speed constraints,” [illegible], CAS-28, No. 3, pp. 196–202, March 1981.

[49] S. R. Signell, T. G. Kouyoumdjiev, K. H. Mossberg, and C. G. L. Harnefors, “Design and analysis of bilinear digital ladder filters,” [illegible], Vol. 28, No. 2, pp. 69–81, February 1996.

[50] High Performance 0.18$\mu$ Standard cell library data book, Rev.2.1, Virtual Silicon Technology, Inc., Sunnyvale, CA, January 2001.

[51] Y. Wang, C. Pai, and X. Song, "The design of hybrid carry-lookahead/carry-select adders," [illegible], Vol. 49, No. 1, pp. 16–24, January 2002.

[52] L. Wanhammar, [illegible] Academic Press, 1998.

[53] H. Veendrick, [illegible] Kluwer academic publishers, 2000.

[54] M. Vesterbacka, "On implementation of maximally fast wave digital filters," [illegible], Dissertations No. 487, Linköping University, Linköping, Sweden, 1997.