Continuous-Time and Companding Digital Signal Processors Using Adaptivity and Asynchronous Techniques

Christos Vezyrtzis

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2013

©2013

Christos Vezyrtzis

All Rights Reserved

Abstract

Continuous-Time and Companding Digital Signal Processors Using Adaptivity and Asynchronous Techniques

Christos Vezyrtzis

The fully synchronous approach has been the norm for digital signal processors (DSPs) for many decades. Due to its simplicity, the classical DSP structure has been used in many applications. However, due to its rigid discrete-time operation, a classical DSP has limited efficiency or inadequate resolution for some emerging applications, such as processing of multimedia and biological signals.

This thesis proposes fundamentally new approaches to designing DSPs, which are different from the classical scheme. The defining characteristic of all new DSPs examined in this thesis is the notion of "adaptivity" or "adaptability." Adaptive DSPs dynamically change their behavior to adjust to some property of their input stream, for example the rate of change of the input. This thesis presents both enhancements to existing adaptive DSPs, as well as new adaptive DSPs.

The main class of DSPs that are examined throughout the thesis are continuous-time (CT) DSPs. CT DSPs are clock-less and event-driven; they naturally adapt their activity and power consumption to the rate of their inputs. The absence of a clock also provides a complete avoidance of aliasing in the frequency domain, hence improved signal fidelity.

The core of this thesis deals with the complete and systematic design of a truly general-purpose CT DSP. A scalable design methodology for CT DSPs is presented. This leads to the main contribution of this thesis, namely a new CT DSP chip. This chip is the first general-purpose CT DSP chip, able to process many different classes of CT and synchronous signals. The chip has the property of handling various types of signals, i.e. various different digital modulations, both synchronous and asynchronous, without requiring any reconfiguration; such property is presented for the first time CT DSPs and is impossible for classical DSPs. As opposed to previous CT DSPs, which were limited to using only one type of digital format, and whose design was hard to scale for different bandwidths and bit-widths, this chip has a formal, robust and scalable design, due to the systematic usage of asynchronous design techniques.

The second contribution of this thesis is a complete methodology to design adaptive delay lines. In particular, it is shown how to make the granularity, i.e. the number of stages, adaptive in a real-time delay line. Adaptive granularity brings about a significant improvement in the line's power consumption, up to 70% as reported by simulations on two design examples. This enhancement can have a direct large power impact on any CT DSP, since a delay line consumes the majority of a CT DSP's power. The robust methodology presented in this thesis allows safe dynamic reconfiguration of the line's granularity, on-the-fly and according to the input traffic.

As a final contribution, the thesis also examines two additional DSPs: one operating the CT domain and one using the companding technique. The former operates only on level-crossing samples; the proposed methodology shows a potential for high-quality outputs by using a complex interpolation function. Finally, a companding DSP is presented for MPEG audio. Companding DSPs adapt their dynamic range to the amplitude of their input; the resulting can offer high-quality outputs even for small inputs. By applying companding to MPEG DSPs, it is shown how the DSP

distortion can be made almost inaudible, without requiring complex arithmetic hardware.

Contents

List of Figures

List of Tables

Acknowledgments

I feel the need to acknowledge many people that helped me throughout the past 30 years of my life. This includes the past almost 7 years that I spent at Columbia University.

Even though I am having trouble figuring out the exact order, I know that I have to start with my parents. They are the ones who raised me, helped me build my good attributes and put up with my bad ones. The fact that they helped so that I can spend my years here stress-free from financial issues is nothing compared to their support and good values (which I believe that have been passed down to me).

I have had the luck and sincere privilege of doing my PhD under the guidance of two amazing advisors. Both Professor Tsividis and Professor Nowick (whom I am mentioning in this order solely by order of advising years) are researchers and personalities that influenced me the most. I am sure that a large part of my future career would not have been feasible without their useful advice, which have shaped both the way I think and the rigor and professionalism with which I approach every matter. I also thank NSF for funding me throughout the duration of my PhD studies, through the following grants awarded to my advisors: NSF CCF-0964606, NSF CCF-0811504 and NSF CCF 07-01766. A special word of thanks should also be addressed to my committee members, Mingoo Seok, Thao Nguyen and Mihai Sanduleanu, for their precious feedback, as well as for their understanding and flexibility throughout the distribution process.

Then come my friends. It's both the ones with whom I spent my years here, as well as those whom we grew up together back home. Both gave me an immense amount of support through the bad times, as well as a good amount of feedback to keep me on the ground during the good times. The list will be made very long if I decide to state all names, but I want to record a special

thanks to Bill, Nick, Eugene and Alex. Also to all my CISL lab mates: Yu Chen and Kagan Irez for sharing a room with me, as well as Maria Kurchuk, Colin Weltin-Wu, Jayanth Kuppambatti, Baradwaj Vigraham, Weiwei Jiang for their support and input during tapeout.

Finally, a special thanks to Mania. I got to know her during the most critical part of my thesis, submitting my major papers, taping-out and testing my chip and I know how hard it is being around me during such phases. Her support is also a piece of this thesis.

Concluding, a special thanks to everyone at Columbia. It has been a wonderful 7-year experience.

Chapter 1

Introduction

This thesis presents alternatives to classical digital signal processing (DSP) systems.[1] In particular, the thesis investigates design methods and signal processor structures which are complementary to conventional synchronous DSPs. All contributions of this thesis have the common property of being *adaptive*: they internally vary their structure and operation according to their actual input signals to offer significant advantages. For many decades, DSP systems [1], [2], [3] have been fully synchronous. The synchronous operation of classical systems imposes some limitations to their performance, which the alternative DSPs of this thesis attempt to address.

This chapter serves as both an overview of the classical DSP system, as well as a sketch of the contributions of this thesis. First Sections 1.1 and 1.2 present an overview of the structure and key limitations of classical DSP systems respectively. Following this, Section 1.3 describes the main focus of this thesis, which is the enhancement of DSPs with adaptive hardware. Sections 1.4 and 1.5 present a detailed overview of the different approaches on adaptive DSPs, divided into two

[1] The term "DSP" will be used to denote both a "digital signal processor" and the term "digital signal processing" for the remaining of this thesis.

parts: adaptive hardware for continuous-time (CT) DSPs (Section 1.4), and adaptive paradigms for other types of DSPs (Section 1.5), such as level-crossing and companding. Section 1.6 concretely states the various contributions of this thesis, and Section 1.7 presents the structure of the thesis.

This thesis' work involves two major axes: use of adaptive design methodologies, with focus on continuous-time (CT) DSPs [4], [5], as well as asynchronous design [6], [7]. CT DSPs and asynchronous systems are a natural fit for one another. From the viewpoint of DSP design, asynchronous design is the tool that enables the design of a CT DSP. From the viewpoint of asynchronous systems, CT DSPs are a new application domain, which combines asynchrony with the requirement for real-time designs, i.e. designs which precisely preserve the time distance between consecutive events.

1.1 Overview of classical DSP systems

This section briefly reviews classical DSP systems. First the structure and operation of a typical DSP system are presented, followed by a sketch of the different encoding styles that the system can use.

1.1.1 Structure and operation

Fig. 1.1 shows a typical classical DSP system. All three components, the analog-to-digital converter (ADC), DSP core and digital-to-analog converter (DAC), only operate at discrete and uniformly-spaced time points set by the global clock. Such a uniform-in-time, or "discrete-time" (DT), operation is a defining characteristic of classical DSP systems.

The system's operation is fairly simple. The ADC assigns an N-bit digital code to the analog

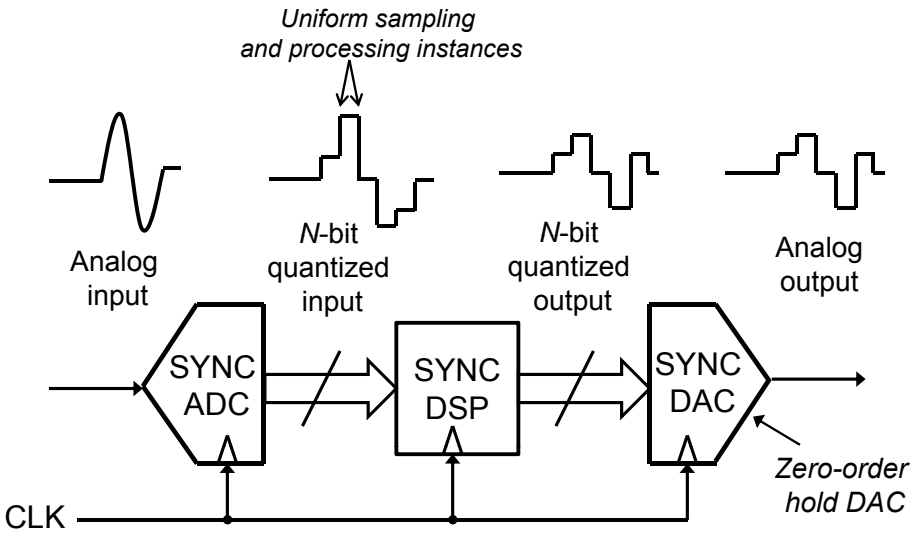

Figure 1.1: Top-level view of a classical ADC/DSP/DAC chain, i.e. a classical DSP system.

input samples obtained at uniformly-spaced time points. The sample rate must be greater than at least twice the maximum frequency of the analog input, as imposed by the famous Nyquist criterion [8]. The DSP core processes the digital input for a digital output in DT points. The digital-to-analog converter (DAC) converts the digital output to an analog quantity; in the typical case of a zero-order-hold DAC, the output is simply a piece-wise constant analog waveform.

Secondary components of a DSP system, not shown in Fig. 1.1, are analog filters before the analog input and after the analog output. The analog input filter ensures that the input to the ADC is band-limited, and the output filter smooths out the piece-wise constant analog output. These two filters are not part of the discussion in this work.

Classical DSPs are widely used because of their simplicity. They are easily programmable for different frequency responses, and their overall programmability by far exceeds the one provided from analog filters. Furthermore, the power consumption of DSP systems is smaller compared to that of analog filters, since the former do not require static power consumption for biasing, as

analog filters do.

1.1.2 Signal encodings and formats

The "encoding" or "format" refers to the way the ADC maps the analog signal to a digital waveform. Such encoding can affect the characteristics of the entire DSP system.

Various ways have been proposed to encode an analog signal into a digital waveform. These approaches are also called "formats" or "encodings," since an analog signal is encoded into a digital bit stream. They are also called "modulations", since the value of the analog signal is used to modulate a characteristic, i.e. amplitude, period, duty cycle, etc., of a digital waveform. The synchronous versions of the most widely-used formats include:

- *Pulse-code-modulation* (PCM) [9], which is similar to the sample digital waveforms shown in Fig. 1.1. The binary code, representing the amplitude, at each period of a pulse train with fixed duty-cycle is modulated according to the sampled signal's value.

- *Pulse-width-modulation* (PWM) [10], where the duty-cycle of a fixed-amplitude pulse train is modulated by the sampled values.

- *Sigma-Delta* ($\Sigma\Delta$) modulation [11], where the average density of 1s and 0s of the pulse train is modulated by the sampled values.

Asynchronous versions are also available for these modulations, in which cases the provided modulated digital signal is not clocked. Information on asynchronous digital modulations can be found in [12], [13], [14].

Different formats lead to different signal-to-error ratios (SERs) by using different bit widths and sample rates. Typically, PCM uses low-to-medium sample rates and many bits, PWM uses

medium sample rates but only 1 bit, and $\Sigma\Delta$ uses high sample rates and a few bits. The classification of sample rates as "low", "medium" and "high" are all with respect to the Nyquist rate, i.e. the minimum sampling rate for a given signal bandwidth. The ratio of the sample rate to the Nyquist rate is also called "over-sampling". See [15], and the references therein, for an overview of all ADC methods.

1.2 Limitations of classical DSPs

Apart from their simplicity, ease of design and benefits, classical DSP systems also come with certain limitations to their performance, which make classical DSPs less efficient solutions in certain applications. It is these limitations that the work in this thesis aims to alleviate through CT operation, asynchronous design methods and alternative DSP structures. An overview of four key limitations is given in this section; understanding the origin of each type of limitation is important before trying to address it.

All four limitations that are considered here involve the sub-optimal way in which classical DSPs process particular classes of signals. For such signals, there is significant room for optimization beyond the operation of a classical DSP system. These cases include many signals found in nature, like speech signals or those encountered in biomedical electronics. The latter have begun to attract an increasing amount of attention, especially ones involved in implantable devices, which call for circuits with minimal energy drain, given the inability to easily re-charge these devices. These classes of signals share the common property of *being silent* for the majority of the time, with the quiescent periods interrupted by *short, burst-type parts* with small or medium amplitude relative to the system's full-scale. Exploiting the actual signal characteristics toward the design of

application-oriented and efficient DSPs will lead to devices with increased battery life, which is an important ongoing research target.

1.2.1 Effect of reduced signal amplitude

The signal quality of classical DSP systems is affected by the amplitude of their analog input. This effect is due to the uniformly-spaced, i.e. equidistant, quantization levels used in classical DSP systems, which leads to a fixed quantization error affecting all samples regardless of the signal's amplitude. The signal-to-error ratio (SER) is therefore reduced for small-signal amplitudes.

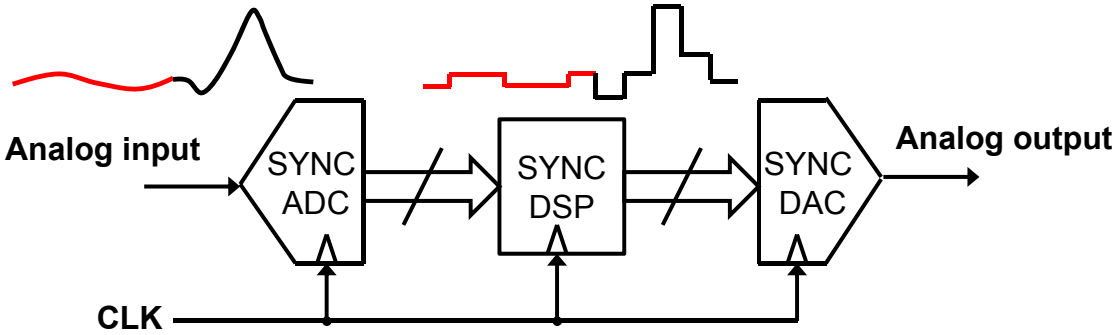

Figure 1.2: Effect of small input amplitude to a classical DSP system.

The net effect is shown in Fig. 1.2. In the presence of large inputs, the resulting digital signals (shown in the right half of each waveform in Fig. 1.2) span most of the system's available bits. In contrast, when the input is small (as in the left half of the waveforms in Fig. 1.2) , only a few of the system's LSBs [16] are exercised. Since the magnitude of the quantization error is fixed, on average, the SER reduces linearly with the input amplitude. A classical DSP system therefore needs large signals to offer fine resolution for these small-amplitude signals [3], [2].

There are two main reasons for this limitation. First, due to the finite-resolution uniform sam-

pling employed by classical DSP systems, *all samples include quantization error*. Sampling uniformly in time reduces the efficiency of the sampling process, since sampling points are not chosen efficiently with respect to the signal activity. Furthermore, a classical DSP system *does not adjust its dynamic range*, i.e. the range of its quantization levels, to the analog input's amplitude. The magnitude of the resulting quantization error is not adjusted to the signal's strength.

1.2.2 Power waste during quiet inputs

Classical DSP systems also have limited ability to adjust their activity to their input. By being triggered by the clock at regular times, the input is always sampled and processed at the same rate, regardless of whether it changes or not. Slower inputs will be sampled at the same rates as fast ones, despite the lack of need for such a high rate. Sampling and processing with no adjustment for the input activity results in more operations than absolutely required, hence less efficient processing. At the limit when the input is quiet, either zero or at a constant value, in principle no processing is required.

A number of classical DSP systems use some ad-hoc power management techniques to reduce their power dissipation. In particular, using heuristic methods classical systems make binary decisions as to whether they must process or not during each cycle and apply "clock gating" [17], [18], [19] to eliminate processing and dynamic power. A simple example consists of clock-gating part of the DSP system when multiple consecutive input samples are identical. These methods are not always successful [20], [21] at reducing the power consumption of the DSP system to the bare minimum. At extreme cases where the input alternates between short quiet periods and fast bursts, clock gating has little success.

Finally, as all synchronous systems do, classical DSP systems also require a clock distribution network. For large clock frequencies, the distribution network takes up a significant amount of power, which can not be fully eliminated through clock gating. This network effectively also adds to the leakage power, increasing the minimum power consumption of a classical DSP.

1.2.3 Dependence of the frequency response on the clock

Changing the clock frequency largely affects the behavior of any classic DSP system [1], [22]. The latter has no internal notion of time, and therefore of frequency; it blindly processes samples and relies on the clock to map these samples to time points. The end result is that the clock simply scales the entire response of the system. Fig. 1.3 shows an example: the response of a 16-tap synchronous finite-impulse-response (FIR) filter operating at 50 and 100 kHz. The two responses are scaled copies of each other.

In a practical case, the above implies that in any DSP system, programmed for a frequency response, the clock rate is **not** allowed to change. If the clock rate does change, then the entire DSP system needs to be re-programmed, to compensate for the clock change and restore the response. This process must happen off-line, meaning that the entire system must stop to be **re-programmed**. In many applications this requirement comes with a large cost or may not even be allowed. Hence, classical DSPs are not inherently fit for multi-rate applications, at least not without allowing for re-programming in order to track the clock rate.

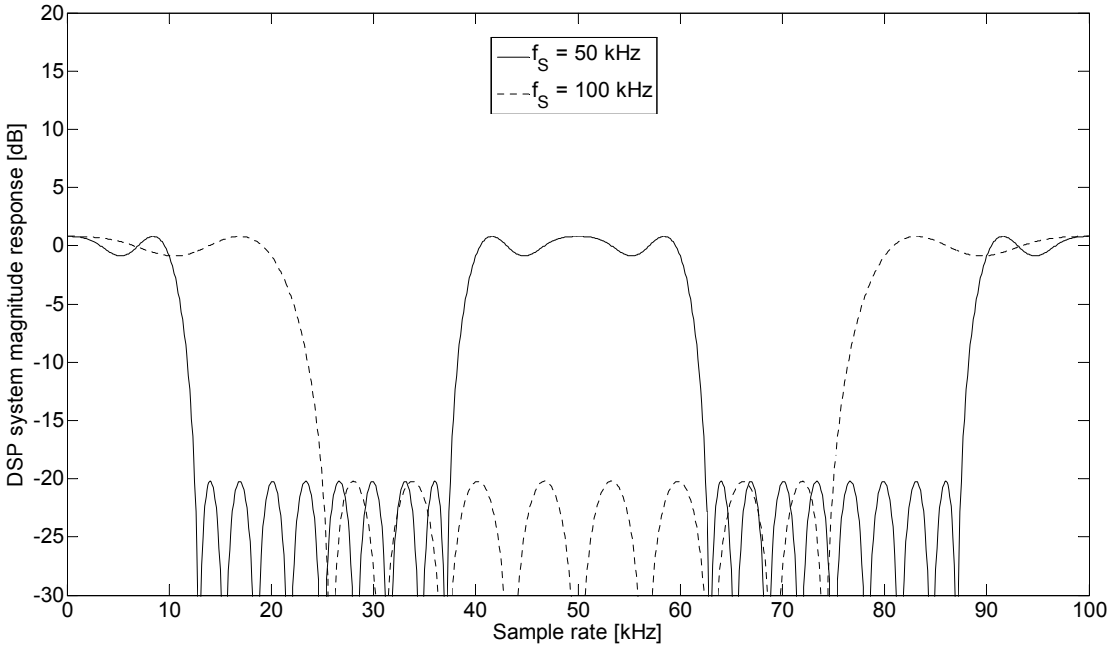

Figure 1.3: Dependence of a classic DSP system's frequency response to the clock rate, demonstrated through the response of a 16-tap synchronous digital FIR filter at two clock rates.

1.2.4 Limitation in the DSP resolution

Finally, the fidelity of DSP system is highly affected by the use of a clock. As previously explained, the use of a clock effectively introduces quantization error to nearly all samples [23], since signals are sampled inefficiently with respect to the quantization levels.

Fig. 1.4 shows the case of uniform sampling for a synchronous PCM ADC. At each clock tick, the analog input is compared to $2^B - 1$ decision levels, and based on that one of 2^B discrete values is assigned to the digitized version of the input signal.

As shown in Fig. 1.4, error is introduced in most, if not all, samples. The times at which most samples are taken is unrelated to the points where the analog signal crosses the decision levels. Such times would make ideal sampling points, since zero error would be introduced when sampling at those times. The effective error introduced in a classical DSP system is considered

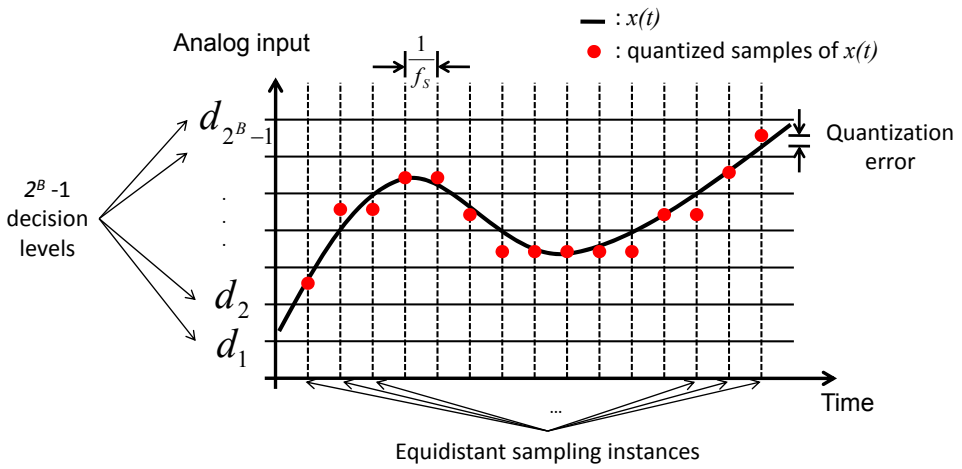

Figure 1.4: Uniform sampling and introduced quantization error in a PCM ADC.

random (called "noise" by many), with an average absolute value of half the quantization level distance.

To increase signal fidelity, classical DSPs employ two techniques. They either try to minimize the quantization error itself, or filter it digitally. The first technique involves using more bits, i.e. more quantization levels, to represent the signal. In this case, the resulting error is made proportionally smaller. Such a strategy is typically employed in Nyquist-type, e.g. PCM.

The second technique involves filtering of the quantization error in the digital domain. In over-sampled systems, like one using $\Sigma\Delta$ modulation, the quantization error is digitally processed using feedback. In such systems, the sample rate is much higher than the signal band. The quantization error is suppressed inside the signal band, increasing the SER. Such systems no longer require fine amplitude resolution (i.e. large number of bits), but do call for large oversampling ratio.

The relation between the SER, the number of bits and oversampling depends on the signal encoding. The two most widely known formulas are

$$SER_{PCM} = 6.02 \cdot B + 1.76 + 10 \cdot log_{10}(O) \text{ dB} \tag{1.1}$$

, for a B bit PCM DSP with over-sampling ratio O and

$$SER_{\Sigma\Delta} = 6.02 \cdot B - 3.41 + 30 \cdot log_{10}(O) \text{ dB} \tag{1.2}$$

, for a 1-bit first-order $\Sigma\Delta$ system with over-sampling ratio O. These formulas hold for maximum-amplitude inputs; amplitude reduction brings down the SER by 20 dB for every reduction in input amplitude by a factor of 10.

1.3 Research focus

Having reviewed the synchronous paradigm for a classical DSP system, this section now presents the main focus of this thesis: techniques to combine DSPs with adaptive hardware. This section sketches the main concepts of this thesis' research and the associated challenges, before going into greater detail in Sections 1.4 and 1.5. All of the different approaches that are presented in the thesis revolve around the context of *adaptivity*, where the functionality or structure of the processor tailors itself to some characteristic of the input. As a result, the processor's operation is made signal-dependent, offering various benefits, ranging from power gains to higher signal quality.

As a result, this thesis deals with DSP systems that differ from classical synchronous ones. For many decades, classical DSP systems have had a rigid and monolithic structure. Such systems are designed and configured for the worst-case input, e.g. maximum signal amplitude or frequency, and treat all their inputs in the exact same way as the worst-case one. As indicated previously, this behavior leads to certain signals being processed in a sub-optimal fashion.

In particular this thesis involves two main axes. These are: *asynchronous design methodologies* for CT DSPs, and *adaptive designs* for two new classes of DSP systems, CT DSPs and companding. The meeting point of all the different works is the adaptive behavior of the entire DSP system or parts of it, but the way in which each system or DSP adapts is different in each case.

At an abstract view, any adaptive DSP can be broken down to two parts, the "datapath" and "control" [24]. The "datapath" resembles a classical DSP system, with the key difference that it can – either as a whole or parts of it – operate in two or more distinct modes, which usually involve a tradeoff between power consumption and signal quality. The datapath can be considered as being blind and having no ability to sense its environment. The "control" part is the core of the adaptive system: it senses the input and performs the decision as to which mode the datapath will operate on.

1.3.1 Research challenges

The migration from classical, i.e. non-adaptive, to adaptive DSPs comes with three major design challenges. These are the requirements for *safety*, *light-weight control* and *scalability*.

"Safety" simply translates to the requirement of not disturbing the system during transition between operation modes. From an outsider perspective, such transitions must not be noticed at all. The adaptive system needs to internally reconfigure when needed and allowed, in a manner that does not affect its output. This requirement is often not straightforward.

Second, the added control to make the DSP adaptive should be light-weight. The overall adaptive DSP should offer clear benefits compared to the classical one, in order to justify the usage of adaptivity. Typical requirements involve power consumption, i.e. static+dynamic, and chip area.

Finally, any adaptive approach for a DSP should be easy to scale, including re-designing the system in different IC technologies or for different system parameters, including datapath bit-width, sample rate, etc.. Failure to do so means that such systems are only fit for one particular application, and do not offer similar advantages in other cases. Classical DSPs have been the norm for decades due to their simplicity and scalability, and any scheme proposed as alternative or complementary to the classical one should also have a scalable structure.

The following sections, namely 1.4 and 1.5, give an overview of the proposed approaches to adaptive hardware for DSPs, followed by our contributions in Sec. 1.6 and the thesis' structure in Sec. 1.7.

1.4 Asynchronous design methodologies for continuous-time DSPs

This section briefly sketches the theory behind continuous-time (CT) DSPs, as well as the different works of this thesis on this domain. First a small background is given on CT DSP theory and previous implementations, and then follows the description of the two projects related to CT DSPs: a new CT DSP chip prototype, and a method for optimizing the CT DSPs power through enhancement of the CT DSP's "delay line".

Continuous-time (CT) DSPs are a recently-proposed class of signal processors [25], [26], alternative to the classical scheme. The theory behind them was introduced at Columbia University [25], and over the past decade there have been a number of proposed approaches, such as [27], [28], and [29]. The work in [29] is synchronous, which uses a high-rate clock to emulate CT operation. In parallel, a number of different approaches have been presented for CT data acquisition, i.e. CT ADCs, as motivated by CT DSPs, such as [30], [31], [32]. Such CT DSP

systems have similar structure to classical DSPs, but now operate without using a clock, i.e. in a an event-driven fashion. A CT DSP system contains asynchronous ADCs and DACs, as well as a real-time DSP core.

By processing without synchronizing to a master clock, CT DSP systems obtain various benefits. Such systems have increased signal-to-error-ratio (SER) [33] by eliminating frequency-domain aliasing, and signal-dependent power consumption [33], [34], [35]. On the downside, a few CT DSP prototypes have appeared, given their requirement for complex mixed-signal designs, as well as elaborate analog and digital tuning. The recent CT DSP chip prototypes were non-scalable designs oriented around particular applications. In contrast, this thesis presents significant contributions to facilitiate the design of general-purpose, scalable CT DSPs. This thesis includes various contribution to the class of CT DSPs, including design considerations for a general-purpose, scalable CT DSP. This work led to a new silicon chip prototype [36], presented in Chaps. 5 and 6.

A large part of this thesis is devoted to asynchronous design methodologies for power-optimized and scalable CT DSPs. CT DSPs and asynchronous design are a very good fit, given their common ground on signal-driven operation and the absence of a global clock. Asynchronous designs and methodologies investigated in this work can enable the formal design of CT DSPs; the latter combine the asynchronous with the *real-time* domain, where exact timing between samples is preserved by design. The real-time domain is parallel to the asynchronous one, focusing on digital hardware with a built-in notion of time. The similarities are illustrated in Fig. 1.5, where an asynchronous (a) and a real-time (b) systems are shown along with their input and output events. The latter are marked as pulses for ease of illustration. Both asynchronous and real-time digital systems operate in a clockless fashion, and events are not uniformly spaced in time. In asynchronous systems (Fig. 1.5a) the spacing between events typically carries no information and is not preserved from input

to output. The case is the exact opposite in real-time systems (Fig. 1.5b).

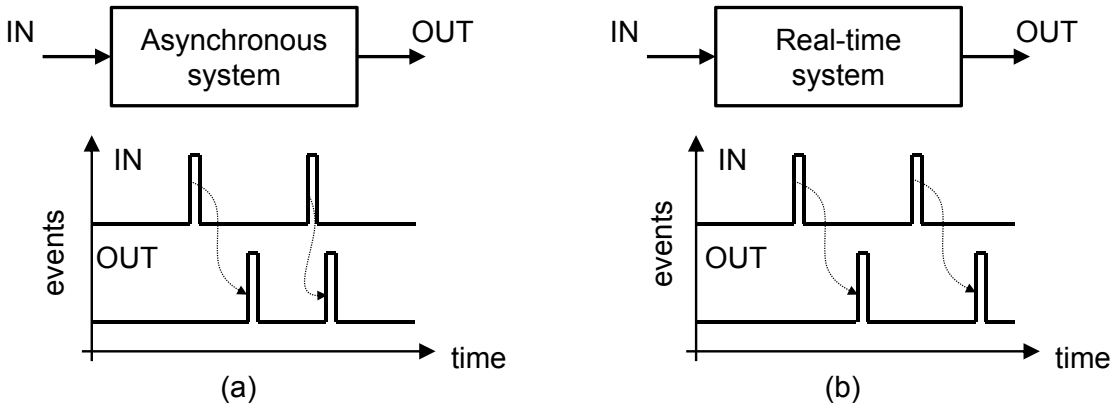

Figure 1.5: Difference between (a) asynchronous and (b) real-time digital hardware.

1.4.1 A CT DSP chip for multiple digital formats and sample rates

In all prior work, CT DSP implementations have been based on non-scalable approaches, oriented around particular applications. Each prototype was limited to using a single digital format for representing signals. Certain design parts were ad-hoc, not following any formal design styles or protocols.

In contrast, this work presents the design of a digital CT finite-impulse-response (FIR) chip with *much wider programmability and scalability*. This chip can be claimed to be the *first general-purpose CT DSP core*. Exploring the inherent property of CT DSPs to have an internal time notion, the implemented prototype was shown in silicon to maintain its frequency response intact while fed with inputs using a variety of different encodings and/or sample rates, synchronous or not, as abstractly illustrated in Fig. 1.6. To the best of our knowledge this is the first DSP, synchronous or CT, with this property, i.e. the preservation of frequency response for different sample rates.

The optimization of the processor's design from an architectural viewpoint leads to a highly

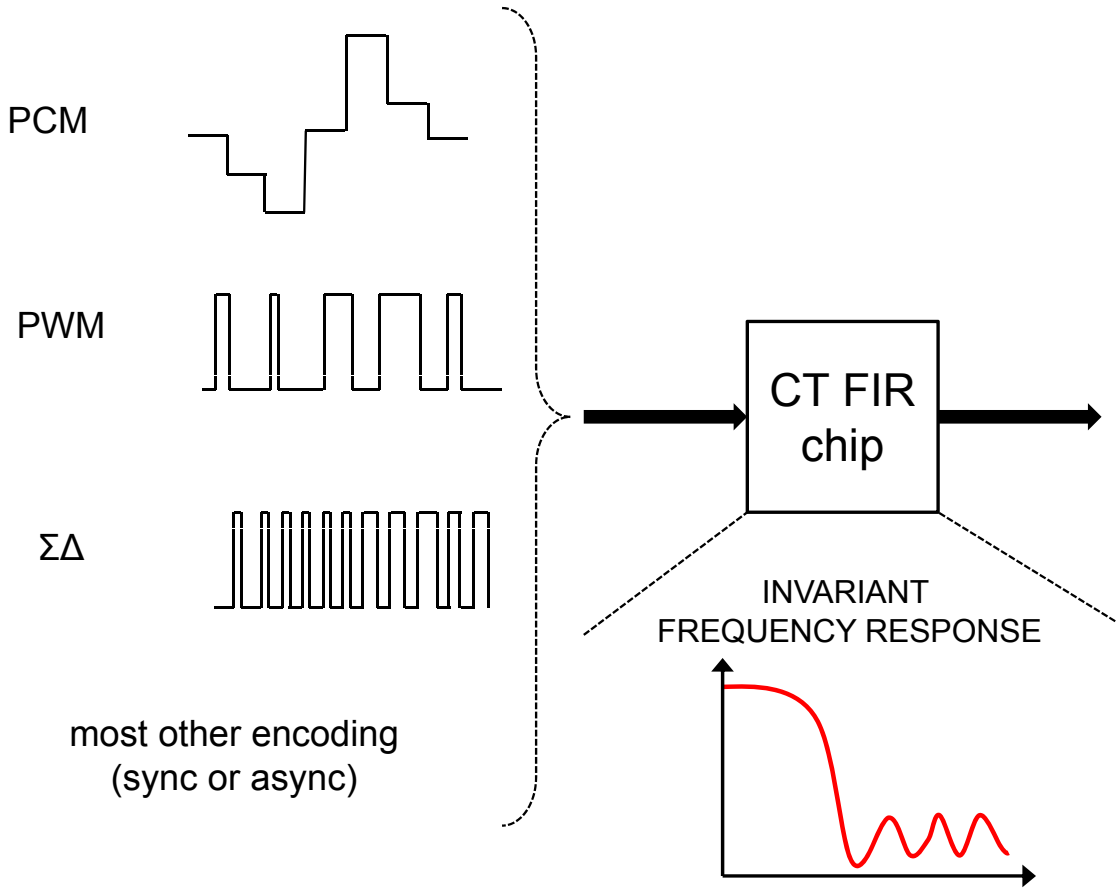

Figure 1.6: Conceptual view of the designed CT DSP, showing the ability to handle digital inputs of different rates and encoding formats.

scalable solution. This optimization leads to the decomposition of the CT DSP into modular components, using a mix of asynchronous and real-time domain design solutions. The result is a filter architecture with separate paths for timing management and data storage, and a pipelined arithmetic uniti i.e. multi-way adder. Both are easy to extend, i.e. re-design, for different specifications, such as maximum data rate, bit width or number of taps. Such a decomposition also enables a more energy-efficient approach for data movement and storage, compared to previous approaches.

Finally, the CT DSP chip prototype shown in this thesis is the *first to include a on-chip tuning*

mechanism required for programming the most important component of the processor, the real-time delay line. The two designed tuning blocks automatically program the delay of the line's parts (called "delay segments") for precise timing, as well as program the line's granularity. The automated tuning blocks combine design concepts from the real-time, asynchronous and synchronous domains.

1.4.2 Adaptive granularity management for delay lines

By using asynchronous design, it is shown how the power of a CT DSP delay line can be further optimized. The power contribution of such a delay line can be up to 70% of the CT DSP's power [33], so reducing the power consumption of such a component is critical. A delay line consists of many "delay cells", each providing a small delay, combined together into a pipeline structure. This work also has applications to other systems, beyond CT DSPs, which use delay lines.

In Chap. 7, a complete design methodology for dynamic management of the line's granularity, i.e. number of cells is introduced, [37], [38] based on incoming traffic. Given the largely varying traffic of a delay line, both in CT DSPs and in other applications, dynamic granularity management can be proven highly beneficial.

Such an adaptive delay line switches between different power settings in real time. It operates at its most conservative, and less energy-efficient, setting *only* when absolutely required, and usually operates in energy-optimized settings for the majority of time. Fig. 1.7 shows an intuitive example, where the granularity of the line is cut to half when the traffic is low. The overall latency of the line is kept invariant in all settings, but the energy consumption decreases proportionally to the granularity reduction.

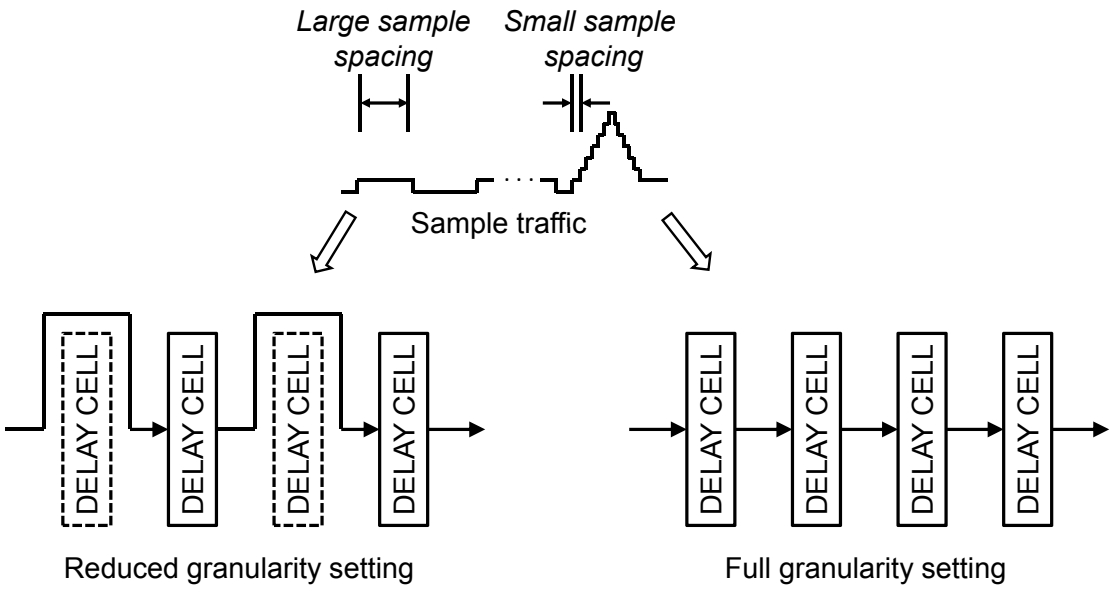

Figure 1.7: Conceptual view of the operation of a bi-modal adaptive-granularity system.

In particular, two new system components are introduced for dynamic granularity management. A traffic controller unit classifies incoming traffic in real time and chooses the appropriate granularity mode for the entire line. An asynchronous control line applies the chosen control to the entire line safely and without disturbing the traffic already inside the line. Both components are asynchronous, but properly interface with the real- time delay line so that their operation does not disturb the line's timing.

These components have a decomposed structure, offering a very scalable and modular solution. By handling a variety of design parameters, such as the number of settings and the granularity reduction in each setting, a large design space opens up toward the optimization of a delay line, given knowledge on its environment. Such an effort can also prove beneficial to CT DSPs, given the major power contribution of the delay line to them. The modularity of the solution allows simple re-designs for adaptive delay lines with two or more distinct granularities.

The reduced-granularity settings bring down the power of the delay line almost in proportion to

the grain reduction. As an example, a tri-modal system operating at either full, half or one quarter granularity consumes less power by a factor of two and four respectively in the two optimized settings.

Two concrete examples, for a bi-modal and tri-modal version, are implemented and simulated. Extensive simulations, at the level of schematic and post-layout extraction, confirm the theoretical expectations. While this methodology is presented in the context of CT DSPs, it is expected to attract attention in other domains as well, especially given the recent trend to shift applications from the amplitude-domain to the time-domain, due to the degrading amplitude resolution and increasing time resolution in modern IC technologies.

1.5 Adaptive DSPs: level-crossing and companding DSPs

The second major axis of this work is new DSP structures, alleviating the limitations of the classical scheme. Such adaptive schemes, either CT or DT, will change their operation dynamically in some way as demanded by the environment. As a result, the operation or performance of such systems adjusts to the characteristics of the input. This section sketches the operation of these DSPs.

Two adaptive DSP systems are explored: (i) a CT DSP processing level-crossing samples using accurate interpolation, and (ii) a DT companding processor of MPEG audio. The former offers high-resolution processing by using level-crossing samples, obtained through a CT ADC, and precise interpolation between the samples to reconstruct the signal with high accuracy. The latter increases resolution for small-amplitude signals, by modifying the internal structure of the DSP core accordingly.

1.5.1 A method for high-resolution processing of level-crossing encoded signals

This work delivers a new method for processing signals encoded via level-crossing-sampling (LCS) [39]. This method employs CT sampling but, due to the computations it employs for numerical interpolation, requires (at this early stage) off-line processing. Therefore, this method is a mix between CT and off-line DSP, i.e. a high-precision DSP usually performed on a large computer.

Instead of relying on amplitude resolution, *efficient interpolation* is utilized to achieve high resolution. This even holds when signals are coarsely sampled, i.e. by only using a small number of quantization levels. Furthermore, the scheme of Chap. 3 [40] involves fully signal-driven sampling and processing, by exploiting the inherent property of LCS to sample only when the signal is active, resulting in variable-rate data acquisition and DSP activity.

The method of Chap. 3 is not limited to LCS. It can also operate on any samples of an analog signal obtained with little or no quantization error. A complex arithmetic interpolation between samples can be designed to provide large SERs, comparable to the ones of over-sampled DSP systems. As opposed to the latter, however, the operation of this method remains signal-driven and is automatically disabled during input silence without requiring power-management scenarios. This method, therefore, combines high-resolution processing with signal-driven operation, at the cost of much increased numerical requirements.

Given these capabilities, the scheme of Chap. 3 is a promising future direction for DSPs, given its good match with new technologies for integrated circuits (ICs). The requirements for high fidelity are moved from the amplitude domain, which poses many design challenges given the reduction in supply voltage in new technologies, to the time and numerical domain. Given

the rising clock speeds of modern systems as well as the capability to integrate much arithmetic hardware, LCS processors relying on these resources have much room for improvement. Even though the numerical requirements of this method are very large, almost impossible to handle from a DSP chip, it is hoped that future work can relax such numerical requirements, opening the way toward a feasible version.

1.5.2 Companding DSPs for MPEG-encoded audio

The second class of DSPs explored perform discrete-time (DT) processing using the *companding* technique. This technique was introduced in non-dynamical systems for signal transmission [41]. Although, for dynamical systems, this technique has been previously introduced [42] and implemented in DSPs [43], a further optimization using adaptive techniques is provided in the context of MPEG audio [44].

Companding (*comp*ressing and exp*anding*) is an adaptive technique that adjusts the DSP system's dynamic range to the input signal [43]. As a result, a companding dynamic system keeps its SER large even for small inputs. Companding dynamical systems scale their inputs before sampling, so that all inputs are relatively affected by quantization error in the same way. They adjust their internal structure accordingly, to compensate for the time-varying input scaling, and restore their output back to its original order. MPEG-encoded audio is a perfect fit for this type of processors, given the normalized manner in which all MPEG information is stored.

The companding MPEG DSP of Chap. 8 process in the sub-band domain [44]. Signal information is broken down to sub-bands, and MPEG processing happens within each sub-band separately. Besides the compatibility of this structure to the MPEG standard, sub-band processing has the ad-

ditional advantage of limiting the noise of each sub-band processor within the sub-band itself. As will be shown, this translates to large noise being always close in frequency to large signals. Large noise of MPEG processors is almost inaudible, as shown by listening tests, allowing high-quality companding DSPs to be built using low-complexity hardware.

1.6 Contribution of this thesis

In summary, this thesis delivers several important contributions for adaptive DSP systems. Such structures have the property of modifying their operation, or even structure, dynamically and according to some property of their input. The thesis discusses existing approaches, such as CT and companding DSPs, and delivers design methodologies and enhancement to their operation, as well as new DSP approaches, such as a new class of level-crossing-sample processors. The different approaches of this thesis can be classified into two categories. One part of this thesis presents techniques to convert a non-adaptive DSP, or part of it, to a version which adapts to some property of the signal. The second category is design methodologies and enhancements for existing DSP systems which are already adaptive.

CT DSP chip: The largest portion of this thesis, namely Chaps. 4 to 6, are centered around the design and results from a silicon implementation of a new CT DSP chip. The chip is a flexible general-purpose CT DSP, which can process signals of many different sample rates and modulations. This contribution is a direct improvement compared to both previous CT DSP chips [33], [34], which were restricted to narrow operating environments. As any CT DSP chip, this chip too has signal-dependent power consumption, unlike synchronous DSPs, and absence of frequency-domain aliasing. For some cases, the signal-to-distortion ratio through this chip can

exceed that of synchronous DSPs.

The chip builds loosely on a previous voice-band CT DSP chip [33], but has a significant number of new contributions. It also corrects several design issues in [33]. The new chip has a scalable design methodology, based on decoupling between timing and data paths, to handle much larger data widths internally to the chip, namely 8 bits, while only 1 bit was used in [33]. This chip also has significantly increased programmability options for its frequency response, and can handle a larger variety of inputs compared to [33]. Furthermore, this chip makes uses of a systematic and formalized asynchronous design style, thereby correcting some design flaws in the previous implementation. It also has a new design approach for the DSP's multi-way adder, which is both more energy-efficient as well as easily scalable for different filter orders. Finally, this chip is the first CT DSP to include automatic, on-chip delay tuning. Compared to previous CT DSP chip prototypes, this one has a wider usage; we take the view that this chip is the first step toward building general-purpose CT DSPs.

Delay line granularity management: The second main contribution of this thesis, shown in Chap. 7, is a methodology to build delay lines with adaptive granularity. Such adaptive delay lines can be directly used in a CT DSP, among other applications. Our deliverable is a scalable methodology to build such pipelined delay lines, whose number of stages, i.e. delay cells, varies according to input traffic density; a systematic approach to design such lines for any number of granularity modes is shown. Two asynchronous controllers perform the dynamic granularity management; both come with minimal overhead with respect to the line itself, in terms of both area and power.

Even though dynamic management for pipeline depth is not new, our work clearly differentiates from previous work by treating the overall delay of the pipeline as an invariant, which is carefully

preserved as the line's operation varies dynamically. Also in contrast to previous approaches, this method does not come with restrictions for a small number of pipeline stages.

Adaptivity in other types of DSPs : Finally, this thesis includes two more examples of how adaptivity can be used in the context of DSPs in Chaps. 3 and 8. Chap. 3 shows a method for directly processing samples obtained through an adaptive sampling scheme, namely level-crossing sampling. This method can be used for processing of non-uniform samples without the need to re-sample to convert to synchronous ones. Contrary to previous work on asynchronous sampling, this method uses a precise interpolation and thereby results in high signal-to-error ratio outputs. Even though this work is at an early stage and currently requires intense numerical computations, it demonstrates that adaptive sampling has the inherent capability for high-quality processing.

In Chap. 8, a companding DSP, which is inherently adaptive, is enhanced with heuristic techniques in order to further reduce the required complexity. In particular, Chap. 8 shows how to eliminate much of the overhead that a companding DSP requires for the calculation of the required signal envelopes. Such envelopes are key control signals, which perform the main functionality of the companding technique, i.e. adapt the DSP's dynamic range to the input signal's magnitude. Our contribution, i.e. the heuristic methods for envelope extraction were used in a software implementation of an MPEG DSP, result in a DSP whose quality resembles the one from a floating-point version, even though using a coarse fixed-point companding implementation.

1.7 Structure of the thesis

This thesis is organized as follows. Before going in-depth into the core thesis work, first a thorough background is given in Chap. 2 on the two main areas of this thesis' contribution: CT DSPs and

asynchronous design. The former are the main category of DSPs used in this work, and the latter is the framework for formally designing asynchronous (i.e. signal-driven) digital hardware. The combination of the two is required for the largest part of the core work.

Chap. 3 presents a theoretical study for a high-resolution DSP using samples obtained via level-crossing sampling. This DSP has the potential of leading to CT DSPs which can combine high resolution with signal-driven operation.

Chaps. 4, 5 and 6 focus on a general-purpose CT DSP chip design, demonstrating interesting properties of this class of processors such as the decoupling of the frequency response from the data rate.

Chap. 7 provides a complete design methodology for a delay line, such as the one used in CT DSPs, with adaptive granularity, exploiting sparse traffic toward energy savings.

Chap. 8 shows a design paradigm for an adaptive synchronous DSP, this time using the companding technique to increase signal fidelity for small-amplitude inputs.

Chap. 9 provides suggestions for future research. Finally, appendices A and B offer supplementary material used for understanding some parts of the thesis. Appendix A gives a thorough analysis on jitter in CT DSP delay lines and does a fair comparison between two different delay line approaches (serial and parallel). Appendix B presents burst-mode (BM) specifications for asynchronous controllers used in the adaptive delay line of Chap. 7.

Chapter 2

Background

This chapter serves as background for the two major axes of this thesis' work. Continuous-time (CT) DSPs constitute the main class of DSPs examined as an alternative to the synchronous paradigm. Asynchronous design is mainly used in all this thesis' work on CT DSPs, including the design of the chip prototype and the adaptive delay lines. Fundamentals of both concepts are important to be presented, since both asynchronous digital designs and CT DSPs will be used throughout the following chapters.

2.1 Continuous time (CT) DSPs

CT DSPs [25], [4] are an emerging class of DSPs, introduced as a complement to classical synchronous DSPs. The main difference from their synchronous counterparts is that processing is *not* performed at discrete times, fixed by a clock, but rather at any point on the continuous time axis, as imposed by input the signal itself.

2.1.1 Introduction: CT DSP theory

The theory behind CT DSPs was introduced by Yannis Tsividis in [25], and a number of chip prototypes have followed up over the past decade. More details can be found in [4] and [5]. CT DSP systems are digital signal processing systems (ADC/DSP/DAC) operating without a master clock. They offer signal-dependent power consumption and cleaner-spectrum outputs by entirely eliminating aliasing in the frequency domain. They do, however, require manual design currently and mixed-signal hardware, sensitive to timing, as well as analog and digital tuning.

At a top-level system view, a CT DSP is similar to a classical one, but operates on samples at irregular and signal-driven instances as illustrated in Fig. 2.1. The system has the same key building blocks as a synchronous DSP, i.e. ADC/DSP-core/DAC, however operating without the aid of a digital clock but rather in an event-driven fashion. Fig. 2.1 shows this difference at a top-level view, highlighting the similar structure of the two systems and the key absence of the clock as an extra input to the CT DSP system. As a result, the operation of the entire CT DSP system is determined by the actual activity of the analog input. At the extreme point when the latter is completely quiet, the dynamic power of the system is entirely eliminated and the total power reduces to the standby power, i.e. bias and leakage.

The replacement of the clock-driven operation with an event-driven one has implications on all system components. A block-level view showing important signals in a CT DSP system is shown in Fig. 2.2. This figure shows the case where a LCS ADC is used, resulting in the ADC output being equal to a quantized version of the analog input. A new sample is *only* created when the analog input crosses a pre-determined y-axis (i.e. amplitude) value, i.e. a threshold, resulting in non-uniformly-spaced sampling instances. At each of these sampling instances, the DSP core is

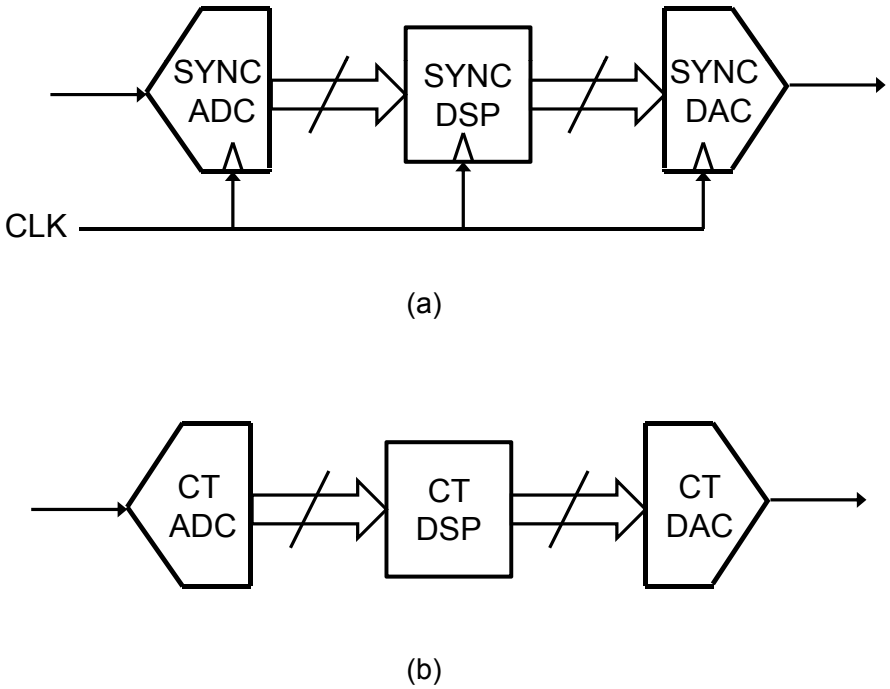

Figure 2.1: Top-level-view comparison between (a) a synchronous and (b)a CT DSP.

triggered and fed with a new sample. The core then processes the input samples and produces non-uniform output samples, which are then converted by the CT DAC to an analog output.

The resulting power consumption of a CT DSP system scales *almost linearly* with the average number of samples of the analog input. Processing is performed only when required, on demand, and each sample is handled independently of others. As the rate of change of the analog input decreases, so does the number of obtained input samples, linearly reducing the system's dynamic power. In the extreme case of a fully quiet input, the system power reduces to the standby power. Such a property of on-demand energy consumption, without the need for instrumentation of fine-grained clock gating circuitry, can highly benefit systems where energy consumption is critical. Such systems include certain bio-medical applications, and others with no access to rechargeable power sources. Exploiting the signal environment of such applications to prolong the battery life is natural in CT DSPs without requiring any power management schemes.

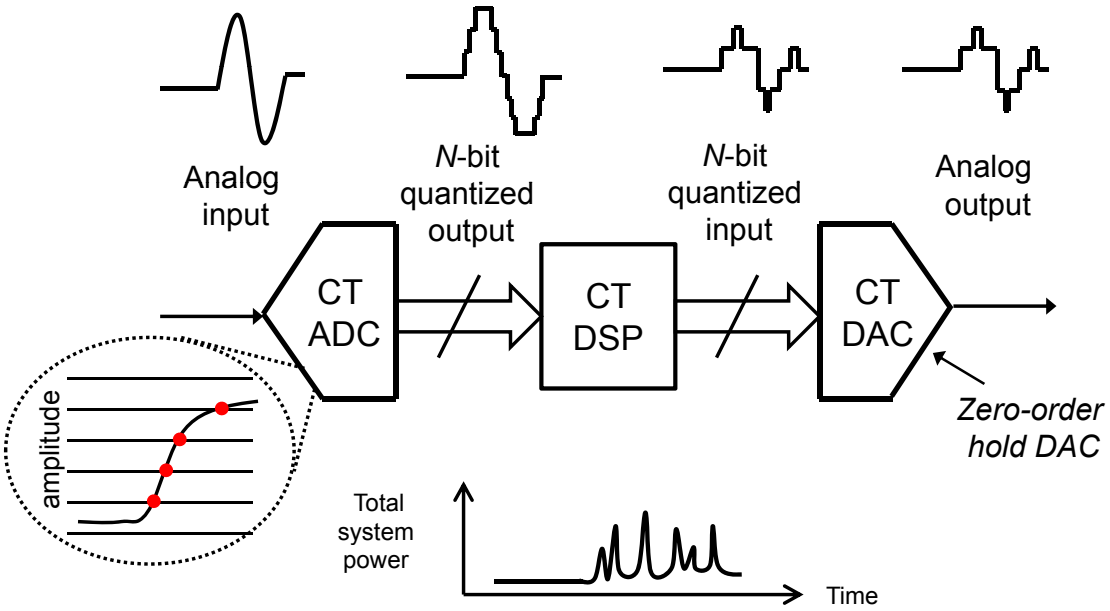

Figure 2.2: Top-level-view of a CT DSP system, showing important signals.

Further benefits are introduced by the absence of a clock as a system driver. Frequency-domain aliasing, caused by the presence of the clock during sampling, is *entirely eliminated* [4]. The result is that quantization components, introduced by the finite resolution of the CT ADC, do not become aliased back to the signal base-band to reduce the signal-to-distortion ratio (SDR), leading to significantly improved SDR compared to the case for synchronous DSPs [27]. Furthermore, any transients of the analog input are sampled and processed faster, since they do not have to first be synchronized to the master clock [45].

2.1.2 CT DSP implementations: previous work

A number of limited silicon prototypes [46], [33], [34] have been fabricated during the past decade, serving as proof of concept for CT DSPs. So far all prototypes have been truly application-oriented, showing the benefits of CT DSP in a variety of domains. In parallel, there have also been some reported designs using FPGAs and/or discrete components, emulating CT operation using fast

synchronous clocks [45], [47].

All CT DSP systems so far demonstrated the signal-dependent power consumption of the proposed architecture, as well as the absence of aliasing. In [46] and [33], absence of aliasing led to a significant improvement in signal fidelity (SDR), while in [34] it was exploited for good rejection of out-of-band interferers in a wireless receiver system.

All systems so far used a level-crossing-sampling (LCS) CT ADC and implemented an FIR filter as the CT DSP core. The former was used to provide truly-CT sampling and A/D conversion, thus eliminating both sampling and DSP core activity during periods of input silence. The finite-impulse-response (FIR) structure, through its limited duration of the impulse response (as the name implies), also enables a true system self-power-down during input silence: when input samples stop arriving, an FIR structure will "flush out" the remaining samples after some time and will no longer perform any processing. This behavior is not present in infinite-impulse-response (IIR) filters, which through internal feedback process each sample multiple times, in principle infinitely but in practice up to a limited time due to finite arithmetic resolution. FIR filters are also inherently stable and very suitable for many applications like audio processing due to their good phase properties, despite the fact that a larger filter order is needed (compared to IIR) to achieve similar amplitude rejection in the stop-band.

All prototypes used a *calibrated delay line* with multiple *delay segments* to generate the delayed input copies, as required for the FIR filter's operation. Each segment in [46] and [33] was constructed using a large number of *delay cells*, each of a very small delay, so that multiple samples can be processed concurrently in each segment. This requirement arises due to LCS, which generates samples with much smaller spacing than the time spacing between the FIR segments.

In parallel with CT DSPs, there is much ongoing work on CT ADCs and DACs. Apart from the

designs in [46], [33], [34], [29], [32], there has been a large number of different ADC approaches with signal-dependent operation [31], [48], [30], [49], [50], [51], [52], [53]. The combination of signal-dependent analog or mixed-signal hardware, as well as asynchronous logic, is present in multiple variations across these ADCs to achieve specifications targets like speed [30], resolution [50] or to even tackle some inherent problems of level-crossing sampling such as signal-dependent sampling latency and the large oversampling inherent to LCS [31] by skipping some samples without distorting the output SDR [54]. Finally, the work in [55] discusses an optimization of the level-crossing decision levels taking signal characteristics into consideration.

Furthermore, several variations to the original CT ADC/DSP/DAC system in [26] have been proposed at a conceptual level. These include different approaches for the FIR operation [56], [57], [28], or use of other ADC formats like asynchronous $\Sigma\Delta$ modulation [58]. Such works are yet to be demonstrated in silicon; however, they are indicative of the attention that CT DSPs have attracted.

2.1.3 Limitations of previous CT DSP prototypes

While the prototypes presented thus far have made significant impact and demonstrated interesting properties, they have some significant limitations. Being early prototypes, they did not address certain design goals which remain open until now. Several key properties of CT DSPs, like the decoupling of the frequency response from the data rate, have never been demonstrated in silicon. Furthermore, previous chip prototypes were not generalized and scalable designs which would help establish CT DSPs as a complete class of processors. Part of this thesis covers both of these open ends, demonstrating a CT DSP chip with wider programmability and able to handle many more signals, showing a CT DSP chip with a much broader usage.

In greater detail, all previous CT DSP systems were limited to a *single sampling scheme*, i.e. LCS, resulting in consecutive digital input samples to the DSP core being different by at most 1 LSB. In [46] and [33], these digital samples were encoded using digital Δ-modulation, where a 1-bit difference (positive or negative) was enough to represent consecutive samples. While having a number of benefits, this encoding scheme also resulted in a DSP core which could only be used for the particular digital scheme, and was unable to handle any other digital format. In [34], the resulting 3-bit digital format was encoded as a thermometer-type digital code, which involves exponential complexity with respect to the system bit resolution and is not practical for larger bit widths. As a result, previous CT DSP designs are not easily scalable for different specifications and applications. No complete CT-DSP system with non-trivial bit-width and capability to handle a variety of formats has yet been presented.

The difficulty in scaling the previous approaches comes from one significant bottleneck. In these prototypes, *data always travels down the FIR delay segments along with the timing information*: within each delay segment data was always passed from one delay cell to the other, even though it was only used at the beginning and end of each segment. In [46] and [33] data was only 1-bit wide, coming from Δ-modulation; delay cells only handled a single bit of data. However, this may not be the case when using a different digital encoding. In cases where larger data width are used, the scaling of delay cells for much larger data widths is very difficult (and will come with additional power penalty, besides the obvious one in area).

As a second-order but important set of limitations, previous chips lack an on-ship tuning approach for the required analog and digital tuning.

In one part of this thesis, we present a CT DSP core which can help establish CT DSPs as general-purpose building blocks, complementary to classical DSPs. We demonstrate a fully scal-

able design methodology, proven in silicon [36] to successfully handle more than one digital format, and showing the inherent property of CT DSPs of fully decoupling their frequency response from the input sample rate.

2.2 Asynchronous system design

A background in asynchronous circuit design is presented before the core sections of this thesis. Such a review is essential for understanding this thesis' work in depth; asynchronous design largely appears in the CT DSP chip prototype and the adaptive delay line designs. Asynchronous techniques are also recently finding usage in a variety of digital systems.

Asynchronous methods define the communication and design of digital systems that operate and interface without using a digital clock. Such systems operate at irregular intervals and on demand, reducing their activity and power consumption in a natural way by taking advantage of their environment. At an abstract view, the global clock is replaced by a very-fine-grain clock, different for each system component, controlled naturally by the data itself, with varying frequency and duty cycle. Asynchronous systems have been investigated since the 1950's [59], [6], [60], [7], and during the past two decades a large number of breakthroughs have been reported [61], [62], [63], [64], [65], [66]. Such breakthroughs are motivated by the recent trends in VLSI systems, such as multi-clock operation of large processors and the demand for portable devices with low-energy consumption for prolonged battery life.

By eliminating the synchronous operation, a number of interesting advantages open up. Apart from the above-mentioned reduction in dynamic power which tracks actual input activity, power savings also arise by eliminating the clock distribution network, which takes up significant power in

many modern systems, especially large multi-core processors. Different parts of an asynchronous system draw supply current at uncorrelated times, reducing the maximum transient power drawn by the system, as opposed to a synchronous case where all activity and supply current is synchronized. Power footprints and spectral emissions of asynchronous systems are often significantly smaller [63], [67], [62], and some asynchronous chips alleviate on-chip hot spots. Good tutorials on the usage and properties of asynchronous systems can be found in [68] and [69]. Industrial examples of such systems' applications include Sun Microsystems' micro-pipelines [70], many commercial products by Philips [67], Achronix Semiconductor's GHz FPGAs (world's fastest to date) [71], routers and SRAMs by Fulcrum Microsystems (now Intel's "Switch and Route" division) [60], asynchronous-style discrete-time DSPs by Oticon Inc. [72], asynchronous high-throughput FIR filters from IBM [61], and many more.

As stated in [72], in most designs (including those performed for this thesis) doing a simple shift to asynchronous-style design does not guarantee lower power consumption. What makes the biggest impact in the performance of such systems is using them to exploit the characteristics of their environment, such as the variable operation rate, the characteristics of processed data, etc. Asynchronous systems can offer the potential for large speed-ups in modern ICs, since data carry their own information on completion, indicating themselves when it is safe to stop processing (instead of having to wait for an entire clock period). Such data-dependent processing [73], [74] can boost up the average speed of these systems; in other cases asynchrony provides high throughput hardware using pipelining [75], [76], [77], [66], [78], [79]. Asynchrony was also proven to be an extremely useful tool in modern, multi-clock-domain ICs, for safe synchronization between domains, leading to "globally asynchronous locally synchronous" (GALS) systems [80], [81], [82], [83], [63], [84], [85].

2.2.1 Asynchronous communication protocols

Asynchronous systems replace communication using a global clock with local hand-shaking. Asynchronous protocols define such master-slave or sender- receiver communications and describe both the synchronization of the transaction as well as the movement and encoding of data.

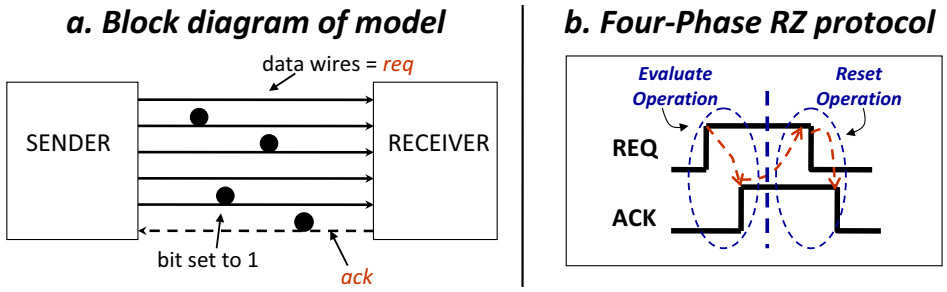

Figure 2.3: 4-phase asynchronous handshaking protocol.

There are two leading variations of hand-shaking protocols, called 4-phase and 2-phase. They describe the order of signal transitions for the sender's "request" (req) and the receiver's "acknowledgment" (ack) wires to complete a full transaction. Fig. 2.3 shows the case of the 4-phase hand-shaking, in which by default all transactions start and end with identical signal values of '0'. The transaction is divided to an "evaluation" phase and the "reset" phase. The 2-phase protocol omits the reset phase, so one transaction is described as a toggle on req followed by a toggle on ack. By experience 2-phase systems are faster due to omitting one of the two communication phases, at the cost of larger area required to implement them.

Usage in this thesis : The majority of the designs of this thesis use 4-phase protocol, such as in the entire CT DSP chip prototype in Chap. 5 and the majority of the asynchronous control and datapath for the adaptive delay line in Chap. 7. Only a small part of the latter is implemented in 2-phase.

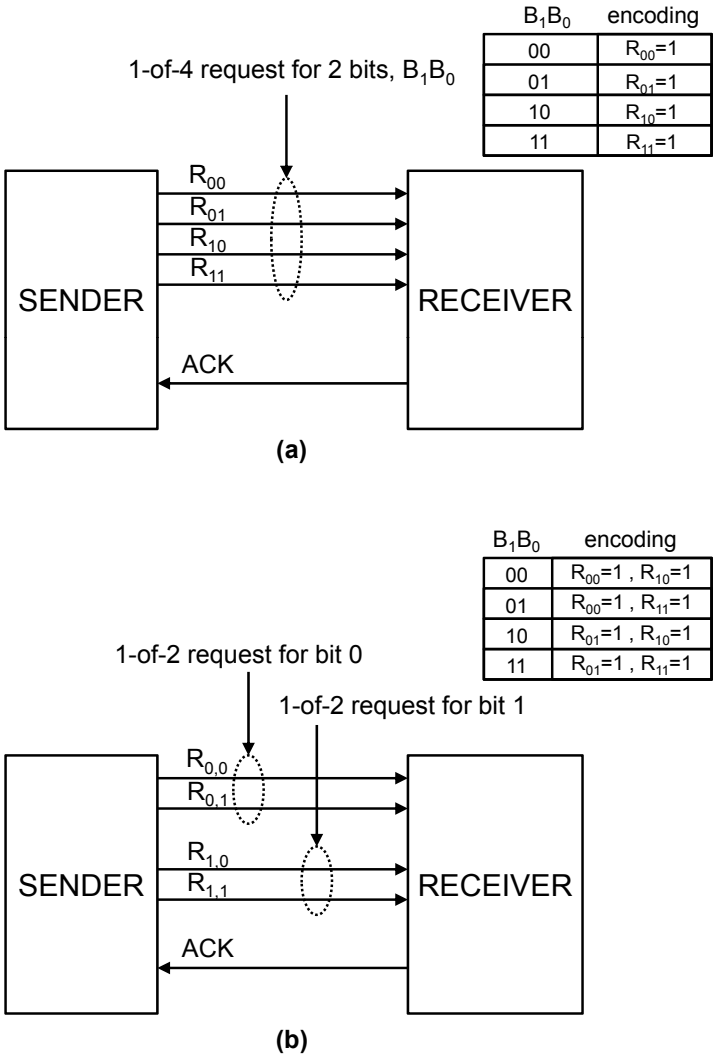

Figure 2.4: Delay-insensitive data encodings: (a) 1-of-4-hot and (b) dual-rail.

2.2.2 Asynchronous data encoding

Many different styles are used for data encoding, tailored to various design styles and involving several trade-offs [6], [69]. Data transmission is either part of the synchronization, leading to delay insensitive protocols, or completed before the synchronization to allow the latter to indicate data arrival (involving some sensitivity to timing). The "bundled" data encoding, part of the "delay sensitive" class, consists of data transmission in advance and then synchronization to indicate the arrival, after data stabilization at the receiver end is guaranteed. In this case, the "bundling

constraint" indicates the minimum time amount that has to separate the data arrival and the new request event, very similar to the setup condition in synchronous systems.

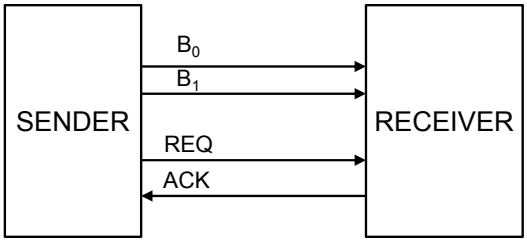

Figure 2.5: Bundled data encoding for 2-bit transmission.

"Delay-insensitive" encodings offer significant robustness to path delays and mismatches [6], [69]. A typical case is "one-hot" (or 1-of-N) encodings, where only one of $N = 2^M$ wires is active to indicate a transmission of the particular M-bit word. Extensions of this include m-of-n codes, e.g. 3-of-8. Fig. 2.4 shows a typical case for 1-of-4 and 2-of-4 encoding, where the sender uses 4 wires to transmit 2 digital bits. The difference is that in the 1-of-4 case (a) only one req-wire is active during each transmission, while in the 2-of-4 case (b) one req-wire is active per data bit. The extension to 3 bits would translate to 1-of-8-hot for case (a), and 6 wires (three versions of 1-of-2-hot, one for each bit) for case (b). Most delay insensitive encodings are 4-phase, but much ongoing work is also on 2-phase protocols, for example [86], [87]. Fig. 2.5 shows the same data communication scheme using the bundled data protocol.

Fig. 2.6 shows the signal dependencies (marked with dotted arrows) for the transmission of the 2-bit word $B_0B_1 = 01$, for the cases of (a) dual-rail (1-of-2-hot) and (b) bundled data encodings, using a 4-phase protocol. In the former case each bit, encoded in two rails, carries its own request by raising one of its two rails, depending on the transmitted value. Each bit's request can arrive at arbitrary times, and the receiver waits until all bits have been sent. Then the receiver acknowledges

by raising its *ACK* signal, which causes each bit to eventually de-assert its active rail low, again at arbitrary times. After all bits have been de-asserted, the receiver finally de-asserts its own *ACK* and completes the transaction.

In the bundled-data case, first all bits (each using a single wire) are set to their proper values, and *then* the sender asserts its request to indicate the valid sample. A safe margin, depicted by the "bundling constraint" limits how fast the sender can request once the bits have been set. The remaining protocol resembles the regular 4-phase one. Note that the data bits need not be de-asserted at the completion of the transaction. Finally, the 1-of-4-hot case, shown in Fig. 2.4(a) is similar to the case of dual-rail, but with only one active wire from the sender per 2-bit sample.

The 2-phase variation of the communication shown in Fig. 2.6 would simply involve only one toggle per transmission for each active wire. Hardware support for a 2-phase communication involving multi-bit data is typically more complex, due to the fact that different rails in each bit do not reset to the same value after each transaction. The receiver and sender both need to keep track of the past states, in order to successfully decode the sent data. This communication style can lead to higher throughput, due to the elimination of the reset phase in each transaction, but at the cost of increased complexity. However, various efficient implementations have been proposed [75], [83], [63], [86], [87], [88].

$Tradeoffs$: Different data encodings lead to different design styles, trading off complexity, energy consumption and speed. Bundled encodings usually come with hardware closely resembling the ones found in synchronous systems, allowing glitches on intermediate and output signals and including self-timed control paths to sample outputs at proper times. This design style is often fast and simple, but is simple and consumes little overall energy. However, the approach requires that one-sided (i.e. bundling) constraints are satisfied in the implementation. Such examples will

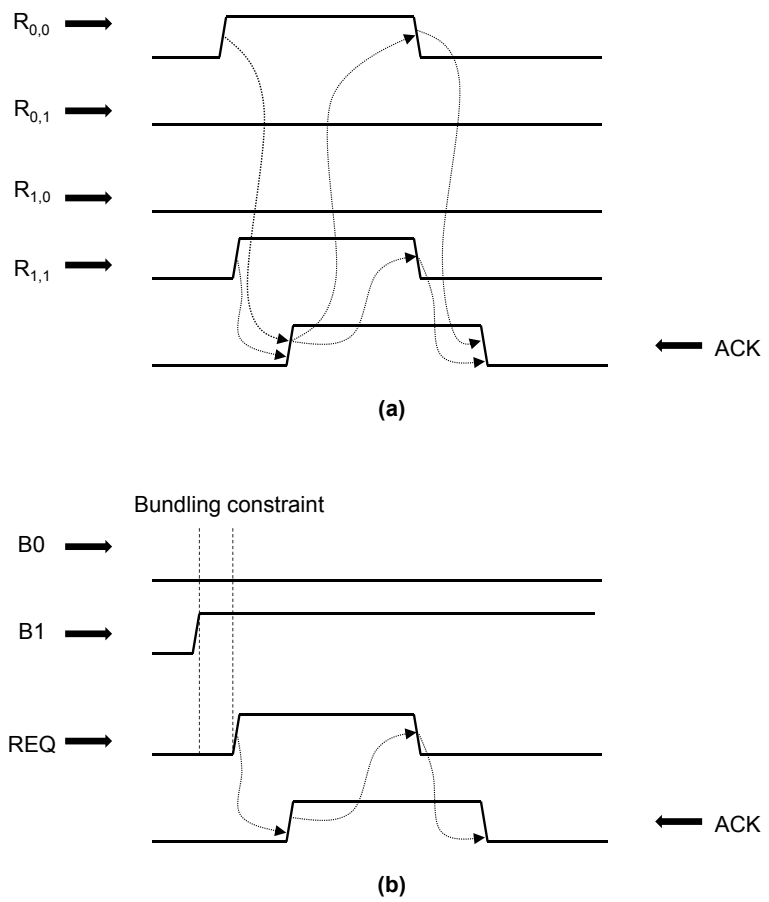

Figure 2.6: Transmission of the 2-bit word $B_0B_1 = 01$ in the cases for (a) 2-of-4 and (b) bundled data encoding, showing signal dependencies for a 4-phase protocol.

be presented throughout this thesis' work. Delay-insensitive encodings use more complex hardware, which does not allow glitches at any point, since glitches can be falsely interpreted as parts of transactions. This functionality comes with the added cost of increased energy consumption and complexity. However, these systems have great resilience to timing variability, and almost no timing constraints in their physical layout. See [89], [6], [69] for detailed tutorial.

Usage in the thesis : Both delay-insensitive and sensitive styles have been used in the work of this thesis. In particular, single-rail bundled encodings have been used throughout the CT DSP chip design in Chap. 5, and both delay-insensitive and single-rail bundled data encodings in the

adaptive delay line in Chap. 7.

2.2.3 Asynchronous controllers and burst-mode machines

Asynchronous controllers are an integral part of larger asynchronous systems, used for, among others, the previously-described operations such as synchronization and data transfer. They are finite-state machines with the proper asynchronous interfaces, either manually designed or automatically synthesized.

Burst-mode (BM) controllers are a subset of asynchronous machines [90], [91]. They are asynchronous Mealy machines, designed for communication using hazard-free interfaces. BM controllers receive "bursts" at their input channels and provide bursts at their output channels, where a burst is a set of pre-defined signal transitions which can happen in any order and with arbitrary relative timing. Fig. 2.7 shows an example of a BM machine with 2-inputs, 2 outputs and 3 state bits. An input burst can consist of any combination of A and/or B transitioning once, either positive or negative (symbolized $A+$ and $A-$ respectively). Each input burst can lead to any number or output and state bits transitioning once, in a hazard-free way, i.e. without glitches (monotonically).

To guarantee correct behavior of a BM machine, two conditions need to be imposed on the inputs and feedback states, similar to synchronous FSMs. The updated state feedback must arrive safely after the current input burst has been fully processed, i.e. new outputs and states have been created and stabilized. Second, the new input burst must arrive only $after$ the current burst has been completely processed. These constraints are one-sided and easy to satisfy by properly delay-padding the feedback and output paths of the machine. The conditions resemble the setup- and

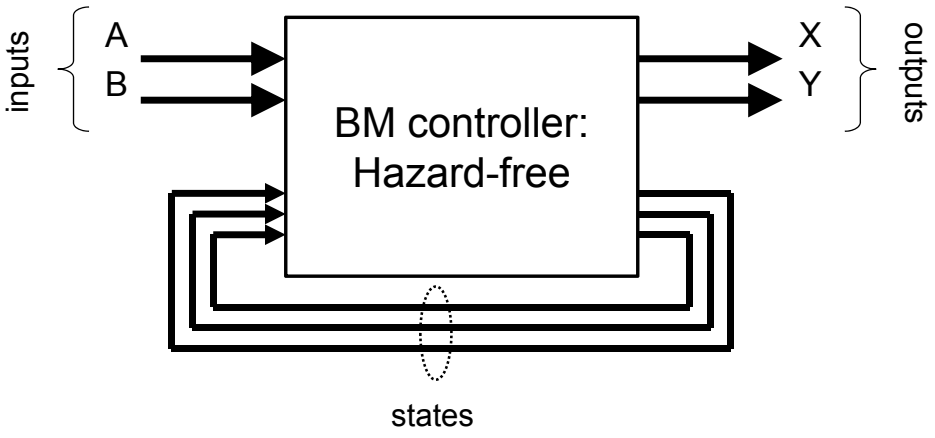

Figure 2.7: Example of a burst-mode asynchronous controller.

hold- time constraints of synchronous designs. More details on both the timing requirements, as well as the conditions for the BM bursts can be found in [90].

The MINIMALIST tool [90], [92], [93] is a design package for synthesis of BM controllers [94]. It provides hazard-free implementations of BM controllers using standard cells. It includes tools for exact and heuristic a 2-level-logic minimization, optical state assignment and others [95], [96]. The tool also provides Verilog-encoded versions of the controllers using initialization, as well as decomposition of high-fan-in gates for mapping to practical sets of standard cells with reasonable fan-ins. Given the standard-cell implementations, synthesized controllers can be easily translated to a layout through automated place-and-route tools. Various synthesis modes are provided, offering speed-area-latency trade-offs. MINIMALIST has been used in various academic and industrial designs, for example in the works in [97], [98], [99], [95], [100].

Usage in this thesis : The MINIMALIST tool is used in this thesis to synthesize all burst-mode asynchronous controllers for the adaptive delay line in Chap. 7. Examples of the specifications of the implemented controllers will be provided in Appendix B.

Chapter 3

A Method for Processing Level-Crossing-Encoded Signals

Level-crossing sampling (LCS) is an asynchronous sampling technique with a number of significant potential advantages. LCS results in signal-driven, high-precision operation, and is the main sampling method used in CT DSP systems.

In this chapter, we present a theoretical study on a DSP for processing such LCS samples with very high accuracy. Such a system, henceforth called "LCS DSP", is shown to potentially provide outputs of very high precision, even when they use very few bits. LCS DSPs use numerical *interpolation* to generate high-resolution outputs, with SER of 100 dB or more, from input LCS samples. LCS DSPs can still be classified as CT DSPs; however the amount of numerical computations they require, at this early stage, is prohibitive for an on-chip, real-time implementation, unlike the one shown in Chaps. 4 to 6. The method we describe in this chapter is more suited for off-line processing, e.g. in the case where the samples are first obtained in CT and are then sent to a computer for processing. We describe the mathematical procedure which can be followed in this

off-line processing.

LCS DSPs require very few input samples, but intense numerical computations. Furthermore, as we will show later in this chapter, they also require fine time resolution. These two resources, i.e. timing accuracy and numerical capacity, are becoming increasingly available in modern IC technologies. It is hoped that later versions of the method we demonstrate in this chapter, optimized to require significantly less computation, can lead to integrated approaches for LCS DSPs.

This chapter begins with an introduction on LCS, as well as a comparison of LCS to uniform sampling, in Sec. 3.1. The next section, i.e. Sec. 3.2, explains the motivation behind our investigating LCS DSPs. Sec. 3.3 gives an overview of existing techniques for numerical interpolation, which is main method that LCS DSPs use to achieve high resolution. Sec. 3.4 explains the operation of LCS DSPs and addresses implementation issues. Sec. 3.5 gives simulation results for a computer-based implementation of an LCS DSP, and Sec. 3.6 concludes this chapter.

3.1 Uniform sampling vs. level-crossing sampling

This section presents the properties of LCS sampling and emphasize its main differences from uniform sampling. Contrary to the uniform case, LCS has the potential for samples which do not include quantization error.

In classical A/D conversion, a signal $x(t)$ is sampled uniformly, and the samples are quantized to the nearest quantization level. This is shown in Fig. 3.1, where the quantization levels, shown as q_a, q_b, etc., belonging to a set Q, are assumed uniformly spaced.

Classical sampling comes with two main limitations. As illustrated in Fig. 3.1, the amplitude levels of the quantized signal do not represent, in general, the correct amplitude values of $x(t)$ at the

corresponding instants. Also, the required sampling rate is such that the fastest-changing portions of the signal are sampled frequently enough; because the sampling rate is constant, the slowly-varying signal portions are sampled at the same high rate as the fast ones, although in principle this is not necessary [101].

Consider now the situation for LCS [39], [102], shown in Fig. 3.2. The times at instants at which the levels q_a, q_b, ... are crossed are denoted by t_1, t_2, ... The signal can be represented by the pairs $\{t_k, x_k\}$, where the values $x(t_k) = x_k$ belong to the set Q of quantization levels.

LCS alleviates the two limitations of uniform sampling. Assuming that the time axis is continuous, i.e. that no error is involved in representing the time instants t_k, it is seen that the pairs $\{t_k, x_k\}$ fall exactly on $x(t)$. It is clear that the sampling times t_k are not set by a global clock, but rather depend on the form of $x(t)$ and the quantization levels. A/D conversion using this scheme has been discussed in the literature [103], [104], [31], [32].

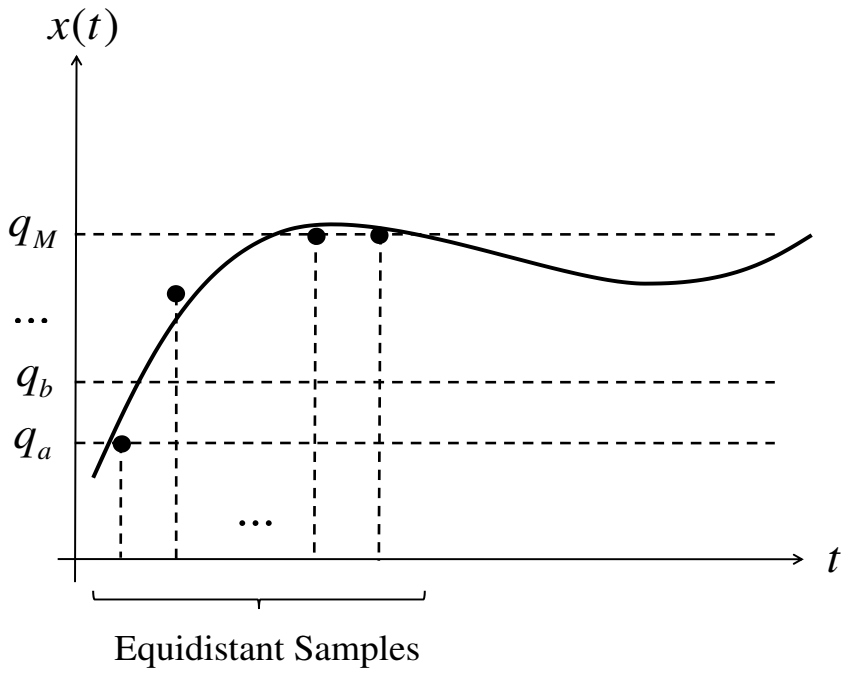

Figure 3.1: Uniform sampling and quantization.

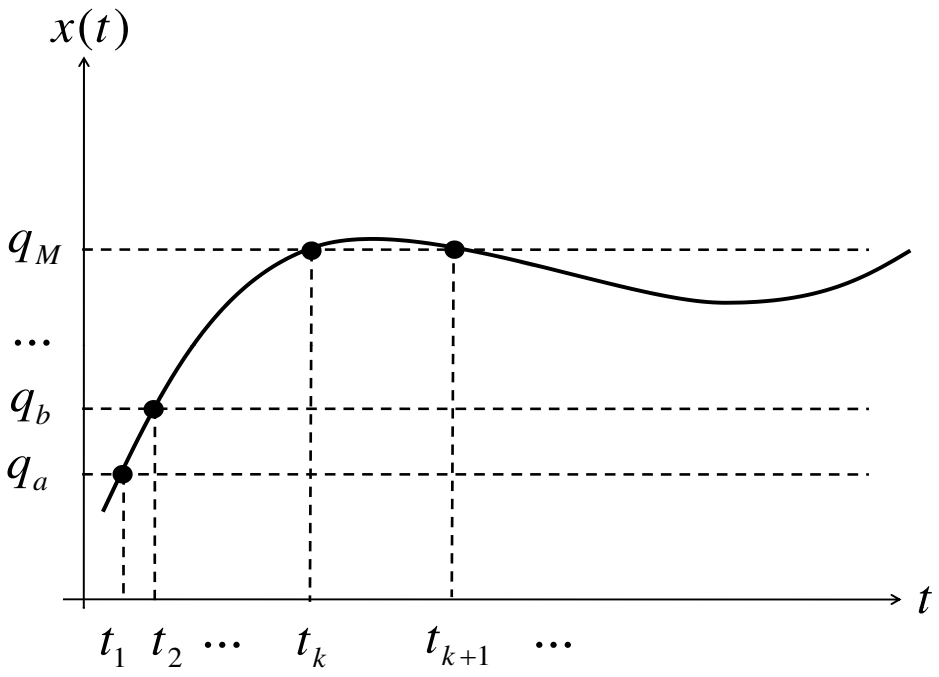

Figure 3.2: Level-crossing sampling.

LCS moves the need for sampling resolution from the amplitude to the time domain. It obviates the need for high amplitude resolution, since sampling is performed at points where the amplitude error is zero; at the same time, high time resolution is needed. Fortunately, techniques that emphasize time resolution benefit from recent advances in VLSI [105], whereas those that emphasize amplitude resolution encounter more and more problems. This makes the representation in Fig. 3.2 more and more attractive, in contrast to the classical technique of Fig. 3.1. For example, it is becoming easier and easier for remote sensors to generate signals of the form of Fig. 3.2, and for channels to transmit such signals.

Various techniques have been proposed for processing LCS samples. Recently-proposed approaches have been demonstrated for direct (i.e. real-time) processing, such as the ones found in [29], [33]. However, as we will show in Sec.3.3, these methods have limited accuracy, due to their simplistic approach for the problem of *interpolating* between the non-uniform samples. As

a result, such approaches still need sampling with many quantization levels in order to provide high-SER outputs.

In this chapter, we investigate ways to process such signals by using a more efficient interpolation technique. We show that even a coarse amplitude resolution, i.e. only sampling using a few quantization levels, is enough to provide a practically error-free output. Such a method is (at least for now) impossible for on-chip implementations, and much more suitable as a post-processing method, e.g. through a computer.

3.2 Motivation for use of LCS DSPs

Before getting into detail on the operation of LCS DSPs, this section first gives concrete view on their potential applications, as well as the motivation and open problems behind them.

The motivation for LCS DSPs are applications calling for the combination of high-resolution and low energy consumption. Such applications, e.g. for implantable devices, often involve sensing and processing burst-type signals, that consist of long quiescent periods interrupted by small bursts.

Classical DSP systems cannot easily combine high-accuracy and small power consumption during sampling. Nyquist-type systems can successfully power down to a certain extent using clock gating, but require amplitude sensing (i.e. ADC resolution) of high accuracy; this demand poses many design challenges in modern technologies (given the reduced supply voltages). On the other hand, over-sampled type systems (e.g. $\Sigma\Delta$) only require on time resolution for precision, but inherently require a large number of signal samples to provide accuracy; clock-gating in such systems can very rarely be successful. Classical systems come with the penalty of either the need

for fine amplitude resolution or lack of signal-dependent power consumption.

Given the characteristics of LCS mentioned previously, LCS DSPs are a good fit for such applications. By sampling only during periods of signal activity, i.e. during non-quiescent periods, and processing samples with inherently zero quantization error, LCS DSPs can potentially combine high-resolution processing with purely signal-driven power during sampling. LCS DSPs are a good match for cases when low-energy sampling is performed, and then digital samples can be transmitted to a digital computer for processing.

At this early stage, integrating the processing methods of this chapter (shown in Sec. 3.4) onto a single chip is impossible due to complexity. However, we hope that optimization of our methods, combined with further scaling of IC technologies, can enable a future on-chip implementation. As IC technologies continue to scale, the amount of arithmetic units that can be put on a single chip will keep increasing, to the point where the integration of hardware handling such complex interpolations is feasible. Such an outcome will make the performance of LCS DSPs comparable to that of synchronous ones.

3.3 Interpolation in LCS processing

Important details on "interpolation" techniques are now presented. Such techniques are key in LCS DSPs. Interpolation is used to estimate values of a signal between samples. Since the approach we present in this chapter is based on efficient interpolation, it is important to provide a good background on the topic.

Contrary to classical, i.e. uniformly-sampled, DSP systems, samples and signal paths in LCS systems are not synchronized to a master clock. Samples are signal-generated and not clock-

generated, so their spacing is significantly varying across time, as shown in Fig. 3.2. This fact gives rise to the need for inter-sample interpolation, explained using a simple example.

Assume the simplest, 2-tap, analog and continuous-time FIR filter prototype (similar arguments can be made for IIR):

$$y(t) = \alpha_0 \cdot x(t) + \alpha_1 \cdot x(t-\tau) \ , \tag{3.1}$$

where α_i are coefficients and τ a time-delay. The equivalent of 3.1 for a synchronous DSP is simply:

$$y_Q(n)(T_S) = \alpha_0 \cdot x_Q(nT_S) + \alpha_1 \cdot x_Q((n-1)T_S) \tag{3.2}$$

At each clock cycle (with clock period $T_S = \tau$), this synchronous FIR filter simply weighs and adds its latest two input samples $x_Q(nT_S)$, $x_Q((n-1)T_S)$ to form an output sample (here $x_Q(nT_S)$ denotes the quantized value of $x(t)$ at the $n-$th sample instant). Since samples are always equidistant, no intermediate signal value is ever required.

If, however, the input $x(t)$ is first sampled through an LCS encoder, intermediate values between samples have to be estimated. This is due to the fact that the FIR delays, τ, and the spaces between samples are no longer equal. This is explained with the aid of Fig. 3.3, which shows a typical case involving LCS encoding. All LCS samples are exact, i.e. $\hat{x}(t_n) = x(t_n)$ for all n (the hat denotes sampled values from an LCS encoder).

At time t_n, the output $\hat{y}(t_n)$ has to be computed as follows:

$$\hat{y}(t_n) = \alpha_0 \cdot \hat{x}(t_n) + \alpha_1 \cdot \hat{x}(t_n - \tau) \tag{3.3}$$

However, the value of the second required term $\hat{x}(t_n - \tau)$ is unknown since no sample was taken at

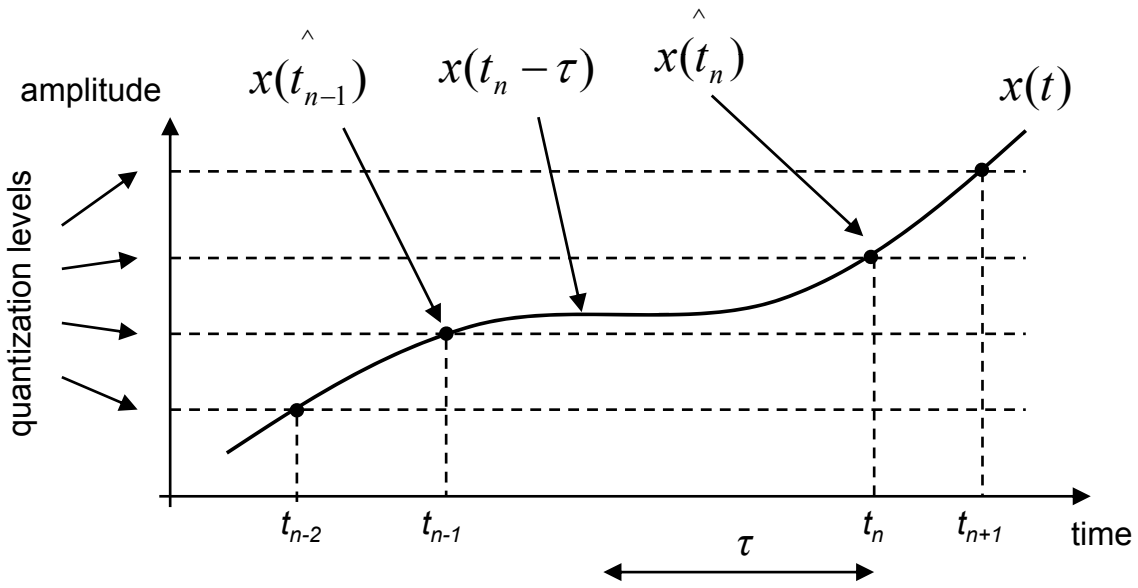

Figure 3.3: Requirement for interpolation between samples in the case of LCS encoding.

time $t_n - \tau$. An estimate of the signal's value at time $t_n - \tau$, namely $\hat{x}(t_n - \tau)$, must be provided to compute the output in eqn. 3.3.

The most simple and common way to estimate $\hat{x}(t_n - \tau)$ is to use the last sampled value prior to the time of interest. In Fig. 3.3, this would mean that $\hat{x}(t_n - \tau) = x(t_{n-1})$. This approach is called "zero-order hold" or "zero-order interpolation", since it results in fitting a 0-th order polynomial between samples. In such cases, it is typical that the interpolated values are not precisely equal to the signal values, i.e. that $\hat{x}(t_n - \tau) \neq x(t_n - \tau)$ since $(t_n - \tau)$ is typically not a sampling instant; even though sampling is accurate, error is introduced through this mechanism.

The above gives rise to the requirement for more efficient interpolation between samples, a problem common to all DSPs for asynchronously-sampled signals. Various types of interpolation have been studied, with the general trend being that computational complexity is traded off for higher fidelity (i.e. signal quality) interpolation. Besides the degenerate case of zero-order hold, other cases include linear interpolation and spline (or "Lagrange") using polynomials. All

such techniques are not limited to asynchronous sampling, and are instead widely-used in many synchronous applications as well (e.g. up-sampling an already sampled signal); however, the following discussion will be limited to LCS systems.

So far systems using LCS have been limited to zero-order hold [29], [33], [34]. As shown in [27], zero-order hold limits the SER of such systems to some extent. Even though their SER is higher than that a classical DSP system, LCS DSPs using zero-order hold still require fine amplitude resolution, i.e. many quantization levels, to obtain high SER. Some further constraints need to be imposed on LCS systems with zero-order hold, such as good matching of the quantization-level spacings (requiring calibration), as well as maintaining the analog input's amplitude below a certain maximum level (which guarantees that the error imposed by the zero-order interpolation is bounded). The proposed sinc-based interpolation techniques can significantly suppress these limitations, by relaxing the requirement for calibration of the quantization levels and only posing the requirement for precise measurement of their values (even when some small error is included).

3.4 Method for processing LCS-encoded signals

Having described LCS sampling and interpolation techniques in the previous sections, this section now considers how LCS DSPs combine them. The result is a DSP which uses the error-free LCS samples for high-quality output using interpolation.

The section first sketches the mathematical method behind LCS DSPs. It then examines the sampling conditions for this method to be accurate, and finally addresses some implementation issues that will be present in such LCS DSPs.

3.4.1 Basic principle

As a starting point, a prototype filter is considered. It is this filter whose operation LCS DSPs are trying to replicate using level-crossing sampling and efficient interpolation.

Consider an N-th order analog transversal prototype, more general than that of eqn. 3.1, with input-output relation given by

$$y(t) = \sum_{m=0}^{M} a_m \cdot x(t - m \cdot \tau) \tag{3.4}$$

Taking the Laplace transform of both sides, and using $s = j\omega$, we have, for the frequency response:

$$H(e^{j\omega\tau}) = \sum_{m=0}^{M} a_m \cdot (e^{j\omega\tau})^{-m} \tag{3.5}$$

which is seen to be the same as the frequency response of a corresponding discrete-time transversal filter with the same coefficients, and unit delay of τ. One can thus use well-known techniques for frequency response synthesis.

Were the continuous-time signal $x(t)$ to be used directly in (3.4), the output $y(t)$ would also be analog. It would be continuous-time and would contain no quantization error, since (3.4) is linear.

Instead, assume only the pairs $\{t_k, x_k\}$ (Fig. 3.2) are available, coming from LCS. It has been shown that if certain conditions are satisfied (see below), there exist coefficients C_k such that the reconstruction [106], [107], [108]

$$\hat{x}(t) = \sum_{n=-\infty}^{+\infty} C_n \cdot sinc[W(t - t_n)] \tag{3.6}$$

is exactly equal to the signal $x(t)$. In the following sections we assume that, instead of C_k, we can

find coefficients $\hat{C}_k$ such that the reconstructed output

$$\hat{x}(t) = \sum_{n=-\infty}^{+\infty} \hat{C}_n \cdot sinc[W(t - t_n)] \tag{3.7}$$

is very close to the ideal one of eqn. 3.6. Using the reconstructed version of the input, namely $\hat{x}(t)$ of eqn. 3.7, in lieu of $x(t)$ in the right-hand side of (3.4) we obtain, instead of (3.6):

$$\begin{aligned} \hat{y}(t) &= \sum_{m=0}^{M} a_m \cdot \hat{x}(t - m \cdot \tau) \\ &= \sum_{n=-\infty}^{+\infty} \sum_{m=0}^{M} f_{nm} \cdot sinc[W(t - t_n - m \cdot \tau)] \end{aligned} \tag{3.8}$$

where

$$f_{nm} = \hat{C}_n a_m \tag{3.9}$$

We can obtain regular samples of $\hat{x}(t)$ be evaluating (3.8) at $t = lT_s$, with $T_s = 1/f_s$ the sampling period and l integer:

$$\hat{y}(lT_s) = \sum_{n=-\infty}^{+\infty} \sum_{m=0}^{M} f_{nm} \cdot sinc[W(lT_s - t_n - m \cdot \tau)] \tag{3.10}$$

3.4.2 Proper sampling and reconstruction

After stating the ideal operation of an LCS DSP, the theoretical conditions under which the LCS can properly operate are now examined.

For the above to be useful, the representation in (3.6) must accurately approximate $x(t)$. The process of (alias-free) non-uniform sampling and error-free signal recovery for band-limited signals has been extensively studied for over five decades; see [101], [106], [107], [108] and the references therein. It has been proven in [107] that a function $x(t)$, band-limited to a bandwidth

W, can be reconstructed from its non-uniform samples $\{t_k, x_k\}$, if the *average* sampling rate f_s is larger than $2W$ (see [101], [107] and the references therein for a definition and discussion of average sampling rate). We will refer to this as the generalized Nyquist criterion. Among the various methods proposed for signal reconstruction [106]- [108], in this work we use sinc-based reconstruction [106], as in (3.6), where the weights are assumed chosen such that $\hat{x}$ is equal to x at the sampling instants [109]:

$$\sum_{n=-\infty}^{+\infty} \hat{C}_n \cdot sinc[W(t_k - t_n)] = x_k, \qquad \text{all } k \tag{3.11}$$

It can then be shown [109] that the representation in (3.6) becomes exact, i.e.

$$\hat{x}(t) = x(t) \tag{3.12}$$

which, by comparing the first branch of (3.8) to (3.4) gives

$$\hat{y}(t) = y(t) \tag{3.13}$$

Consider now uniform sampling with sampling period $T_s = 1/f_s$:

$$\hat{y}(lT_s) = y(lT_s), \quad \text{all } l \tag{3.14}$$

If this sampling is performed at a proper rate (i.e., $T_s \leq 1/(2W)$, no aliasing will occur in the ideal case of a perfect reconstruction; this is because $x(t)$ is properly band-limited, and thus so is its

perfect reconstruction $\hat{x}(t)$ and the linear combination $\hat{y}(t)$[1].

We thus have error-free continuous-time (3.8) and discrete-time (3.14) representations of the output, based on only the level-crossing samples $\{t_k, x_k\}$ and the filter coefficients a_m. The process is shown schematically in Fig. 3.4. The parameter W in (3.6), (3.8) and (3.10) is not unique. Tt can

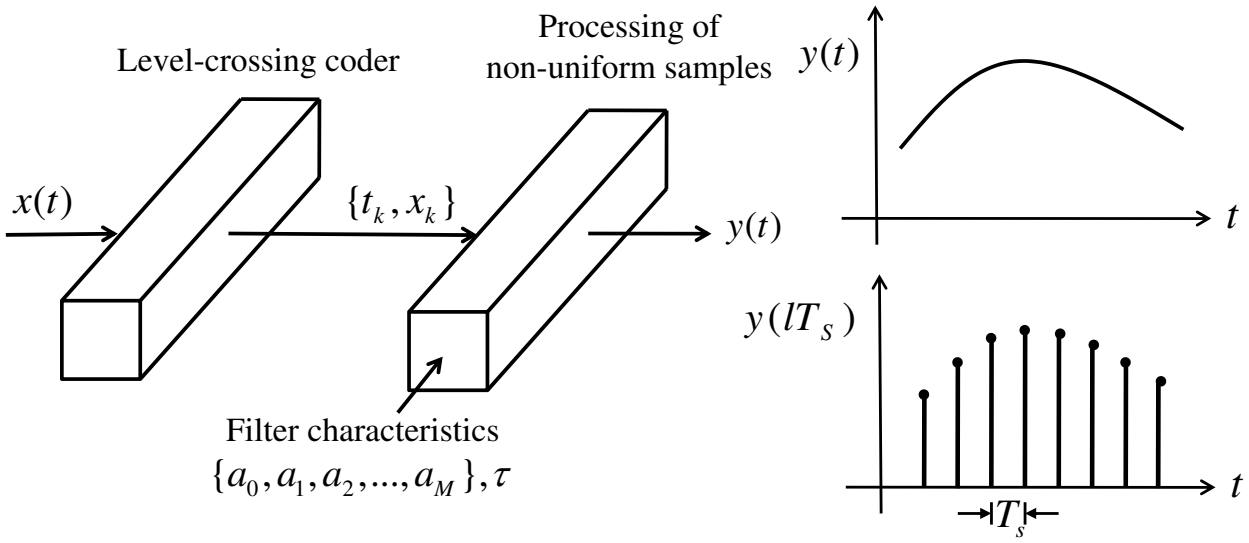

Figure 3.4: Level-crossing quantization and signal processing.

be replaced by any other value, W', satisfying $W < W' < f_s/2$, where f_s is the average sampling rate, assumed to satisfy the generalized Nyquist criterion as above. This holds because any signal band-limited to W can also be considered to be band-limited to higher bandwidths. Note here that the parameter W, which can be considered as the "reconstruction bandwidth", the average sampling rate and the output sampling rate can have different values. This decomposition is a characteristic differentiation of LCS processors from classical DSPs, where everything is tied to the sampling clock.

[1] In the case when the reconstruction is not ideal, e.g. due to LCS sampling errors or errors in the calculation of $\hat{C}_k$, then some components may be aliased back to the baseband.

3.4.3 Implementation issues

In practical cases, the interpolations that we presented can not be performed in an ideal fashion. The main practical limitations involve the finite number of available samples, as well as finite resolution in sensing the sampling times.

Finite number of samples

Perfect reconstruction as above requires an infinite number of samples, just as it does in the classical case of uniform sampling. In practice, for both cases, a finite number of samples must be used. Thus, instead of (3.8) we use a time window of N samples, starting at some time t_{n_0}. This, in lieu of (3.11), gives N equations:

$$\sum_{n=n_0}^{n_0+N-1} C_n \cdot sinc[W(t_k - t_n)] = x_k, \quad \text{all } k \tag{3.15}$$

which can be solved to determine the N coefficients C_n. These can now be used in an equation corresponding to (3.8):

$$\hat{y}(t) = \sum_{n=n_0}^{n_0+N-1} \sum_{m=0}^{M} f_{nm} \cdot sinc[W(t - t_n - m \cdot \tau)] \tag{3.16}$$

In any practical system, incoming samples (from the LCS ADC) will first be divided into groups of N samples, within each of which interpolation will separately be performed using eqs. 3.15 and 3.16. In order to guarantee the continuity of the interpolated input $\hat{x}(t)$ and hence output $\hat{y}(t)$, successive blocks must overlap by at least one sample. It has been proven that, for a sum as in (3.15), the error in representing $x(t)$ diminishes as N increases [110]. Since the output is a finite

sum of such functions, the output error inherits this property. Unlike in the case of direct signal reconstruction [106], here we operate on filtered versions of the input. For N input points, this gives us $N \cdot (M+1)$ points on which to do the interpolation. It is noted that the values f_{nm} do not correspond to samples of the reconstructed output at the instant $t_n + m \cdot \tau$.

Time quantization

In LCS quantization error is not introduced in the amplitude domain, but rather in the time domain [103]. The error can originate from sensing of the sampling instances t_k by means of a counting clock during the sampling process (or, equivalently, from limited computation accuracy). The result is that instead of the sampling instances t_k, we have quantized versions, $t_{k,q}$:

$$t_k = t_{k,q} + e_k = n \cdot T_{CLK} + e_k \tag{3.17}$$

where $T_{CLK} = 1/f_{CLK}$ is the clock period, and we assume that the absolute error $|e_k|$ is smaller than T_{CLK}.

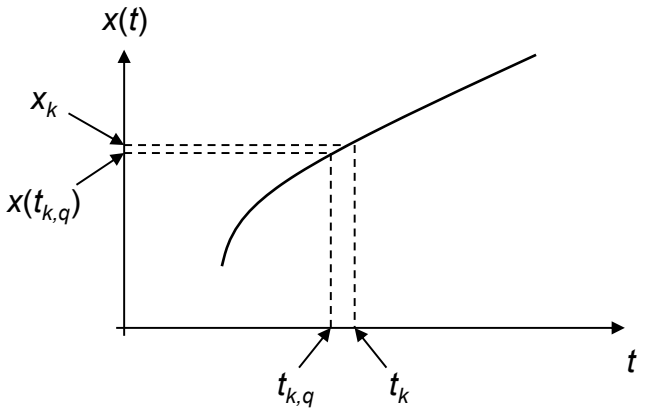

Figure 3.5: Error resulting from time quantization in LCS.

Fig. 3.5 shows a case of the quantization of a sampling instant. In this case, information

about the input signal is only obtained at discrete time points; therefore the system will resemble a very-highly oversampled digital system, but here the clock has control only over the time quantization process and not over the entire system. Furthermore, the values of $x(t)$ at the sensed time instances $t_{k,q}$ do not exactly equal the quantization levels, some small equivalent amplitude error is introduced, proportional to both the time error and the signal's slope at the sampling instance. Assuming the time error is sufficiently small, so that the value of x_k still represents the closest quantization level to $x(t_{k,q})$, the resulting pairs will be $\{t_{k,q}, x_k\}$. If these are used in the algorithm presented in this paper, the output $\hat{y}(t)$ will contain in-band aliased quantization components, which will deteriorate the output signal-to-error-ratio, SER. As will be seen later, this effect is not severe when reasonable time quantization is used.

Algorithm and system properties

The algorithm corresponding to the process just described is shown in Fig. 3.6. The input pairs $\{t_k, x_k\}$, assumed to be generated by a level-crossing coder, are first divided into pairs of N samples each. Within each group, the N samples are used to form the N-vector X and the $N \times N$ matrix D; the coefficients C_n are then evaluated in matrix form as $C = D^{-1} \cdot X$. When attempting to reconstruct the signal $x(t)$, the matrix $D = \{d_{ij}\}, d_{ij} = sinc[W(t_i - t_j)]$ has a determinant which tends to zero as the number of samples, N, increases. This can cause large errors when computing D^{-1} and suggests the use of the pseudo-inverse matrix for the calculation of the inverse of D, which is used [109] to generate the best interpolating function for $x(t)$ in the mean-square sense, i.e. minimize the interpolation error. In this work, the use of the pseudo-inverse resulted in signal-to-error ratios up to 120 dB. The algorithm terminates with the evaluation of $\hat{y}(t)$ at any desired time t, or at the uniformly spaced time instants lT_s.

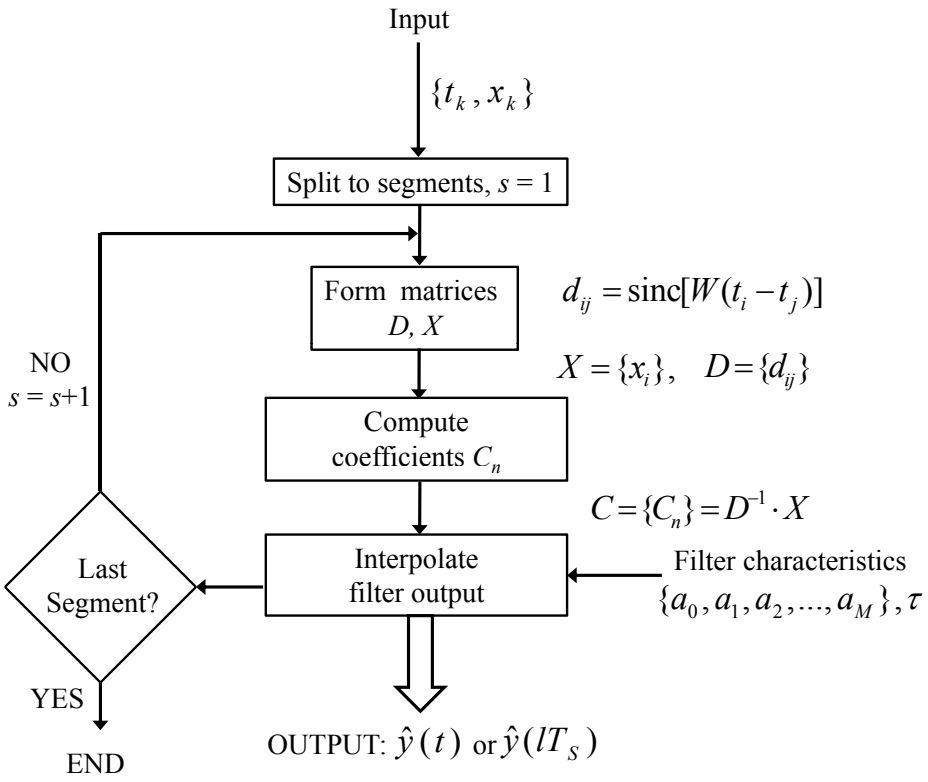

Figure 3.6: Signal processing algorithm for LCS processors.

The parameter W can, according to earlier discussion, be assigned a proper value so that the products $W(t_i - t_j)$, which appear in the calculation of matrix D, are within the numerical precision of the processor. Therefore, no further round-off error is introduced in the calculation of matrix D and vector C.

Contrary to classical DSPs, which require fine amplitude resolution to improve their signal-to-error performance, this DSP *only* requires the use of a large block size N for interpolation. Use of a large number of samples to interpolate comes at a cost of large computational complexity for finding the pseudo-inverse of matrix D, used to compute the interpolating coefficients. LCS DSPs therefore move their requirements for high-resolution sampling and filtering to the digital domain, trading off arithmetic operations for signal fidelity. This tradeoff is very different from the one of classical DSPs, whose demand for fine amplitude resolution translates to analog specifications (for power, device matching etc.). This trade off is more similar to the case of oversampled DSPs (e.g. $\Sigma\Delta$) which call for fine timing resolution to increase signal fidelity; however, LCS DSPs have the principle advantage of a signal-driven operation, contrary to $\Sigma\Delta$-based DSPs which cannot remain inactive, even during quiescent input periods.

Quantization level matching

The system we present in this chapter can potentially offer one more interesting advantage, related to the ADC quantization levels. In essence, the LCS DSP has very relaxed requirements for exact calibration of quantization levels. Such a demand impose additional hardware to a classical DSP, typically calibration hardware, which adds up to the system's power and overall area.

The operation of the LCS processor does not pose very strict requirements on the distance and exact matching of quantization levels. In classical DSPs, where most obtained samples contain

quantization error as shown in Fig. 3.1, it is strictly required to use quantization levels which are both uniformly-spaced and precisely matched, in order to limit the quantization error to 0.5 LSB. Poor matching leads to larger quantization error, degrading SER. In contrary, all obtained samples in LCS processors are error-free, and this can severely relax both above-mentioned requirements. The quantization levels no longer need to be uniformly spaced, and in fact can have any distribution. Furthermore, the levels no longer require calibration to their exact value: as long as the processor measures the actual values of all quantization levels and uses them in eqs. 3.12-3.14, no error will be introduced regardless of the position of the levels. The above relieves LCS DSPs from the typical calibration problem of the ADC quantization levels, from which classical DSP systems suffer.

3.5 Simulation results

After presenting the LCS DSP scheme in the previous section, we now show initial simulation results of a computer-based implementation. We performed MATLAB simulations for a 16-tap transversal FIR filter, configured as a low-pass filter with a 2.5 kHz pass-band and a tap delay of 125 μs. The LCS processor was simulated for both its ideal behavior, using infinite numerical precision, as well as all the imperfections described previously. Part of these results were presented in [40].

The theoretical frequency response is shown in Fig. 3.7. On the same figure, we show the response obtained by performing time-domain simulations using the algorighm above, i.e. shown in Fig. 3.6 and using eqs. 3.15-3.16, and determining the resulting amplitude. It is seen that the two responses are virtually identical. Simulations with other types of responses (high-pass, band-

pass etc.) confirm the perfect matching between the LCS processor's frequency response and the ideal FIR's.

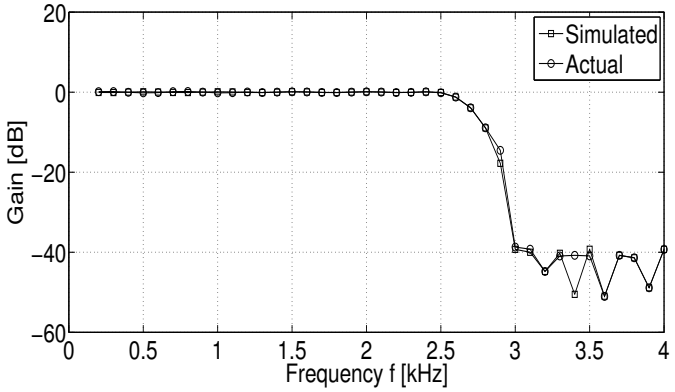

Figure 3.7: Theoretical and simulated filter frequency responses for the sinc-based LCS processor.

Fig. 3.8 shows a sample LCS output spectrum. The case shown involves a sinusoidal input of frequency 1.4 kHz, a coarse 4-bit input quantization and an output sampling rate of 10 kHz; a very fine time quantization is used, by employing full precision in the computations. We used 30 periods of a full-scale input sinusoidal signal. All input samples were used to calculate the output, i.e. the input and output reconstructions were performed by using all samples at once, without dividing into blocks of samples. For computing the spectrum, two periods were dropped on each side of the time window used, to avoid edge effects, as well as to avoid taking the FIR transient start-up behavior into account for the output spectrum. The obtained SER is 116 dB, calculated over the 4-kHz wide baseband. Since the output of the LCS processor was provided in discrete-time format, i.e. as uniformly-spaced samples, aliases of the input tone are also present around the output sampling frequency. However, no inherent aliasing is caused by the LCS processor internally: if the LCS processor output was provided in CT format, i.e. as a CT waveform, then no aliases would be present at the output.

Note that the spectrum contains both the fundamental component at 1.4 kHz, as well as three

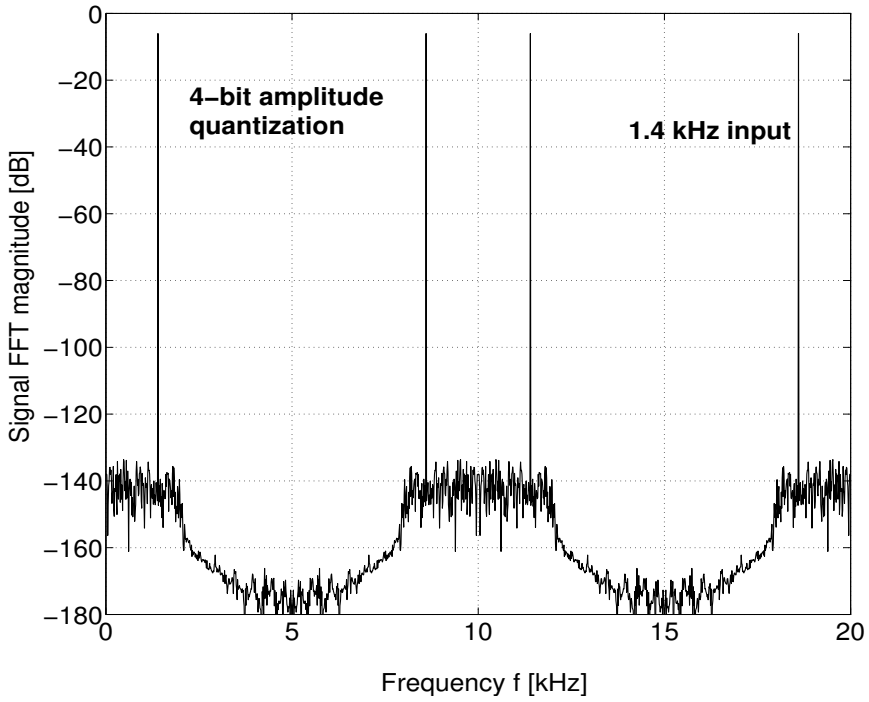

Figure 3.8: LCS processor output spectrum for a sinusoidal input, without time quantization.

aliases of it at higher frequencies. The reason behind this is that the time-domain output used for the spectrum calculation was a discrete-time version of the LCS DSP (see eqn. 3.14), with sample rate equal to 10 kHz. The existence of aliased components at frequencies $k \cdot 10 \pm 1.4$ kHz is, therefore, expected.

Fig. 3.9 presents the spectrum of the filter digital output for the same parameters as above, except that time is quantized according to a 100 MHz time quantizing clock. The in-band SER, calculated over the 4-kHz wide baseband, is now 96 dB, which is still much higher than in the case of a classical 4-bit Nyquist DSP system with a 100 MHz sampling rate with or without making use of the oversampling (in which case the SER would be 67 and 26 dB respectively). This SER is also comparable to that of a 1st order 1-bit $\Sigma\Delta$ modulator of a similar oversampling.

It is also interesting to observe how the performance of the LCS processor varies across the

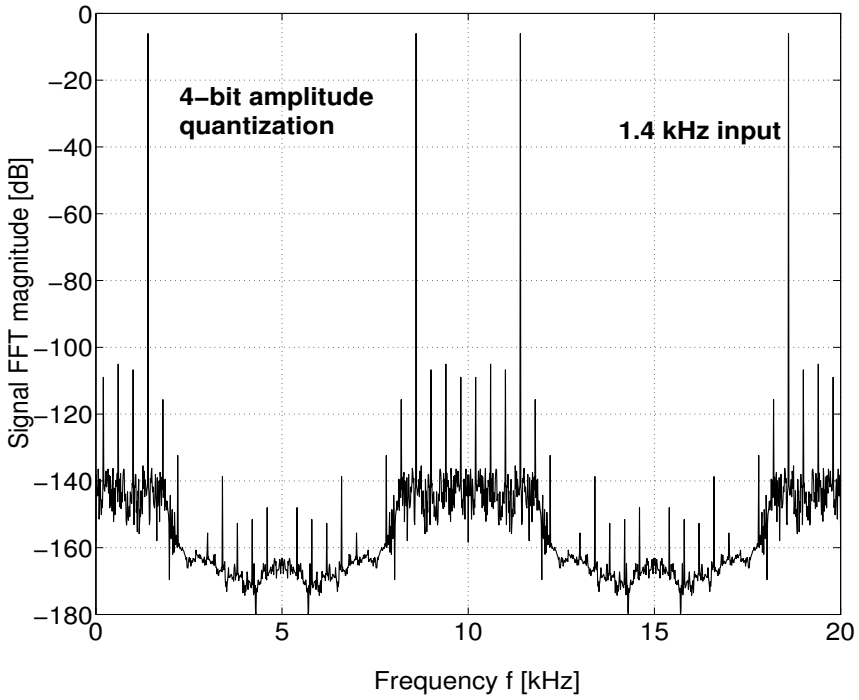

Figure 3.9: LCS processor output spectrum for a sinusoidal input, using time quantization corresponding to a clock frequency of 100 MHz.

system's design parameters. Such parameters, like time resolution and interpolating block size, will highly influence the complexity and power dissipation of the system, so noticing the associated trade-offs is helpful toward a potential practical system implementation.

Finally, Fig. 3.10 shows how the interpolating block size N influences the SER, for various time resolutions. As in all experiments a very coarse, 4-bit, amplitude quantization is used. As expected, SER increases with increasing N: using a larger block size to interpolate increases the accuracy of the processor. Time resolution also increases accuracy, since it minimizes the effective amplitude-domain error in the provided samples, shown in Fig. 3.5. Even when interpolating every 100 samples, i.e. $N = 100$ in eqn. 3.16, LCS DSPs have larger SERs than 75 dB, equivalent to 10-bit classical DSP systems.

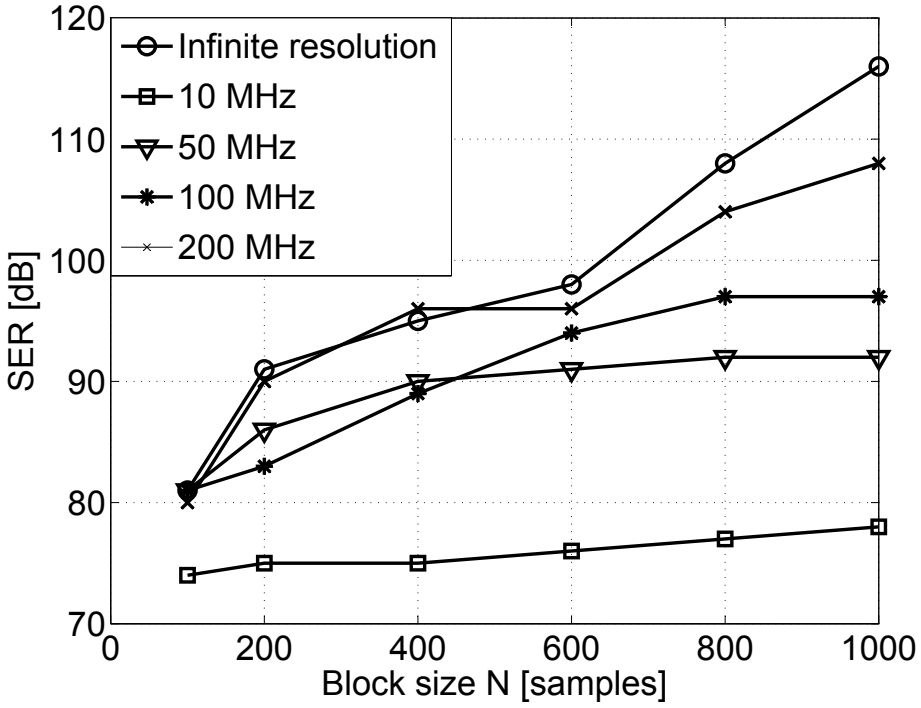

Figure 3.10: LCS processor output signal-to-error-ratio (SER) for various lengths of block reconstruction N and various time quantization resolutions. The experiments involve a 1-kHz input signal, sampled with 4-bit amplitude resolution.

3.6 Conclusions

In this chapter, we have presented a methodology for processing signals encoded using level-crossing sampling. The LCS DSP is based on highly-accurate sinc-type signal interpolation, and combines the interpolation and processing in a single operand. By using computationally demanding, but very precise processing, and making use of level-crossing sampling's properties, the LCS DSP can provide SERs in the 100 dB range, even in the presence of practical system limitations and very coarse amplitude sensing. Such processors combine very-high-quality processing with signal-dependent activity, which can lead to signal-dependent power consumption in real-life implementations.

Unlike synchronous DSP systems (Nyquist or oversampled), LCS DSPs can adjust their power

consumption to the signal activity, since the sampling rate is signal-dependent and adapts to the input. Even though the dynamic power required for the very complex arithmetic of the interpolation algorithm, as presented in this chapter, would be much higher than the one which classical DSP systems consume, it is hoped that future optimizations on the interpolation algorithm itself can lead to practical implementations of this method.

Chapter 4

Designing a Modular and Scalable CT DSP: Initial Considerations

This chapter introduces and evaluates various options for the micro-architecture of a CT DSP chip. We also show which options we chose for the implementation of the chip prototype that we present in Chap. 5. This chapter is the first of three covering a CT DSP chip implemented for this thesis [36]; this chapter covers the early decisions made before the chip's implementation, which influence the entire design.

In this chapter we deal with different options regarding various design parameters of the CT DSP chip. These parameters include the micro-architecture of the delay segments, which asynchronous protocols are used internally to the chip, and strategies and requirements for the arithmetic blocks. The decisions we made on each question is also presented here. Our choices are not targeted toward the most energy- or area- efficient implementation, at least when compared to application-optimized designs. However, they lead to designs which are highly modular and easy to replicate or translate to new specifications.

The contribution of this chapter is, therefore, a *set of decisions* on the structure of a DSP, leading toward a highly scalable implementation (which is presented in Chap. 5) of a 16-tap digital FIR filter. These choices include a localized approach for data storage, a decomposition between the CT DSP's data and timing paths, and a serial approach for the design of the timing path itself. We also deliver and justify the choice of a pipelined structure for the chip's multi-way adder. Many of these choices were different in previous CT DSPs [34], [33]; partly due to this fact, such designs are not easily scalable. We finally sketch some basic properties of an on-chip tuning mechanism for CT DSPs; this work is the first to address this topic.

The chapter begins with an general overview of a CT digital FIR, such as the one implemented in a chip as part of this thesis. We then discuss two different approaches for data movement and storage in Sec. 4.2, and two variations for the timing path of the chip in Sec. 4.3. These two sections cover the design of the entire "delay line", which is by far the largest part of the chip. We then move to considerations on the CT DSP's arithmetic blocks, i.e. multiplier and adder in Sec. 4.4 and the way to represent numbers (i.e. samples) internally to the chip in Sec. 4.5. Finally, we sketch some of the operation of the on-chip tuning for the delay line in Sec. 4.6.

4.1 Overview of the CT digital FIR filter

In this section, we provide an overview of the target architecture. The object we are aiming to design is a CT DSP core, implementing a CT digital FIR filter. The filter is shown in Fig. 4.1.

Conceptually, the structure and operation of the filter is fairly simple. It has a single input channel, coming from an ADC. As we will show in Chaps. 5, 6, various ADCs, such as CT or DT, can be used. The chip also has a single output channel, going to an asynchronous DAC. Both the

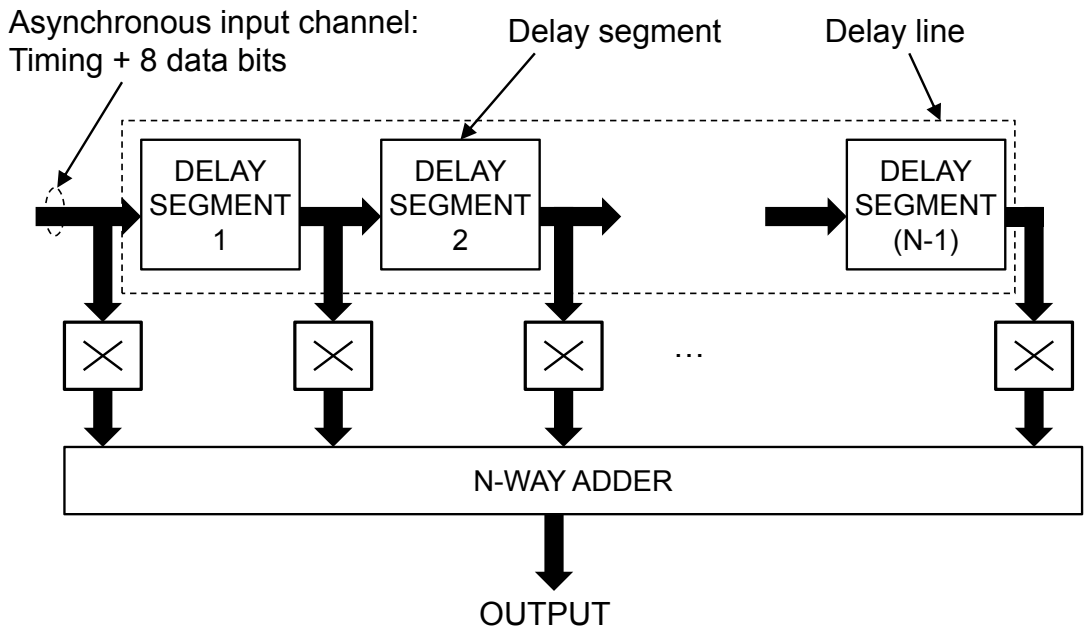

Figure 4.1: Abstract view of a CT digital FIR filter.

input and the output channels are asynchronous ones. The output ideally equals a weighted sum of multiple delayed copies of the input, which are generated by using delay segments.

Each channel contains a data field, carrying the sample values, and a timing signal. As in all CT DSPs, the FIR must precisely preserve the time differences between consecutive input samples, since these differences carry important information on the sampled signal.

Throughout the remaining chapters, we will use the following terms. Referring to Fig. 4.1, the *delay line* is the block inside the FIR which creates the multiple delayed copies of the input channel. It is composed of multiple *delay segments*, where a delay segment is a block which provides a constant delay to a data channel; the latter consists both of a timing signal and of multiple data bits. As we will explain shortly, the delay segment is decomposed to a *timing path*, which only handles the timing of samples, and a data path, i.e. a memory. The timing part consists of multiple *delay cells*. The samples' data bits are stored into a memory, either locally or globally. However,

we will be referring to the delay of a segment, implying the delay provided by the segment's timing part.

The goal of the remaining sections of this chapter is to present various options for the building blocks shown in Fig. 4.1. Furthermore, we will present the choices we made when designing the chip prototype shown in Chaps. 5, 6. We begin with the delay line and delay segments in Sec. 4.2, first showing that each delay segment is decomposed to separate timing and data paths, and then considering two micro-architectures for each. Then we address the arithmetic blocks (Sec. 4.4) and numerical issues for representing samples internally (Sec. 4.5), and finish this chapter with considerations on tuning of the delay line (Sec. 4.6). The latter is not shown in Fig. 4.1, but is an essential component of the chip.

4.2 Data movement and storage

This section discusses the two alternative approaches for storing sample data internally to the chip. The decision on the movement of data in the CT DSP can largely influence the micro-architecture, power and scalability of the chip. Unlike previous CT DSP implementations [46], [33], the target here is to handle data wider than a 1-bit or 3-bit stream. Furthermore custom solutions for small-width data paths like the one in [34], which included 2^N different paths and 2^N separate delay lines for all wires have significant overhead in power and area and are therefore not suitable, are not ideal for general-purpose processors handling wider data paths.

Part of the contribution of the CT DSP chip [36], which is presented in chapter 5, is novel a decoupled architecture for the timing and data paths. Essentially, each of the delay segments in Fig. 4.1 consists of separate paths for timing and data for each segment. The data path is

simply a memory that stores data. Using this approach, we clearly differentiate from previous works [46], [33], in which data bits always travel down the FIR along with the timing signals. The resulting decoupled architecture eliminates all movement of data within a segment; the result is a highly scalable approach with bounded area and power overhead.

The data of each sample is separated from the timing, gets temporarily stored (i.e. enqueued) in an SRAM memory, and is picked up only when required. This significantly reduces the amount if times that each sample needs to be stored, compared to the case in [46] and [33] where it was passed down many times in each delay segment (without a need for this, since samples are only used at the beginning and end of each sample).

Two different methods for data storage are examined. These storage solutions are: (a) a fully global storage and (b) a per-segment storage. In the fully global approach, the individual memories of all segments are united to one monolithic memory, common for all the delay segments. In the second, i.e. the per-segment, approach the memories for each delay segment are separate.

In this section these approaches are presented and compared for size, complexity and resulting power consumption. As a result of this comparison, we chose the local per-segment approach, i.e. (b), for designing the chip.

4.2.1 Global storage

In this approach, a single shared memory is used to store the input samples to the FIR. Essentially the memory blocks for all delay segments are combined for one global memory. Each input sample is en-queued once, upon its entrance to the FIR, and read out multiple times, once for each FIR tap, as controlled by the timing signal exiting the segments' timing parts. Fig. 4.2 shows this global

memory approach.

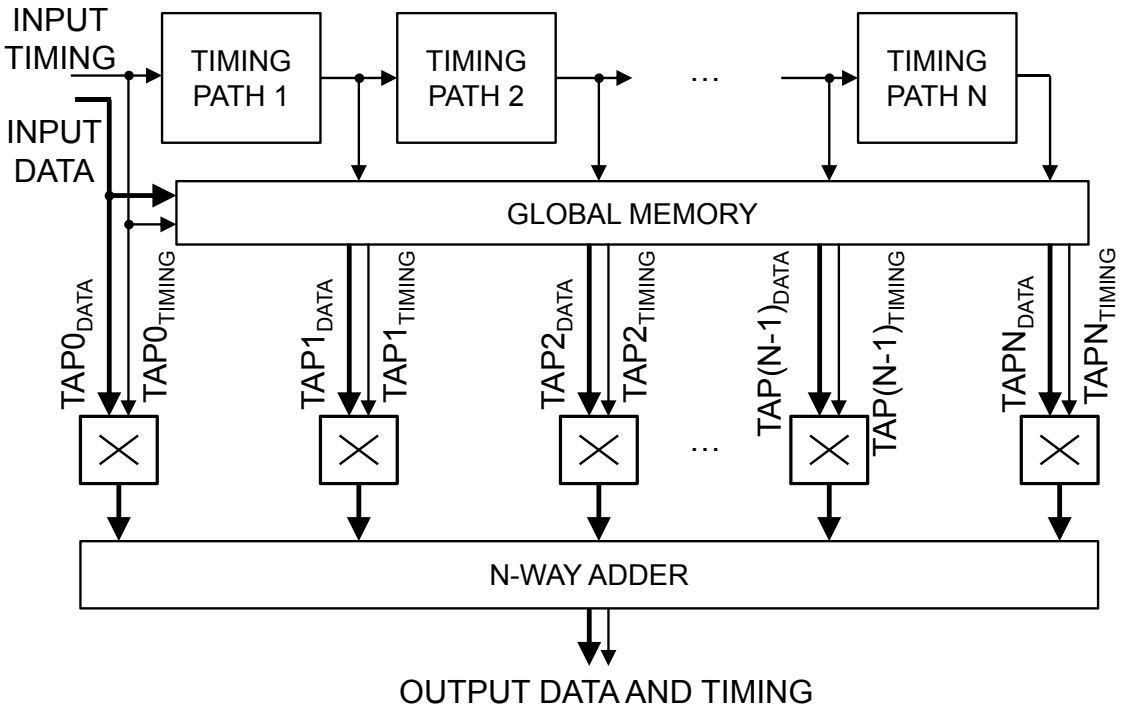

Figure 4.2: Global-memory approach for CT DSP data management.

The size of the memory must be sufficient to store all samples arriving at a maximum input rate. Therefore, it must have one entry for every delay cell inside the timing part of all N delay segments. From an operation perspective, each sample will only be stored once and read out N times, increasing the energy efficiency of the structure.

However, the particular architecture poses serious difficulties for the design of the global memory. In this approach, multiple segments can concurrently be requesting samples from the global memory. Therefore, the global memory must be able to support multiple concurrent read operations, in principle N concurrent reads. Failure to do so will lead to some samples being read out with delays, causing timing deviations to the CT DSP samples, This delay will increase if the memory can only support a few concurrent read operations, since more samples can be kept waiting to be read out, increasing the congestion time. Such a specification, i.e. for multiple concurrent read

operations, is very hard to meet, increasing the complexity of the global memory to a large extent.

4.2.2 Local (per-segment) storage

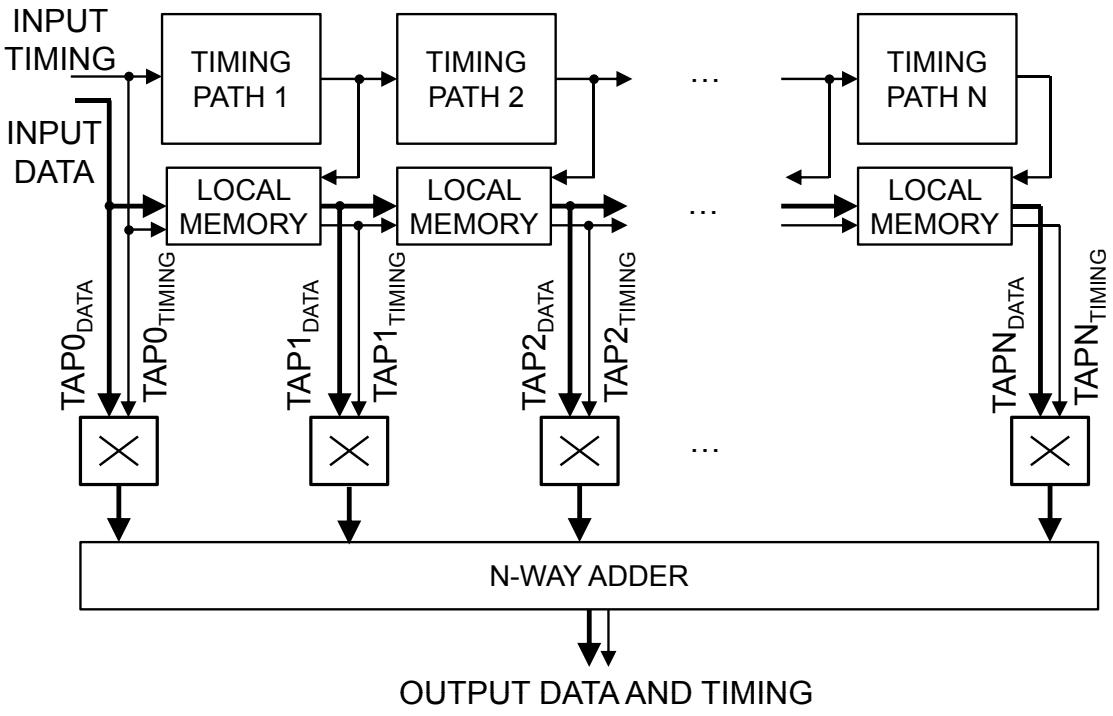

Figure 4.3: Per-segment memory approach for CT DSP data management.

In the second approach, which is the one employed in the design of the CT DSP prototype chip, data is stored locally, on a per-segment basis. The same amount of memory used in the centralized approach is now broken into N individual memory blocks, one for each delay segment.

From the view of macro-level operations, this approach is less efficient than global storage. Each sample will be stored and retrieved N times, one for each segment. Compared to the global storage, each sample is stored $(N-1)$ more times but read out the same number of times.

However, this approach is *much more modular*; this benefit also translates to more relaxed specifications for the memory blocks. Each memory block only needs to be able to store and/or read *a single* sample at a time. This is a much easier-to-meet specification than the one for global

storage, where multiple concurrent operations are required in principle. This approach is also very scalable: adding more FIR taps does not have any impact on the specifications of the memory itself, besides the simple addition of more *independent* memory blocks. In contrast the global approach would require scaling the single global memory to incorporate both an additional storage requirement as well as more concurrent memory access operations. As a result, we chose the per-segment approach for the designed modular CT DSP chip, which we will present further in this work in chapter 5.

4.3 Micro-architecture of the timing path of delay segments

In the last two sections, we presented a decoupled architecture for timing and data, as well as two variations for the micro-architecture of the data storage blocks. This section first presents two variations for the micro-architecture of the timing path in each segment, then two options for the asynchronous communication protocol of the segment. These two aspects are orthogonal.

Each delay segment has its own timing management path, which must provide a fixed delay between the input and output of the segment. The delay of each timing part typically has a much larger delay than the minimum sample spacing; therefore, each timing part must be able to hold multiple samples inside it at any time. This case does not always hold in the case of classical DSP systems, where usually the segment delay is equal to the sample spacing, which in turn is equal to the clock period.

We first address the two options for building such delay segments' timing parts. We start with the structure of the segment, explaining why a serial structure was chosen over a parallel one. Then we move on to the asynchronous protocol of the individual delay cells, explaining why the version

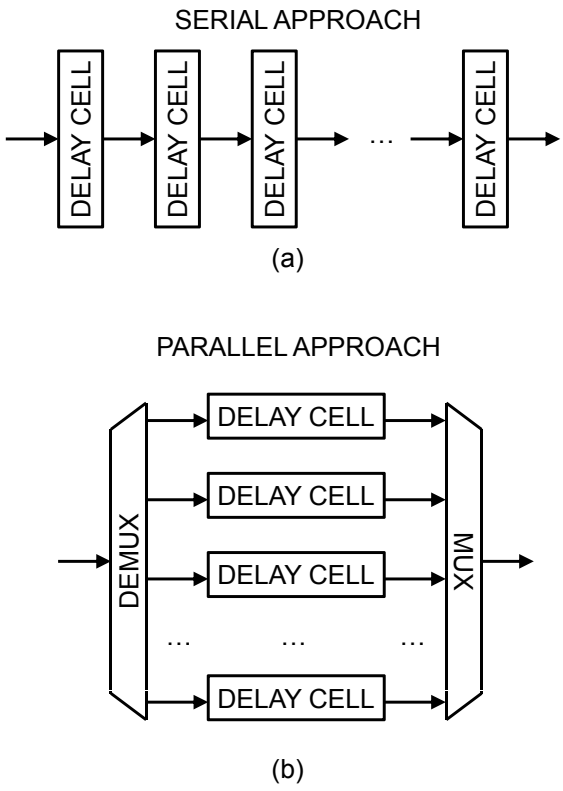

Figure 4.4: Approaches for a CT DSP delay line: (a) serial and (b) parallel.

using a 4-phase protocol (which was used in previous designs as well) was preferred for this design over the 2-phase protocol.

4.3.1 Serial vs. parallel structure

Two structures are considered for building a timing path, serial and parallel [28], as shown in Fig. 4.4 (a) and (b) respectively. In the former approach, each sample travels through all delay cells from the input to the output of a segment, while in the latter each sample travels through a single (but different) cell with much larger delay. The two approaches must be compared for dynamic power, area (which also influences leakage power) and programmability, while keeping all other factors constant, such as line jitter and overall delay.

As we carefully analyze in Appendix A, the serial version is a more reasonable choice for building modular CT DSPs, assuming delay cells similar to the ones of the design in [33]. When the overall segment delay and variation of the segment's delay, i.e. the jitter, are kept equal in both structures for a "fair" comparison, the serial approach has much smaller area for large delay values and fine line granularities, i.e. number of cells in each FIR segment. The parallel approach has a significant advantage in terms of power, since each sample only goes through a single cell, tus avoiding the multiple asynchronous hand-shakes of the serial approach; however the area penalty of the parallel approach is even larger than the corresponding decrease in power, pushing the overall decision in favor of the serial approach. The detailed analysis of Appendix A thoroughly explains the reasoning behind this decision.

Furthermore, serial approaches have a great advantage when tuning and modularity is concerned. Using a serial structure, we have two "knobs" for programming for an overall segment delay: (i) the individual delay of the cells, assumed to be equal here for all cells, and requiring analog tuning, as well as (ii) the total number of cells in each segment, which can be done through digital programming. In contrast, in the parallel approach each samples only goes through one delay cell. The only tuning "knob" in the parallel case, therefore, is the delay of this one cell. As in the serial case, the delay cell can be tuned in an analog fashion. However, by only using one method for tuning the delay of an entire segment, the range of delays that can be achieved by the parallel case is much smaller, when compared to the serial case.

By design experience, obtained through the design of the CT DSP chip with on-chip tuning presented in Chap. 5, the delay of a single delay cell can be programmed within one order of magnitude, i.e. the maximum and minimum delay can only differ by a factor of 10. To achieve a large range for the delay of an entire segment, the extra tuning knob of the number of delay cells

in each segment, of the serial approach, proves to be very helpful. This knob is not available in the parallel approach; this limits the delay programmability of the latter.

Finally, to preserve the ordering of samples in the parallel approaches, which go through different paths, the variability of the cells in the parallel approach must be made very small. In the parallel approach,

Only for such a very small variability and jitter can the parallel implementation of a segment guarantee that the ordering of samples is preserved, even for minimum-spaced samples which get affected by jitter more. Such a demand poses additional constraints to the delay cells, further increasing the difficulty in designing the parallel approach. The serial approach clearly does not suffer from this problem, since the pipelined structure always presents the order of events, i.e. samples.

4.3.2 Delay cell protocol: 2-phase vs. 4-phase

The choice of the asynchronous protocol for the timing path, and hence that of the individual delay cells, is a decision that can influence the entire system. Even though the implementation in [33] used a 4-phase protocol which was highly efficient, we also examine the alternative of a 2-phase protocol as part of a system-level design.

A 2-phase delay cell signals samples as both rising and falling transitions. One sample is communicated using a rising request and rising acknowledge signal and the signals do not reset to their initial value after completion of hand-shaking. The next sample uses the falling edges of these signals. To generate the precise amount of delay, a 2-phase delay cell [34] either charges a capacitor through one calibrated current source or discharges it using a second current source,

which is also calibrated.

In contrast, a 4-phase delay cell uses signals which reset to their initial values at every sample. It only uses one current source to charge the cell's capacitor; the latter is fully charged and discharged to reset at every sample. Clearly this approach is less energy-efficient than the 2-phase one, due to the increased amount of signal transitions. However, the 4-phase cell is much easier to tune, as well less complex and hence occupies less area.

The 2-phase variation of the delay line results in increased area for the delay cells. This is effectively due to the need to have symmetric hardware for the two different types of events, handling samples encoded as rising and falling edges. In the case of the delay cell, where delay is generated by charging a capacitor through a current source, 2-phase protocol translates to each cell having two current sources, each implemented using large transistors, to both charge and discharge a capacitor in a controlled manner. The 4-phase variation, which was the version we chose for the chip prototype, only requires a minimum of one current source, hence less area. Finally, the 4-phase variation led to half the tuning required for 2-phase cells, given the existence of only one current source.

4.4 Arithmetic blocks: multipliers and multi-way adder

Having completed the discussion on the delay line, we can now move on to the arithmetic units. There are two different units as shown in Fig. 4.1, an asynchronous multiplier and a $(N+1)$-way adder. In the case for the chip we designed as part of this thesis, $N = 15$.

The multiplier and multi-way adder can also affect the performance of the CT DSP. Their timing characteristics, such as latency and throughput, can cause disturbance to the arrival of the

samples to the DSP's output. Furthermore, the chosen structure for the arithmetic blocks should enable easy scaling with respect to the number of FIR taps, without severely affecting the blocks' timing characteristics.

4.4.1 Asynchronous multiplier

The requirements for the asynchronous multiplier are more relaxed, when compared to those for the multi-way adder. Each multiplier processes one segment's output, with consecutive samples arriving at the same maximum rate as the input. The spacing between consecutive samples, for the purposes of the chip we designed, is limited to a few tens of nanoseconds. The only demand for the multiplier is to be able to handle a single sample within this time, i.e. before the next sample arrives. Such a requirement is very easily met, given that a multiplication can be completed in the order of a few nanoseconds.

4.4.2 Multi-way adder

The adder's structure and performance is much more critical, compared to that of the multiplier. We first analyze the effect of the adder's cycle time on the CT DSP samples, and then compare two alternative approaches for the adder's structure, a monolithic and pipelined.

The multi-way adder, contrary to the multiplier, receives $N+1$ distinct input streams from the taps, which are in principle entirely uncorrelated in time. Samples from different taps can arrive at the adder input with arbitrary spacing, from the one extreme of each sample arriving alone to the worst-case where all taps near-simultaneously have new samples. The adder must safely accommodate all such cases. As such, the adder is the most critical arithmetic unit. The only

requirement for the multipliers is to operate faster than the minimum sample spacing at their input, which is the same as the input sample rate. This was easily satisfied in this design.

In most cases, samples will arrive at the adder spaced by relatively small time amounts. In this scenario, some samples will be "accepted" by the adder, while the ones which arrive shortly after will be stalled for a small time, to be handled by the next addition operation. This processing time window depends on the cycle time (or the inverse of this, which is the throughput) of the adder. Fig. 4.5 shows the simulated probability of sample congestion for various cases of the adder cycle time. The simulation was performed in MATLAB/Simulink: an analog sinusoidal input was first passed through an ideal LCS ADC, i.e. an ideal quantizer, and the quantized input was fed to a 16-tap FIR filter with segment delay of 20 μs. The times at which all FIR multipliers presented new samples to the multi-way adder were recorded, and congestion was reported every time two or more samples at the adder input were spaced by less than a pre-determined amount. The probability of congestion was then calculated as the number of samples which led to congestion, divided by the total number of samples (at all taps). As expected, a larger analog input frequency increases the average sample rate, through the level-crossing encoding, and this also increases the adder congestion. This probability increases linearly at first with the input frequency, and effectively also with the average sample rate, and tends to saturate for high rates. Similar characteristics were observed with other input encodings too.

Adder congestion can cause timing disturbance to the CT DSP's output. In particular, this means that the delay that certain sample undergo before reaching the adder's output will be slightly increased. The stalling of a sample at the adder's input is equivalent to adding a small delay to this sample, before reaching the adder's output. This delay is considered random, its effect is equivalent to jitter and highly depends on the adder's throughput, or inversely, on its cycle time. An adder with

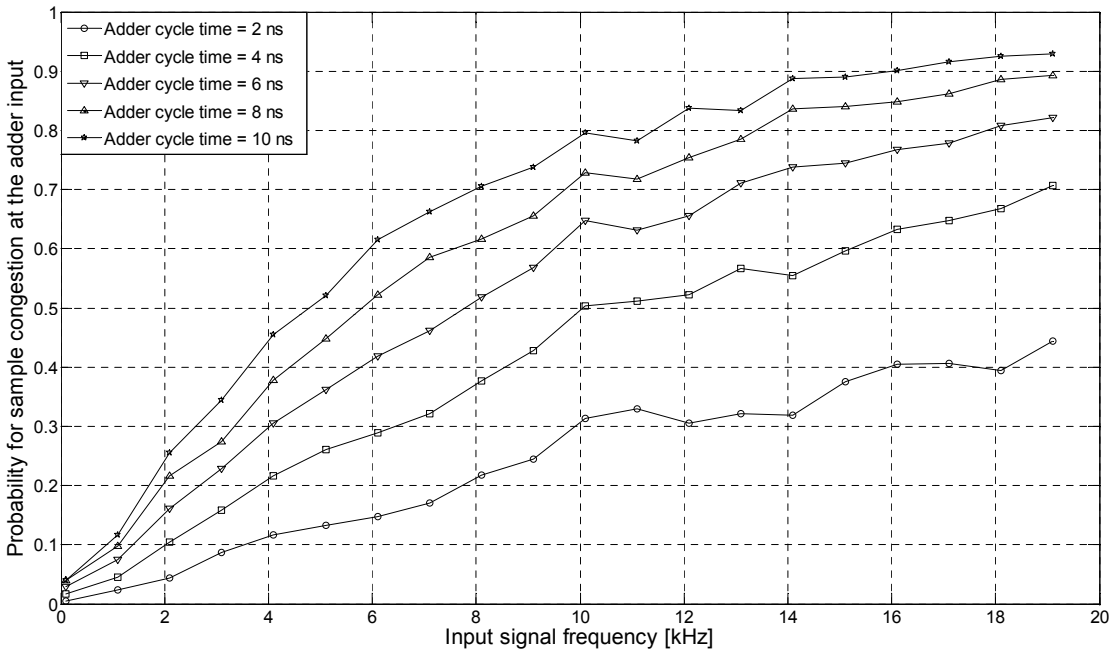

Figure 4.5: Probability of sample congestion at the input of the CT DSP multi-way adder for a 16-tap ideal FIR filter (with no delay-line jitter), fed with a LCS-encoded sinusoidal signal of maximum amplitude, for various adder cycle times.

high throughput, i.e. small cycle time, will clear up with one sample quickly and accept the new ones with small time perturbation, effectively distorting the FIR output by only a small amount.

In case of congestion, the average jitter added to the stalled samples is equal to half of the adder's cycle time. The latter is in the order of a few nanoseconds. Hence, the equivalent jitter introduced via adder congestion is much smaller compared to the one of the delay line for the designed FIR prototype, and therefore its effect on signal quality was not considered here. For a thorough analysis on the effects of delay segment jitter to the output of a CT DSP, see the work in [111]. However, sample congestion can have different consequences in other cases, for example in an IIR structure. Designing a modular adder with small and controllable cycle time is one more contribution of the thesis to formalizing the design of a CT DSP which is re-usable and scalable.

Two major design decisions influence the adder cycle time: (i) the choice of a monolithic

or pipelined structure, as well as (ii) the choice of the adder implementation. The monolithic design, i.e. where all $N+1$ tap outputs are added together in one cycle, is less scalable and has a larger cycle time. A pipelined version decreases the cycle time, while requiring slightly increased complexity to handle the communication between different pipeline levels. Since this added complexity is typically much smaller compared to the entire CT DSP, we chose a pipelined version for the chip's adder. The large reduction in cycle time, through pipelining, enables the use of slower adder implementations.

As described in detail in chapter 5, simple ripple-carry adders (RCA's) were used in the prototype chip. Given the usage of pipelining to reduce cycle time significantly, RCA's can still be used and result in a cycle time comparable to a previous implementation [33]; the latter used much faster carry-look-ahead adders (but a monolithic addition scheme), as well as a faster technology.

To reduce the adder cycle time significantly, we chose a 4-*level deep pipelined implementation*, organized as a *tree of* 2-*way adders*. Therefore, each individual RCA only adds two numbers and can operate with relatively good speed.

4.5 Internal arithmetic considerations

In the previous sections, design issues for the asynchronous multiplier and multi-way adder were presented. In this section, we finalize the discussion on the arithmetic part of the CT DSP chip by addressing the issue of *internal numerical representation*. In particular, two decisions are needed on the arithmetic portion of the CT DSP: (i) the choice between "absolute" and "relative" sample encoding, as well as (ii) the numerical representations for samples.

Samples can be encoded as relative values or absolute values. Relative-type encoding involves

using at each time the difference between the previous and next sample within the FIR. Absolute encoding involves using the difference between each sample and a fixed reference sample, e.g. 0.

Previous CT DSPs have used both types of sample representations. Absolute encoding was used in [34] as well as in most synchronous designs. Relative encoding was used in [46], [33] as an optimization, since consecutive samples only differed by 1 LSB due to the use of level-crossing sampling. Such an environmental constraint will not hold in general-purpose CT DSPs, such as is targeted in this thesis. While relative encoding has the benefit of providing change-based operation, eliminating activity for consecutive samples with identical values, it also has the potential danger for a *persistent distortion* to the FIR's output if one of such samples is lost or processed incorrectly in any of the FIR's blocks (delay segments, taps and adder). Given this, as well as that relative encoding does not offer any reduction to the datapath width for non-level-crossing ADC formats, we picked absolute encoding for the chip design.

The second decision is the numerical representation of numbers, internally to the filter. The available choices include signed binary, offset-binary and two's complement. The choice of two's complement was easy in this case: using two's complement guarantees that intermediate sums are allowed to overflow, but the final sum can be corrected without distortion.

4.6 On-chip tuning

The remaining part of the CT DSP is the on-chip tuning for its delay segments, which constitute the delay line. While not explicitly shown in Fig. 4.1, delay segment tuning was implemented in the chip prototype which is presented in the next two chapters. This section addresses some important aspects of the tuning, as well as some implications that tuning has on the delay segments.

All CT DSPs require tuning to guarantee their proper functionality and correct frequency response. Instead of relying on a clock to set the chip's timing notion, as in synchronous designs) CT DSPs have an internal mechanism to generate timing internally, which is the timing path of the delay segments. This path uses calibrated delay cells, which provide a fixed latency through the controlled charging of capacitors through calibrated current sources. Furthermore, each FIR segment has multiple delay cells, in a serial configuration as we justified previously in this chapter. Details on the design of the cells and segments have been given in brief in this chapter, but will be presented in greater extent in the next chapter.

Tuning involves mechanisms to adjust both of these parameters in a CT DSP, i.e. a method to calibrate the current source in the delay cells, as well as to adjust the number of cells in each FIR segment. Using tuning, two conditions can be guaranteed for the delay segments. First, the individual cell delay in each segment must be smaller than the minimum spacing between two consecutive samples, to ensure that congestion does not occur and the spacing between samples is preserved. The second condition is that the overall delay in each FIR segment in Fig. 4.3 is of a proper overall value.

The requirement for wide-range tuning had an impact on the design of the delay-line timing path. To adjust the number of delay cells in each FIR segment, the cells were organized into binary-weighted groups within each segment. Each group had a binary-weighted number of cells: in the designed chip there are groups of 2, 4, 8, ..., up to 256 cells, and each group can be selected on or off using MUXes and DEMUXes, as presented in the following chapter. Therefore, each segment involves a 7-bit digital setting which can adjust the segment to include any number of cells. The extra cost of the MUXes and DEMUXes for each group was found to be minimal compared to the large number of delay cells required for this chip.

In addition, all cells require analog tuning to set their delay. Such tuning involves adjusting the analog charging current within each cell. The alternative option, i.e. adjusting capacitor values in the cells, is not practical. Changing the currents can easily be performed by providing only one current for multiple cells, e.g. one segment, and using current mirrors to distribute to all the cells. This problem resembles the tuning problem of all analog filters, and master-slave-type tuning was chosen here. In this type of tuning, a replica delay segment is compared to a programmable reference delay, whose value is well defined. Based on the comparison outcome, the delay of the line is adjusted by changing the value of a single current, which is then mirrored to all delay cells in the chip. By designing this current as a digitally-programmable sum of binary-weighted currents, only a few bits are needed to span a wide range of current values, and hence delay values.

4.7 Summary

This chapter covered a variety of system-level considerations for the design of a scalable, general-purpose CT digital FIR filter. It also presented choices for the design of the chip prototype that follows in the next chapter. These considerations serve as a good starting point before the design of any such system, since their outcomes will highly influence the chip's structure, operation, ease of design, power and performance. These decisions also affected many low-level implementation details.

In this chapter, different choices were explored for data storage internally to the FIR, as well as two different micro-architectures of the timing path of the delay segments. We justified the use of a pipelined structure for the FIR's multi-way adder, and addressed issues on numerical representation of samples, as well as for the chip's automated tuning blocks.

Our decisions were motivated by the need to design a highly programmable and scalable CT digital FIR filter. All decisions were such to optimize the scalability of the filter for different parameters, such as operating frequency and data bit-width. We believe that they are the optimal decisions in that direction. The following chapter presents the implementation of the digital CT FIR prototype, as the outcome of the decisions of this chapter.

Chapter 5

A CT DSP Chip with Open Input Format

This chapter presents the complete design of a modular CT DSP core chip with open input format, i.e. the ability to handle digital inputs of widely varying rates and modulations. The resulting chip prototype was implemented in a 130 nm IBM CMOS process. It was used as part of an ADC/DSP/DAC system which can maintain its frequency response intact while using multiple different ADC formats, each with a different sample rate. The system can process both asynchronous and synchronous formats, even while the sample rate is varying, without requiring any internal adjustments or system pauses. Such a property is impossible for synchronous DSPs systems, given that in the latter the response is tied to the actual sample rate. Hence the proposed system can be viewed as a complementary structure to the synchronous technique. To the best of our knowledge, this chip is the first one, either synchronous or CT, able to maintain its frequency response for different rates, without requiring any reconfiguration.

The chapter begins with a short look at the contributions of this chip's design in Sec. 5.1. Some asynchronous cells used throughout this design are first presented in Sec. 5.2. After an overview of the chip's operation in Sec. 5.3, the detailed description of the chip's design will follow. First

the timing path of the delay segments is presented in Sec. 5.4, followed by the segments' memory blocks in Sec. 5.5. The asynchronous multiplier and multi-way adder are then introduced in Secs. 5.6 and 5.7, respectively. The design of the automated delay tuning follows in Sec. 5.8, and the tuning interface blocks are shown in Sec. 5.9. Finally, Sec. 5.10 summarizes the design challenges and contributions of this work.

5.1 Contribution of the designed chip

The new chip is a CT DSP core which can process a variety of digital signals without requiring any internal adjustment to compensate for the type and rate of each signal. The chip can be used to construct ADC/DSP/DAC systems with a variety of different ADCs, of different encodings and sample rates; the frequency response of such a system using this chip will be identical in all cases. As opposed to previous CT DSPs [33], [34], this chip is *not restricted to using just one digital format*; hence this chip is the first proposed design of a general-purpose CT DSP.

There are several individual contributions to design steps for CT DSPs. First, we introduce a method for decoupling the data and timing management paths in each FIR delay segment, which is critical for the design's scalability. We also introduce enhanced delay cells with *multiple operation modes*, as well as a complete asynchronous design for the arithmetic blocks. Asynchronous methodologies enable energy-efficient data management inside the adder by eliminating unnecessary data movement, while also handling the demand-driven operation required in a continuous-time digital system. Furthermore, a safe multiplier-adder interface is introduced, designed using systematic asynchronous protocols to avoid glitches and meta-stability problems, avoiding a potential flaw in a previous design [33]. Finally, an automatic on-chip tuning mechanism for the delay

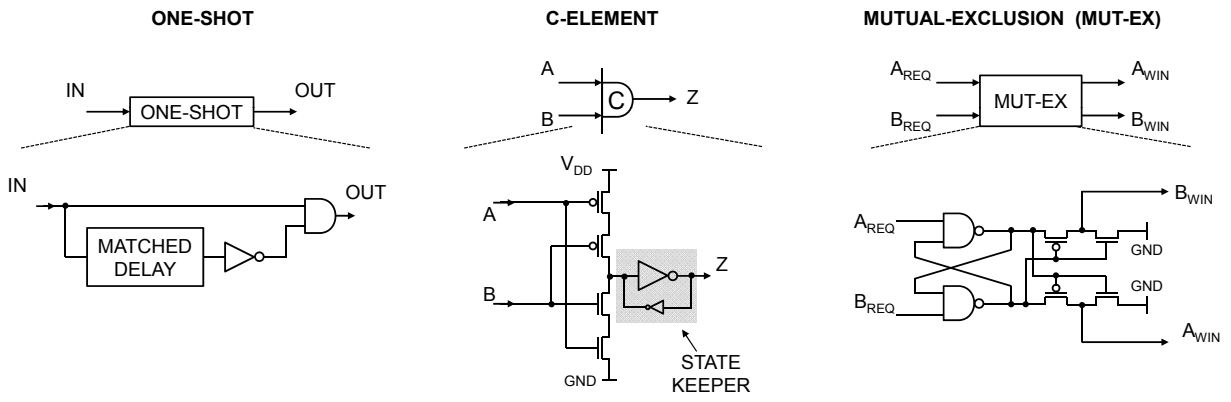

Figure 5.1: Asynchronous cells used for CT DSP chip design..

lines is incorporated, both for adjusting analog charging currents in all the delay cells, as well as digital control to program the exact number of cells in each delay segment individually.

A current limitation of this design is its inability to interface with a synchronous computer at the DSP's output, due to the asynchronous nature of the output data. This limits the use of the chip to real-time ADC/DSP/DAC systems, i.e. systems where data cannot be temporarily stored (and further processed) using a synchronous computer before going to the DAC. In future work, we aim to extend the interface design to handle a synchronous DSP output.

The following sections discuss the design of the implemented CT DSP in detail. Before presenting the chip itself, three digital cells are presented, common to many asynchronous digital systems, that are widely used in this chip's design.

5.2 Basic asynchronous cells

Apart from static CMOS gates, several additional cells were used in the design of the CT digital FIR. Three of these cells, shown in Fig. 5.1, are commonly used in asynchronous design, so they are presented here in greater detail.

The "one-shot" block shown in the left part of Fig. 5.1 is a self-timed pulse generator. It is triggered by a rising edge and produces an active-high pulse of fixed width (set by the matched delay), which self-resets without any additional input. The negative edge of *IN* has no effect. For correct operation, the width of the pulse at the input must be larger than the matched delay. Such a block is typically used in asynchronous designs to first set (or reset) RS-latches and then immediately put them into the "hold" state to wait for their next input.

The Muller C-element [112] is a sequential component which operates using hysteresis. Its output Z is equal to its inputs A and B if they are equal, else it holds its previous value. It is widely used in asynchronous circuits to synchronize between distinct concurrent signals.

The mutual-exclusion element, or "Seitz arbiter" [113] is used to provide access to a resource to only one or two concurrent requests. By using cross-coupled NAND-gates, as shown in the right part of Fig. 5.1, it is guaranteed both that *only one* of the two outputs which denote the access grants to the requesters will be active at any time, and the winner's output will make a glitch-free, i.e. monotonic transition high.

By using such asynchronous cells as well as formal asynchronous design principles, the designed parts of the chip (e.g. tap multiplier) are guaranteed robust functionality and are free of typical problems arising from ad-hoc design methods, such the potential hazards reported for [33].

5.3 CT digital FIR chip overview

Fig. 5.2 shows the top-level view of an ADC/DSP/DAC system containing the new chip. The chip can be connected to a variety of ADCs, with sample rate up to 20 MHz, and processes samples without using a synchronous clock. An asynchronous DAC converts the DSP output to an analog

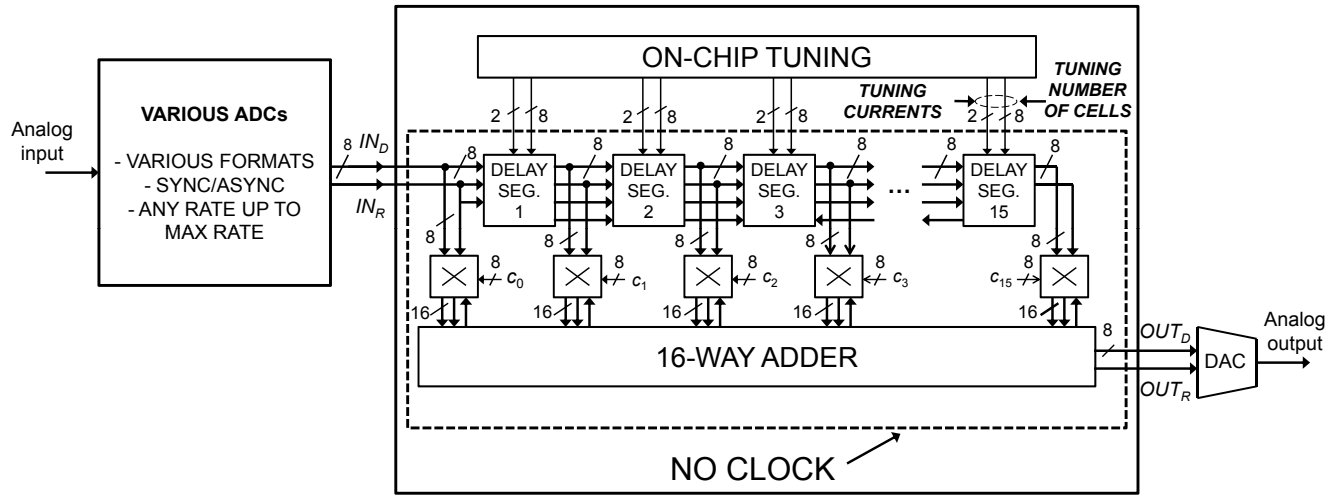

Figure 5.2: Top-level view of the designed CT digital FIR filter as part of a ADC/DAC/DSP system.

output. The chip implements a 16-tap digital FIR filter in continuous time. Four key components are used for the datapath: (i) a delay line with 15 segments, (ii) multipliers, (iii) a 16-way adder, and (iv) on-chip tuning, which calibrates the delay line and sets the filter's frequence response. The design approach is scalable, allowing easy modifications for different specifications.

Samples enter the chip asynchronously. The input channel has 8 data wires (IN_D) and a "data-ready" signal (IN_R) which signifies valid samples. No clock recovery is needed, as there is no possibility of "losing lock" to the data. The 15 delay segments create 15 time-shifted copies of the input. All copies, as well as the input itself, are first weighted using asynchronous multipliers and then summed by the multi-way adder. The output channel of the adder, and also the chip, has 16 data bits, only 8 of which (OUT_D) are sent to the asynchronous DAC, along with the output "data-ready" signal (OUT_R), which uses a pulse to indicate new and valid output samples.

For scalability and low-energy operation, each delay segment consists of two parts, for timing management and data storage, as shown in Fig. 5.3. Sample timing is handled by the timing part, which is a pipeline of delay cells as shown at the bottom of the figure. Sample data is en-queued

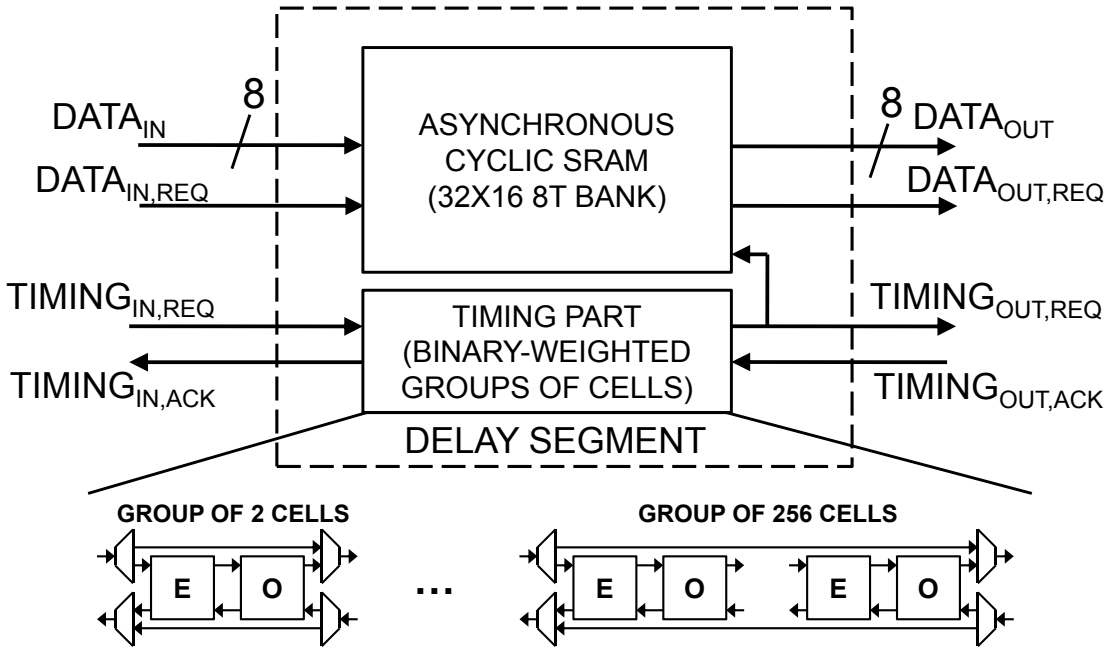

Figure 5.3: Top-level view of a delay segment: decomposition of the segment to data part (top) and timing part (bottom), organization of timing part to delay cell groups with binary-weighted numbers, and two different types of delay cells, "even"(E) and "odd" (O).

(i.e. temporarily stored) in a cyclic SRAM buffer while the timing signal moves inside each segment's timing part, then it is de-queued (i.e. removed) to advance to the next segment when the timing signal exits. Hence, sample data *only move once* within each delay segment, and do not travel along with the timing information from one delay cell to the other as in [33]. Apart from the significant energy saving afforded by eliminating unnecessary data movements, the separation of the segment into the timing path and buffer is a very scalable approach. Different timing specifications/bit-widths would *only* require altering the delay line/SRAM buffers, respectively.

The delay segments require two types of tuning. The first type, global to all segments, adjusts the average cell delay throughout the chip. The second type adjusts the delay of a whole segment, by digitally choosing the exact number of cells within the segment. Each segment can thus be tuned for a very different delay, if desired.

5.4 Delay line: implementing the timing path

The delay line processes sample timing information. It consists of a series of delay segments. Each delay segment has its own timing path, consisting of 510 delay cells, each providing a small delay, ordered into groups with binary-weighted numbers of delay cells, which can be selected on or off using MUXes and DEMUXes. The bottom part of Fig. 5.3 shows as an example the smallest group (of 2 delay cells) and the largest group (of 256 cells). Within each segment's timing part, the delay of each cell, called the segment's *granularity*, must be less than the minimum anticipated sample spacing; this spacing determines the number of cells per segment. The cells can be tuned to provide various delays, as well as operate in different modes which trade off power, inter-sample distance and jitter.

There are two types of delay cells, *even-* and *odd-* numbered, labeled "E" and "O" in Fig. 5.3. The grouping of delay cells to even and odd serves as an extra programmability setting for the timing part. All even-odd pairs inside the chip can be statically programmed to operate in "half-granularity" mode, where the group uses only one of the two cells, i.e. the odd one, to generate the same total delay of the two cells in their regular mode. The entire chip's timing part, for all segments, can be set to half-granularity mode, where it dissipates almost half the dynamic power at the cost of handling samples up to half the maximum rate. This mode can be set with a single digital control bit and without having to use the automated tuning to re-program the delay segments.

The next subsections present the delay cells in detail. Subsection 5.4.1 presents the baseline delay cell, which is the design we used to build the two types of delay cells. Subsection 5.4.2 shows the even and odd cells in detail, and subsection 5.4.3 concludes by presenting the various

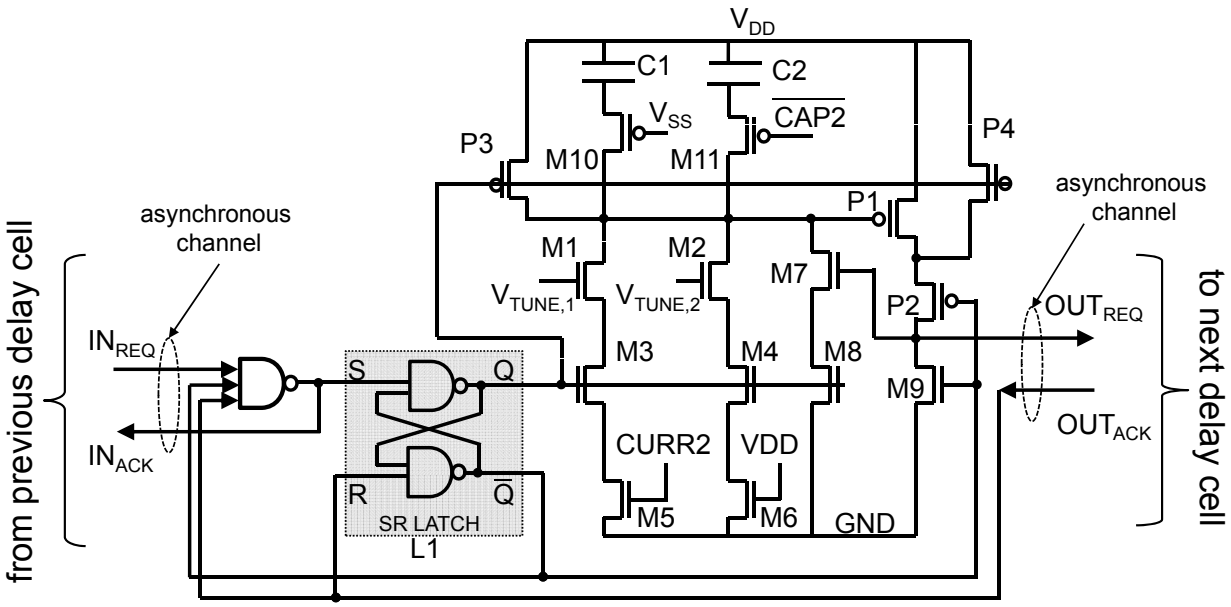

Figure 5.4: Schematic of a delay cell.

operation modes of the two types of cells.

5.4.1 Baseline delay cell

The delay cells are variations of the baseline cell shown in Fig. 5.4, which is an enhanced version of the cell in [33] and will be used to explain the cell operation. Unlike [33], this cell only delays timing information and not data.

The baseline cell has asynchronous, request-acknowledge channels to communicate with its predecessor and successor cell, and provides a fixed time delay via slow-speed charging of a capacitor (C1-2) through a tuned current source (M1-2). The cell operates as follows. First, a sample arrives from the left cell as an active-high request on IN_{REQ}. It then accepts the sample when latch L1 is set, and immediately acknowledges by sending back an active-low pulse on IN_{ACK}, causing IN_{REQ} to be de-asserted by the previous cell. At the same time, the digitally-selected number of current sources (one or two, as selected by $CURR2$) start charging the selected capacitors (one

or two, as selected by the negative-logic signal $\overline{CAP2}$), also gradually lowering the potential at the gate of P1 toward ground. When this potential is lower than V_{DD} by the threshold voltage of P1, the positive-feedback circuit (P1-2, M7-9) quickly completes the charging operation and the sample is passed to the next cell using an active-high OUT_{REQ}. Once the next cell acknowledges with an active-low pulse on OUT_{ACK}, L1 is reset, the capacitor is quickly discharged through P3 and the cell is ready for the next sample. Device P4 also ensures that the drain of P1 is quickly reset and kept at V_{DD}, since in the absence of P4, the node would float. The delay of the cell is controlled from the gate voltages $V_{TUNE,1/2}$ of devices M1/2. The energy consumption of the cell is *independent of its delay*, and depends only on the consumption of the hand-shaking hardware, i.e. L1 and gates, as well as the values of C1-2.

At the input channel, as shown in Fig. 5.4, the cell uses a 3-input NAND gate, while a 2-input NAND was used in [33]. The use of the extra NAND-input provides a more stable handshaking protocol: it does not allow the cell to process a new sample until the acknowledgement from the next cell has been fully released. While $OUT_{ACK} = 0$, i.e. while the successor cell is still acknowledging, the NAND gate blocks the S-input of L1. As a consequence: (a) this gives the delay cell more time to discharge C1-2, making the cell delay less dependent on sample spacing, and (b) this avoids the possibility of L1 receiving concurrent set and reset inputs. This last unsafe case can occur in [33] if the delay through L1 and the NAND gate is small enough; if that happens, then shortly after L1 is reset (caused by $OUT_{ACK} = 0$) the NAND gate output can be pulled low by a new input sample, before the "reset" input of L1 has been released ($OUT_{ACK} = 1$).Such a case of concurrent set and reset of a latch does not cause a problem in the actual design in [33], since the reset always precedes the set operation, but should be avoided in safe asynchronous systems.

The delay cell is initialized using an active-low global $RESET$ signal, as shown in Fig. 5.5.

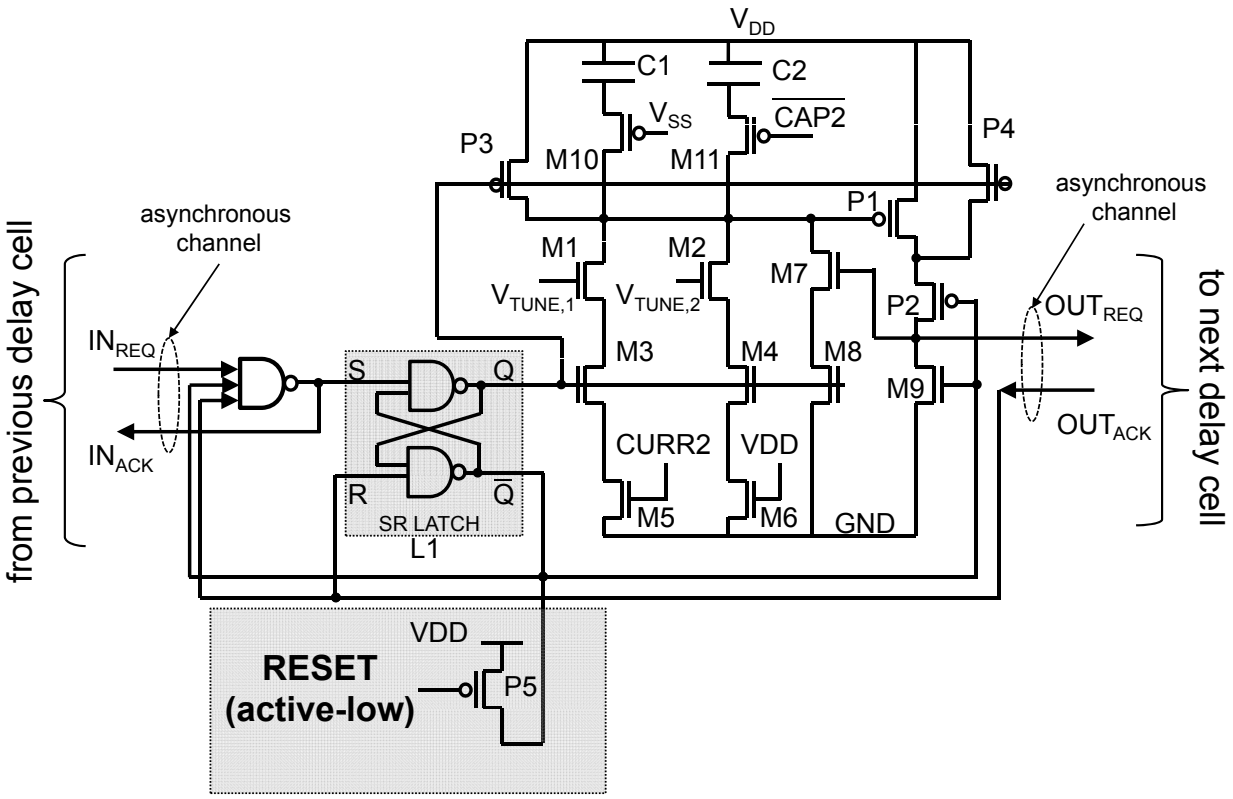

Figure 5.5: Schematic of a delay cell indicating the reset method.

The entire cell is reset using one transistor (P5), pulling the complementary output of L1 to V_{DD} and effectively resetting the latch. This also enables the discharge of capacitors C1-2, through device P3. The sizing and V_T type of all the devices in the cell are shown in Table 5.1. C1-2 were implemented as p-type MOS-caps.

5.4.2 Even and odd delay cells

Detailed designs for the even and odd cells of Fig. 5.3 are shown in Fig. 5.6; each is an enhanced version of the baseline cell. Both receive an extra digital control *GRAIN*, which is used to configure them into one of two digital modes.

The even cell, shown in the left part of Fig. 5.6, has extra pass-transistor MUXes and DE-

Table 5.1: Transistor sizing in baseline delay cell.

Device	**Width / Length**	V_T **type**
M1	0.3 μm / 2 μm	regular
M2	0.3 μm / 2 μm	regular
M3	0.16 μm / 0.12 μm	high
M4	0.16 μm / 0.12 μm	high
M5	0.16 μm / 0.12 μm	high
M6	0.16 μm / 0.12 μm	high
M7	0.16 μm / 0.12 μm	regular
M8	0.16 μm / 0.12 μm	high
M9	0.3 μm / 0.12 μm	high
M10	0.16 μm / 0.12 μm	high
M11	0.3 μm / 0.12 μm	high
P1	0.2 μm / 0.4 μm	high
P2	0.3 μm / 0.12 μm	regular
P3	0.9 μm / 0.12 μm	regular
P4	0.3 μm / 0.12 μm	regular
P5	0.9 μm / 0.12 μm	regular
C1	0.6 μm / 0.6 μm	regular
C2	0.6 μm / 0.6 μm	regular

MUXes surrounding it The arrows in the left part of Fig. 5.6 indicate the MUX and DEMUX paths for the two cases of the control signal *GRAIN*: when $GRAIN = 1$ the cell is bypassed with a small energy drain, by directly connecting the input and output channels, via the MUXes and DEMUXes.

The odd cell has an extra NAND gate controlling one of its two current sources, namely M5, as $\overline{(CURR2 \cdot GRAIN)}$. When $CURR2 = 1$, the odd cell can easily double its delay when $GRAIN = 1$, by switching to one current source instead of two, for twice the delay. All cells are organized into pairs of one even- and one odd-numbered cell. Each pair can operate in two distinct modes, as controlled by *GRAIN*. When $GRAIN = 0$ both cells provide the same delay. When $GRAIN = 1$,

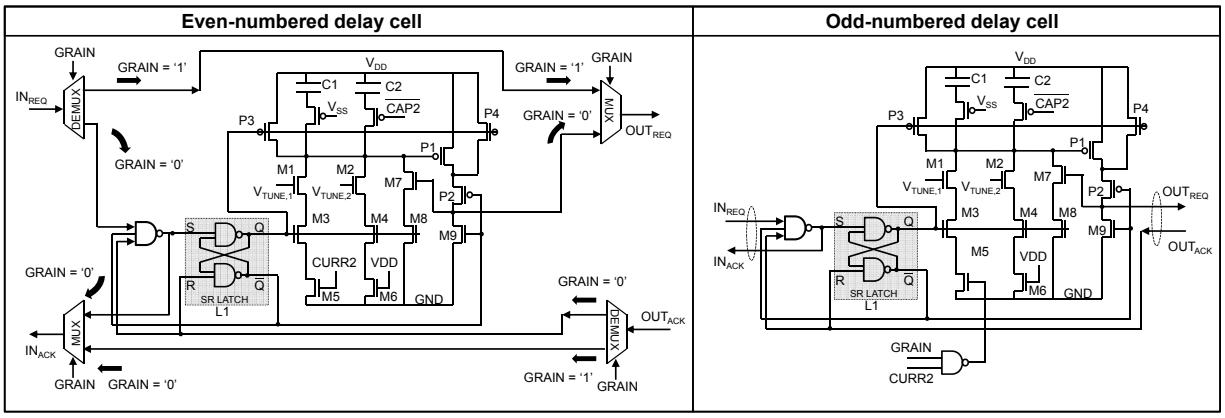

Figure 5.6: Schematic of the even- and odd- numbered delay cells.

the even cell is bypassed and the odd has twice the delay: Each group uses only one cell and dissipates half the energy, but provides the same overall delay.

Note that the setting of $V_{TUNE,1} = V_{TUNE,2}$ does *not* guarantee the precise doubling in delay of a cell, once the latter switches from two to one current sources. The capacitance at the charging node (i.e. the gate of P1) includes, besides C1 and C2 parasitic capacitances from both P1 and M1-2. When changing the number of current sources, the resulting capacitance changes as well (but does not halve), so using exactly half the current does not double the delay. By properly using the automatic tuning to set $V_{TUNE1,2}$, a precise delay doubling is guaranteed.

5.4.3 Operation modes and associated trade offs

The design of the new delay cells supports various digital modes, using the control bits, thereby trading off power, jitter and inter-sample spacing. The normal mode involves one capacitor ($\overline{CAP2} = 1$), 1 current source ($CURR2 = 0$) and also $GRAIN = 0$. Cell delay can be increased/ decreased without using the automated tuning by selecting two capacitors ($\overline{CAP2} = 0$) / two current sources ($CURR2 = 1$), respectively. In the "low-jitter" mode, the same delay as in the normal mode is

produced by using two capacitors and two current sources ($\overline{CAP2} = 0$, $CURR2 = 1$). By doubling the energy to generate the same delay, the jitter power is reduced by 2, as in the case with ring oscillators [114]. The total power in this case is only 10% higher, since the energy to charge C1-2 contributes little to the cell's total energy consumption. The latter is dominated by latch L1 and the handshaking.

Finally, control *GRAIN* can be used to alternate between two granularity settings, while maintaining each segment's delay. When using two current sources (i.e. $CURR2 = 1$), the delay segments can either work in full-grain mode ($GRAIN = 0$), where all cells are active, or in half-grain mode ($GRAIN = 1$). In the latter, each even-odd pair of delay cells produces the same delay using only one cell, reducing dynamic power by almost 50%. However, the cost for this mode is the reduced capacity to store samples: this mode can only be used when the sample rate is up to 50% of its maximum value.

The range of delays that can be produced by each cell is 4 *times wider* than [33], ranging from 15 ns to 500 ns, by exploring all the digital cell's settings, as well as the on-chip tuning, presented in Sec. 5.8. Each cell dissipates 50 fJ per delay operation in the "low-jitter" mode. This is approximately 50% higher than the case in [33], due to the use of a larger-feature technology (130 nm vs. 90 nm in [33]). The delay cells achieve reduced leakage by a factor of 6 compared to the design in [33], due to both the used technology and the proper use of high-V_T devices (M5-6, M8-9). The cell delays vary by $\pm 10\%$ for a $\pm 20\%$ variation of the supply (nominal 1 V) when automatic tuning is de-activated.

5.5 Delay segment: SRAM memory for local data storage

An SRAM memory is provided with each delay segment, to store sample data while the timing signals propagate through the corresponding segments. As shown in Fig. 5.3, the timing and data channels in each segment are kept separate; however, their operation is carefully coordinated. Shortly after a sample's timing signal enters the segment, the data follows and gets *en-queued* (i.e. temporarily stored) in the SRAM. The data bits are first set and then a pulse at $DATA_{IN,REQ}$ enables their storage. When the timing signal exits the segment, appearing at $TIMING_{OUT,REQ}$, it causes a *de-queue* operation in the SRAM. The data bits are first de-queued and then a pulse at $DATA_{OUT,REQ}$ indicates that the output bits are stable, i.e. that a new sample is available. The sample is then sent to the memory of the next delay segment, as well as to the corresponding FIR multiplier. The SRAM of the first segment uses the input "data-ready" signal IN_R in 5.2 to en-queue data.

Each memory is implemented as a dual-ported asynchronous SRAM, shown in Fig. 5.7. It can read from one entry and concurrently write to another one. It is composed of 512 8-bit entries, with 32 rows by 16 columns, and uses 8-transistor cells, with separate sets of nMOS transistors for read and write.

Fig. 5.7 also shows the asynchronous circuitry performing the read, i.e. de-queue, operation. The alternating phases for the read operation are controlled by self-timed circuits, using inverter-chain matched delays. The OS ("ONE-SHOT") block in Fig. 5.7 is an edge-triggered one-shot. Upon starting the read operation, latch L2 enables the address generator to select the proper word (i.e. row and column) while the sense amplifiers stop equalizing the read bit-lines, on which the selected SRAM cells build differential voltages. After the appropriate delay, L3 isolates the bit

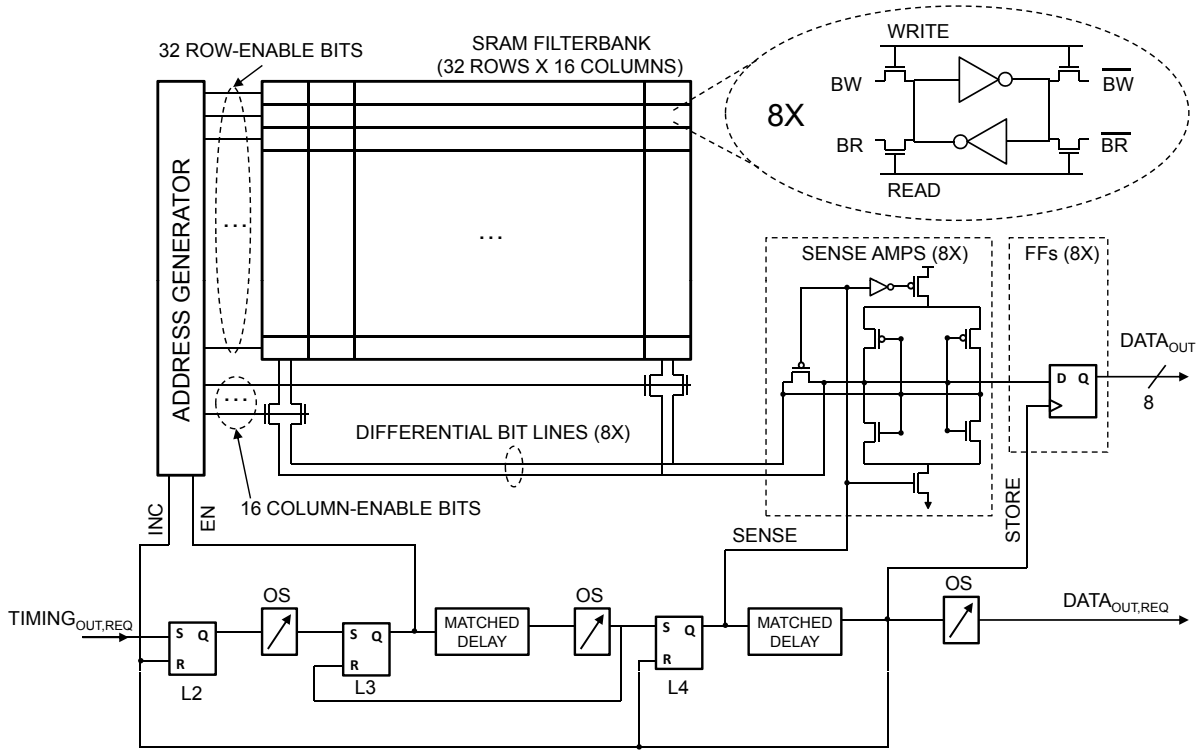

Figure 5.7: Structure of the SRAM memory, as well as the circuitry performing the read (i.e. de-queue) operation.

lines from the SRAM entries, and the sense amplifiers latch to the appropriate digital value, which is then stored into the output flip-flops. Finally, a pulse at $DATA_{OUT,REQ}$ signifies the valid output sample. The write operation is much simpler, and is not shown in Fig. 5.7. The write/ read operation's cycle time is 4/ 12 ns respectively, and the total energy required to handle a single sample is in the order of the consumption of only a few delay cells.

5.6 FIR tap multiplier

Each FIR tap consists of a distinct asynchronous multiplier. Each multiplier interfaces with one delay segment and the multi-way adder. It multiplies each 8-bit samples exiting from the corre-

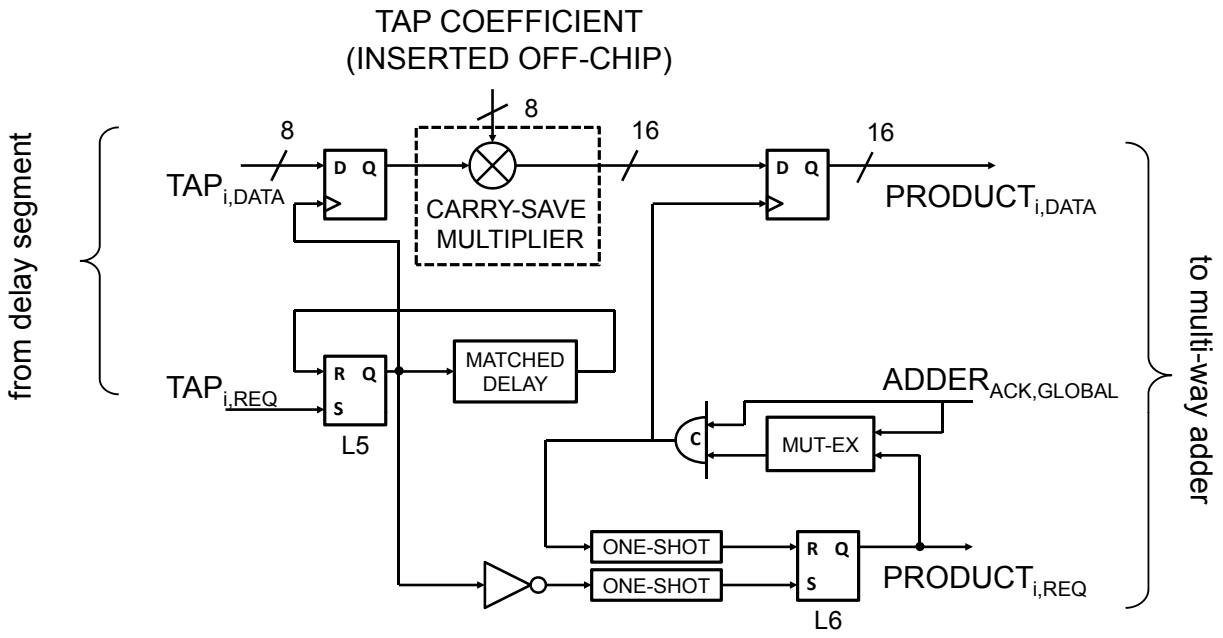

Figure 5.8: Asynchronous FIR tap (multiplier).

sponding delay segment, with a programmable 8-bit coefficient, and the 16-bit output is passed to the adder using four-phase asynchronous handshaking.

A detailed schematic of a multiplier is shown in Fig. 5.8. For a new input sample, the data bits arrive first and a pulse on $TAP_{i,REQ}$ indicates the sample arrival, setting latch L5. The data is stored in the input flip-flops (FFs) and processed by a carry-save combinational array multiplier. After a proper delay, designed to exceed the worst-case path, L5 is reset and the tap attempts to send the new sample to the adder by raising $PRODUCT_{i,REQ}$. The raising of $PRODUCT_{i,REQ}$ is caused by the combination of the inverter and a OS block, which detect multiplication completion and set the output of SR latch, L6.

One or more taps can concurrently push new samples into the multi-way adder, with priority given to the one which "requested" access earliest. When the adder receives its first such tap request, it sends a global acknowledgment $ADDER_{ACK,GLOBAL}$, to all sixteen taps in parallel. Each tap locally arbitrates between its own adder request, $PRODUCT_{i,REQ}$, and $ADDER_{ACK,GLOBAL}$,

using a mutual-exclusion (MUTEX) element. Only if the former signal had been issued before the latter are the output data passed to the adder. This process occurs at each tap, resulting in a protocol where any number of taps can send new samples to the adder. Taps requesting access within this time window, which is shown to be small in the next section, will all push their samples to the adder, merging their samples together as a single multi-way sample. If a tap did not get access, it will hold $PRODUCT_{i,REQ}$ high until the adder is finally free and provides access to the tap, and thus the sample is deferred to the next adder operation. This sample "stalling" causes timing perturbation to some samples, so minimizing the adder cycle time is key.

5.6.1 Improving on an earlier design

The design presented in Fig. 5.8 is slightly modified with respect to the one in [33], correcting a design flaw in the latter which is indicated in Fig. 5.9.

In the previous implementation, the arbitration in each FIR tap is performed using a switch and an AND gate, which "sample" the request inside each tap when the adder acknowledgement arrives. Based on this sampling, which ideally decides if $ADDER_{ACK,GLOBAL}$ arrived before $PRODUCT_{i,REQ}$, data is allowed to pass to the adder or not. Fig. 5.9 shows the part of interest in the approach in [33], which implements the arbitration, and sketches the problem itself in the right part of the figure.

While the concept in both versions is the same, the earlier implementation shown in Fig. 5.9 can lead to erroneous cases like the one shown in the right part of the figure. In particular, if both signals $ADDER_{ACK,GLOBAL}$ and $PRODUCT_{i,REQ}$ arrive within a very small time window, i.e. almost concurrently, then a runt pulse may appear the AND-gate output, not even reaching VDD, instead

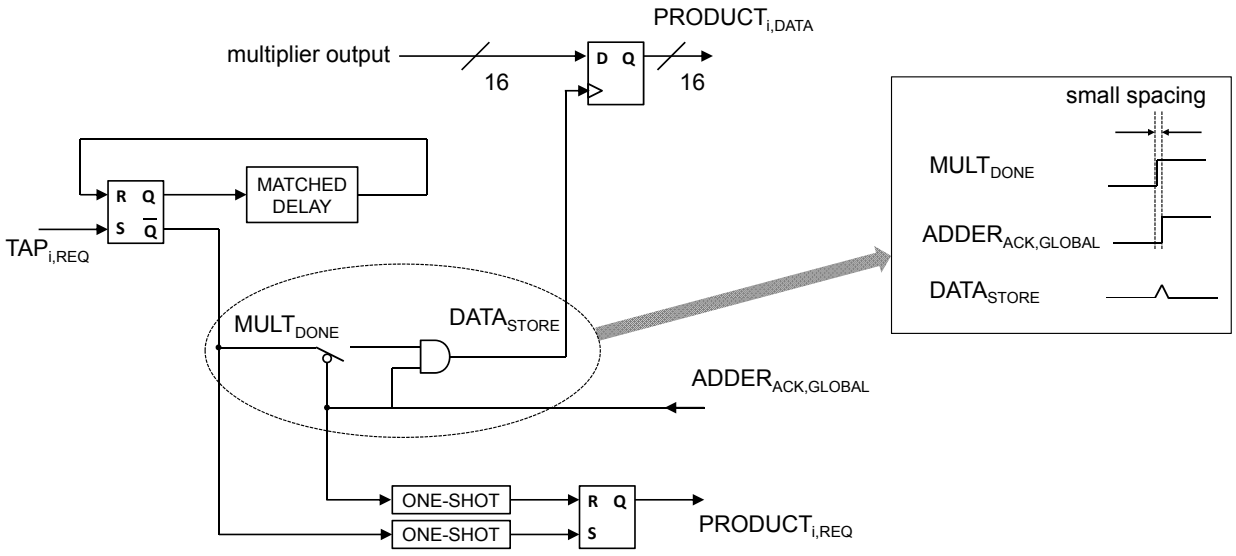

Figure 5.9: Design bug in previous CT DSP multiplier implementation.

of a clean pulse which occurs in the cases where the signals are separated by a wide margin. Such a pulse can appear due to low-level mismatches in either wire delays, where $ADDER_{ACK,GLOBAL}$ is forked to the switch and gate or to the intrinsic delays of those blocks). It can lead to failure to store the multiplier output data for this cycle, leading to output distortion, which is, however, non-persistent since the data at that point has been converted to absolute encoding, so the output can be restored by the next sample. In the extreme case, where $PRODUCT_{i,REQ}$ is sampled when transitioning, one input to the AND gate may simply float to an intermediate value between 0 and V_{DD}, causing the output to also have an intermediate value. This would cause static power consumption at the output FFs until the signal is reset to either 0 or V_{DD}.

The introduction of the proper usage of the MUTEX and C-elements in the new design avoids such cases. Through the MUTEX, the output is guaranteed to transition monotonically; in the case when a pulse is produced to store the data, then the width of the pulse is controlled and can not become arbitrarily small. Finally, the scenario of resetting the multiplier but not clocking the data to the adder is not possible in this design, since the two operations are safely synchronized using the

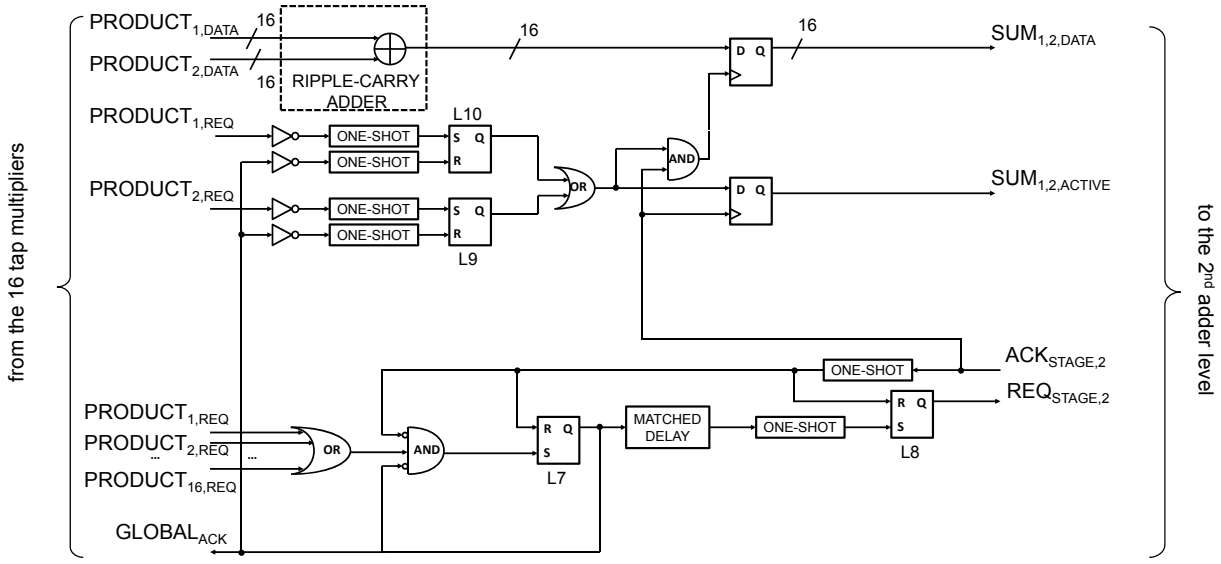

Figure 5.10: Structure of the 1st level of the multi-way adder. The bottom part of the figure shows the asynchronous control for the adder's timing. The bottom part shows one of the eight identical structures performing 2-way tap additions, as well as the asynchronous control which eliminates unnecessary data movement between successive adder levels.

C-element, with data storage being a pre-requisite for resetting the multiplier-adder hand-shaking (latch L6).

5.7 Multi-way adder

Finally, a single multi-way adder combines all 16 taps for the final FIR output. It is structured as a tree of two-way adders, organized as a pipeline with four levels. Each level uses four-phase handshaking to communicate with the previous and next level, and two-way ripple-carry adders for the datapath. Due to the use of 2's complement representation throughout the FIR, overflow in intermediate stages does not cause error at the final output.

Fig. 5.10 shows a detailed design of the first adder level. For simplicity, only one of the eight data paths, i.e. two-way adders, is shown. The bottom part of the figure shows how the first level

communicates with the taps (left) and the second level (right). When the adder is free and receives the first active-high request ($PRODUCT_{i,REQ}$), it immediately acknowledges all taps. Some send new samples to the adder, to start the 2-way additions. After a matched delay, equal to the delay of the worst-case addition, the first stage sends an active-high request, $REQ_{STAGE,2}$, to the second stage. Upon receiving an acknowledgment from the second stage ($ACK_{STAGE,2}$), indicating that the latter is free, the first stage clocks[1] its two-way addition outputs. The cycle time of the OR, AND gate and L7, i.e. the latency between an active-high request and the broadcast of an active-high acknowledgement to all taps, sets the length of the time window during which samples from multiple taps can arrive and still gain access to the adder.

The top part of Fig. 5.10 also shows how the adder handles data movement. A typical addition may involve new samples from only a few taps, or even from one tap. Clocking all of the 2-way adders' outputs while moving to the next adder stage is typically unnecessary. In this design, the adder monitors which taps are "active", i.e. sent new samples: active taps will de-assert their requests, which through OS blocks will set latches such as L9-10, which are used to watch if taps 1-2 are active. Only for active taps will the corresponding 2-way adders have new outputs, and their output flip-flops be clocked. In this way, much unnecessary clocking is eliminated. Information about active taps is forwarded to the next adder level, where it is used in a similar fashion. The 16-bit final adder (and FIR) output is sent off-chip in parallel format, along with the timing signal OUT_{REQ}.

The pipelined structure combines small cycle time with energy efficiency. As explained in chapter 4, a small adder cycle time is needed to minimize congestion. Pipelining reduces cycle time without the use of energy-hungry adder schemes. The current design has a 8 ns cycle time.

[1] By "clock" in this case, we refer to an asynchronous signal that enables flip-flops, not to a synchronous clock.

As has been shown in the previous chapter, this cycle time is enough to limit adder congestion to a certain extent. The 8 ns cycle time is twice that of the design in [33], despite both being implemented in a slower technology, i.e. 130 nm for this design vs. 90 nm in [33], and using much slower adders, i.e. RCA vs. carry-look-ahead in [33]. Use of a smaller cycle time in this approach was achievable, but at the cost of increased energy consumption. However, since an off-chip commercial DAC was used, with speed limited to about 40 Msamples/s, using a smaller cycle time would have little effect on the signal quality, since the DAC could not be able to make use of the increased data rate at the output of the chip.

5.8 On-chip automatic tuning

For proper functionality of the FIR, two conditions must be imposed on the delay segments. First, the granularity, i.e. the individual cell delay, in each segment must be fine enough to handle minimum-spaced samples. Second, each segment must have the proper total delay. These conditions are ensured by analog and digital tuning.

Two tuning blocks are used, global and local. The former tunes the delay of all cells inside the chip, while the latter tunes the delay of each delay segment. Both blocks first compare the delay of a part of a line with a programmable synchronous delay, and then tune based on the outcome. Tuning is the only place where a synchronous clock is used; the clock can be turned off during normal FIR operation. This is the first on-chip tuning system proposed for CT DSPs.

Fig. 5.11 shows the global tuning hardware, used to adjust the delay of all cells in the delay line. Being of master-slave-type, it compares a replica group of 128 delay cells, not used in the FIR filter to a programmable clocked delay. The delay of the replica group is adjusted via a 6-bit

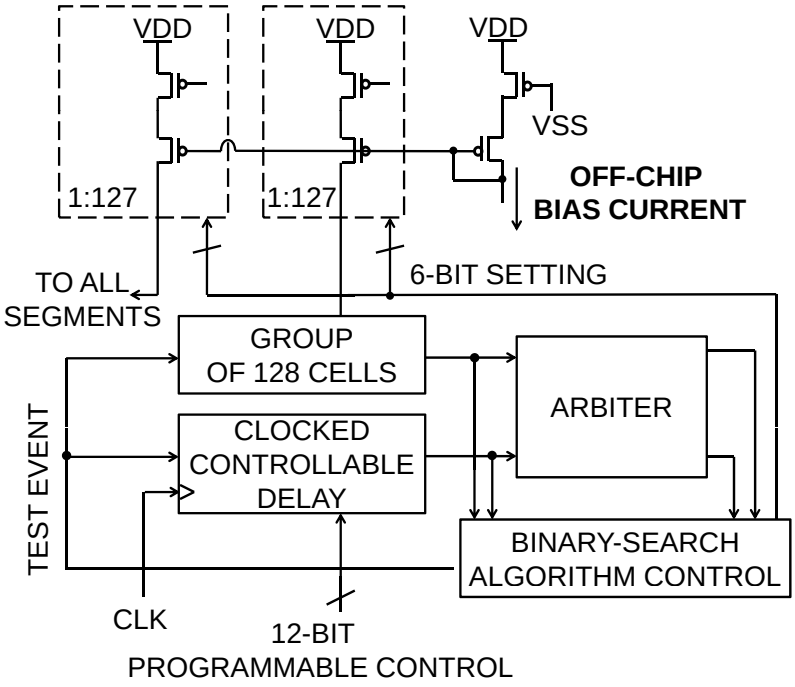

Figure 5.11: Automated tuning: global tuning, adjusting the average delay of all delay cells in the delay line.

charging current, which is fed to the delay cells using current mirrors, adjusting the gate voltages of devices M1-2 in Fig. 5.4. The binary-search algorithm, hardwired in asynchronous logic, is executed to find the best 6-bit current setting, so that the cell delay matches the programmable delay. At each step, a single event is sent to both the synchronous delay and the replica delay group. A mutual exclusion (MUTEX) element finds the fastest path, and the charging current is incremented/ decremented if the replica group is slower/ faster. The final current is then sent to all delay segments in the chip using current mirrors, setting $V_{TUNE1,2}$ in Fig. 5.4 for all delay cells. Separate tuning is needed for each of $V_{TUNE1,2}$.

Fig. 5.12 shows the local tuning, which adjusts each segment's total delay individually. Such tuning adjusts the total number of cells in a segment by choosing which binary-weighted groups of cells are turned on, usually after global tuning. At local tuning startup, the chosen segment

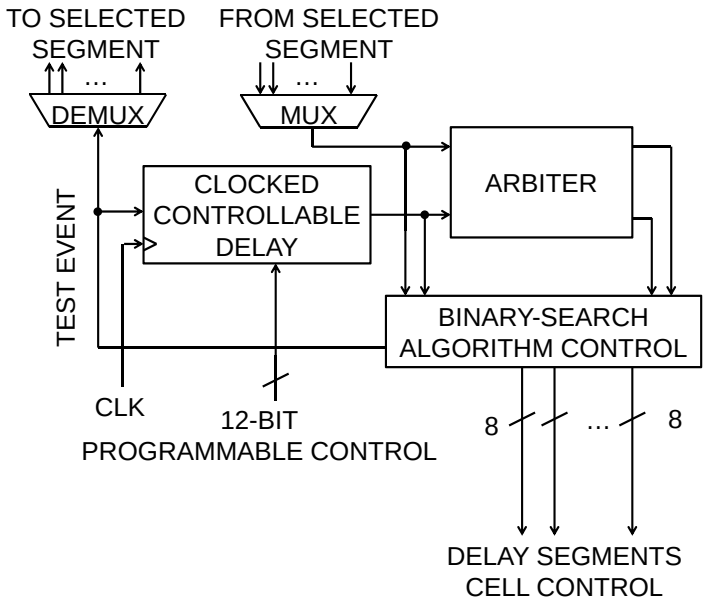

Figure 5.12: Automated tuning: local tuning, adjusting the overall delay of one selected delay segment.

is initialized to only use the largest group of 256 cells. In each step, a single event is sent both to the segment and the programmable synchronous delay. A MUT-EX element finds the fastest path, by deciding which output sample arrived first, and then asynchronous logic switches the next largest cell group on/off if the segment's delay is smaller/larger. Using a 7-bit delay cell setting, the segment's delay can be tuned with 1% error. The FIR must be inactive for both, either to equalize the segment delays or to create FIR filters with unequal delays.

5.9 Tuning interface and FIR-length programming

Fig. 5.13 shows the interface between the delay segments, or more accurately their timing paths, which are also used to interface to the local tuning. These interface blocks are named "connectors", since they connect consecutive segments, and are omitted from Fig. 5.2 for simplicity. Using the

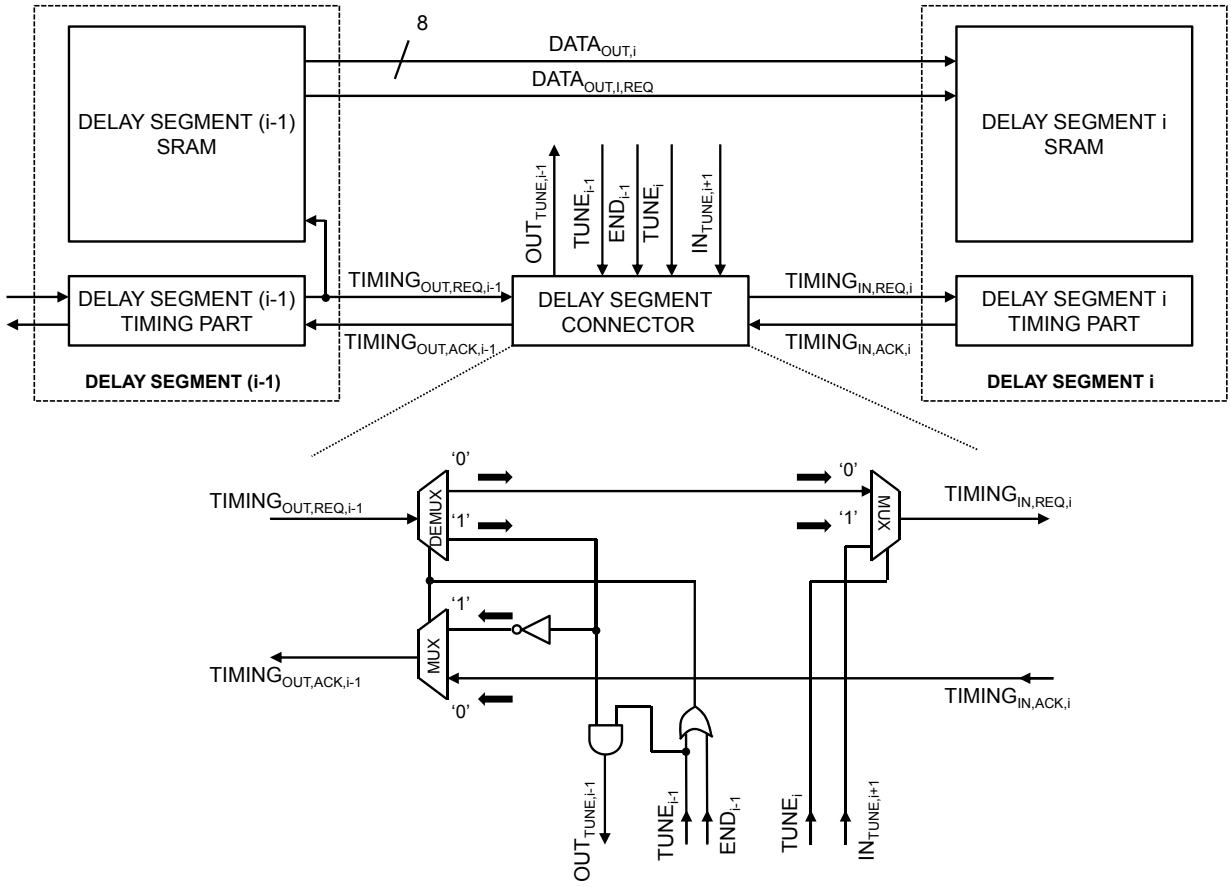

Figure 5.13: Interface between two delay segments' timing paths, enabling easy access to the delay segments for local tuning.

digital control bit $TUNE_i$, a particular delay segment can be isolated for local tuning: by setting $TUNE_i = 1$, the input and output of the i-th segment's timing path switch to interface with the local tuning, instead of their normal interface with the $i-1$-th and $i+1$-th segments. In this case, the i-th segment is isolated from its neighbor segments; it receives a new timing event from the local tuner, and the output of the i-th segment is sent to the local tuning, to be used for comparison to the programmable delay. Note that each delay segment is isolated using two delay segment connectors, the one preceding the segment and the one following it.

The 15 digital bits END_i ($i = 1, ..., 15$) form a one-hot code that determines the length of the FIR filter. Using $END_i = 1$ makes the delay line end at the i-th delay segment, programming the

FIR filter to use only the first i delay segments, hence $i+1$ taps.

5.10 Design challenges

Following the presentation of the chip's overall design, the largest challenges posed throughout the design phase are now considered. The current chip design also has a number of potential disadvantages, compared to both previous CT DSPs as well as synchronous ones.

First, its area is considerably larger than previous CT DSPs. As any CT DSP, this chip too required custom layout and considerable manual routing; this is a general drawback of all CT DSPs compared to synchronous systems, which can be easily synthesized and implemented using synchronous design tools. Automatic tuning also adds significant complexity to a CT DSP, much more to this design in which tuning was implemented on-chip and for wider programmability. The result of the tuning was the requirement to route a very large number of signals, most of them across-chip. The usage of 8-bit samples, as opposed to 1-bit samples coming from Δ modulation in [33], also increased the amount of signal routing significantly.

The usage of per-segment SRAMs also came with a certain number of design challenges. The requirement for truly dual-ported operation, i.e. concurrent read and write operations, imposed the usage of 8-transistor SRAM cells, as shown in Fig. 5.7. This resulted in an increase to the SRAM's area compared to the case in synchronous designs. Furthermore, the asynchronous nature of the SRAM requires custom design and requires precise control of all individual timing signals, to which the SRAM design is fairly sensitive.

Finally, contrary to the design in [33], the output data was not restricted to changing by only 1 LSB at a time; this posed additional requirements for the DAC used on the test board, as discussed

in the next chapter.

5.11 Summary

In this chapter we have presented the full design for a novel multi-rate and scalable CT DSP core, which was implemented in a 130 nm technology. This design can process any digital encoding of 8 bits or less with the same frequency response, regardless of the sample and format of the input. As will be shown in the next chapter, we used this chip to construct a complete ADC/DSP/DAC system which maintains its response while using different ADCs, synchronous or not and under widely-varying input rates.

The design decisions were highly motivated by scalability and modularity. As a result, targeted design approaches were introduced, such as data-and-timing decoupling, pipelined adder structures and grouping of delay cells into binary-weighted groups for easy tuning. This chip is also the first CT DSP which includes automatic tuning.

Chapter 6

Measurement Results for the CT DSP Chip Prototype

This chapter presents the results of measurement on the designed CT DSP chip. The chip successfully processes multiple types of digital signals, exactly as expected. Due to the inherent property of CT DSPs of having a response independent of the input data rate, the behavior of this chip was held constant even when processing a wide variety of signals.

The chapter first introduces details on the implementation of the chip in Sec. 6.1, and on the test board setup in Sec. 6.2. Measurement results follow: first on frequency-response behavior of the system in Sec. 6.3, then on the chip's power consumption in Section 6.4. The effect of the automatic tuning is shown in Sec. 6.5. Additional measurements are shown in Sec. 6.6. A detailed performance table of the measured system is shown in Sec. 6.7, followed by a comparison to state-of-the-art DSP systems in Sec. 6.8. Section 6.9 concludes this chapter.

6.1 Implementation details

The 8-bit, 16-tap FIR chip was implemented in IBM 0.13 μm CMOS technology with 8 metal layers. Fig. 6.1 shows the die photo. The total chip area, including pads, is 9 mm^2, and the active area is 5.6 mm^2. The design, placement and routing was all custom, without any aid of place-and-route tools, due to the asynchronous nature of the chip's operation. This is one of the drawbacks of CT DSPs, at least at the present time, given the lack of CAD tool support for such mixed-signal and real-time circuits. The design involved a significant amount of routing, due to the large amount of programming and tuning incorporated to the chip. Automatic tuning was a big factor in this increase in routing; the requirement for tuning is one more challenge in designing CT DSPs, when compared to synchronous solutions, which do not require such tuning.

The active area of the chip, 5.6 mm^2, is largely dominated by the delay line, due to the latter's fine granularity and programmability. The significant increase in chip area compared to the design in [33] is due both to the technology used (130 nm vs. 90 nm in the design in [33]) and, especially, to the wider programmability of the delay line in this design. The latter has extra operating mode, due to the use of two charging capacitors and current sources in each delay cell, as well as due to the SRAM memories which can handle 8-bit samples. The chip prototype is fully functional with supply voltages ranging between 0.6 and 1.8 V, but all measurements shown are for a 1-V supply. All measurements were performed at room temperature.

6.2 Test setup

The test board for the chip implemented an ADC/CT-DSP/DAC system, where various ADCs can be used, is shown in Fig. 6.2. The board was designed and assembled by Yu Chen. The options

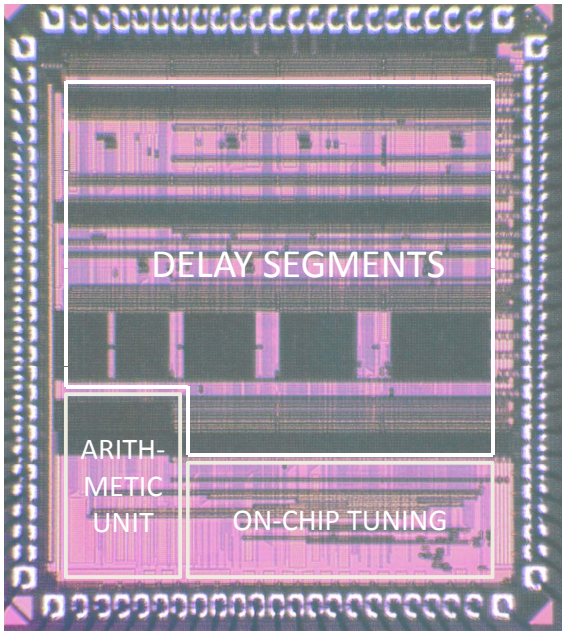

Figure 6.1: CT digital FIR die photograph. The total chip size (including pads) is 9 mm^2.

for the ADC include an 8-bit synchronous PCM and 1-bit PWM and ΣΔ, both synchronous and asynchronous. PCM and synchronous PWM and ΣΔ modulations were created using commercial ICs, while asynchronous PWM and ΣΔ modulators were implemented using discrete components. The asynchronous PWM modulator includes a single 1-bit comparator, comparing a saw-tooth waveform with the analog input; asynchronous ΣΔ was implemented using an asynchronous 1-bit quantizer, i.e. a comparator, and a single operational amplifier. Single-bit modulations can either be padded to extend to 8 bits or be fed as a single-bit stream, with all other bits hardwired to either a logic 1 or 0. Different formats can be chosen very simply through a set of switches, and while operating with each format the sample rate is allowed to change without affecting the system's response, as verified experimentally.

The "data-ready" signal, IN_R, used to indicate the arrival of new input samples to the chip, was generated either using the sample clock, in the case for PCM, or through a 1-bit change detector in

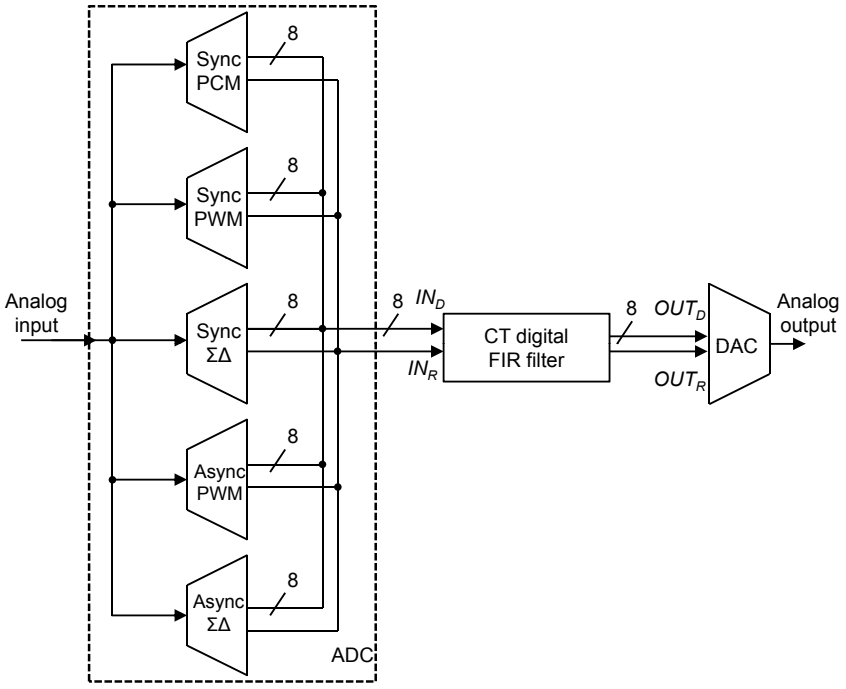

Figure 6.2: ADC/CT-DSP/DAC system with various ADCs, used for chip measurements.

the remaining cases, as shown in Fig. 6.3. The "change detector" block outputs a short pulse for both the positive and negative changes of the single-bit modulations.

The DAC used on the board was a high-speed commercial DAC (AD9742). This DAC is nominally intended for synchronous use, however it was able to successfully function asynchronously while being triggered by the "output data-ready" signal provided by the chip. The DAC was fed with the 8 MSBs of the chip's output. Contrary to synchronous systems, where the DAC speed is equal to that of the ADC, in this case the speed requirements for the DAC were much more stringent. In principle, the data rate at the chip's output is 16 times that of the input, given that the 16 FIR taps are not synchronized, so each input sample causes 16 output samples. Furthermore, in the case of congestion to the adder, samples are outputted with a minimum spacing of 8 ns, as explained in the previous chapter; the DAC should be able to handle such samples. In this setup, the fastest found commercial DAC with parallel data format was used, able to operate up to 40

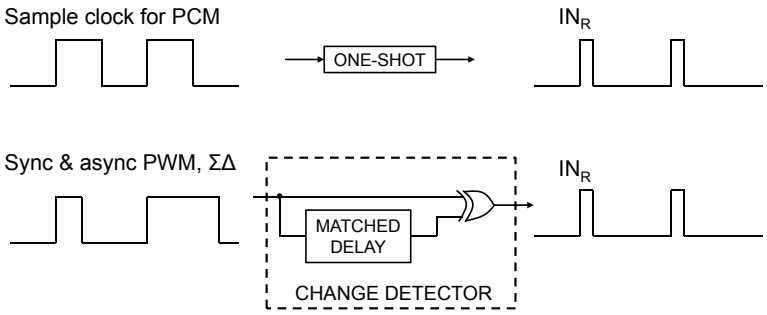

Figure 6.3: Method of generating the data-ready signal in the CT DSP test board.

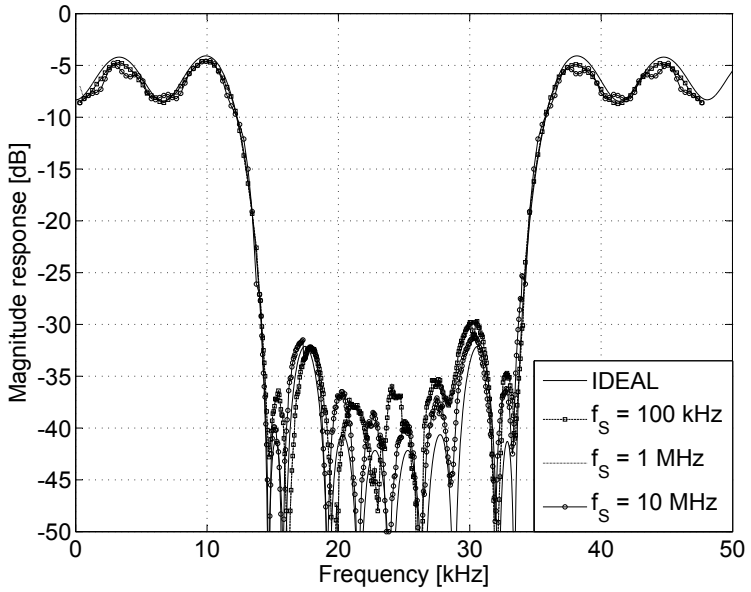

Figure 6.4: Frequency responses demonstrating independence from input sample rate. PCM input, FIR low-pass response following automatic tuning.

Msamples/s. The DAC speed requirement is currently a drawback in CT DSP systems.

6.3 Frequency response measurements

The independence of the system's behavior, i.e. frequency response, from the sample rate was verified for all input formats. Fig. 6.4 shows three measurements of the system's frequency response, measured with PCM input and different sample rates. The experiment involved first programming

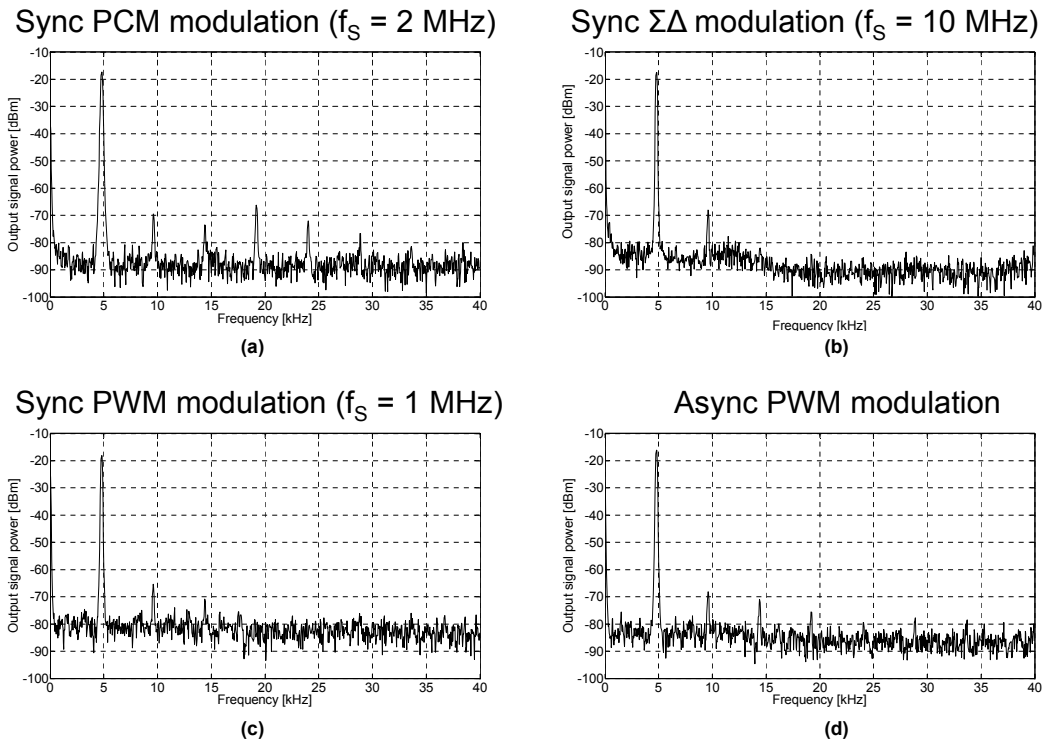

Figure 6.5: Output spectrum for the system of Fig. 6.4 driven with 4.8 kHz full-scale sinusoidal input. Four different ADCs were used; no internal adjustments to the CT DSP or system pauses were made when switching ADCs. A resolution bandwidth (RBW) of 300 Hz was used in the spectrum analyzer.

and tuning the CT DSP chip, then measuring the system's response without pausing normal operation or doing any adjustments to the chip. While the sample rate changed, the response of the system was not disturbed. This was verified in both the frequency domain, as shown in Fig. 6.4, as well as the time domain. This property is impossible for synchronous ADC/DSP/DAC systems.

Fig. 6.5 shows the spectrum of the system's output for a 4.8 kHz analog input processed with four different ADCs, each of different sample rate. As in the previous experiment, no pauses or adjustments were made to the DSP chip when switching between ADCs, or when changing the sample rate. The equal amplitude in all cases is consistent with the observation that the system's response remained intact throughout this experiment. With a classical DSP, neither the frequency

response nor the time-domain experiment described previously would be possible, unless the DSP is re-programmed and/or its clock changed each time the sample rate changed.

6.4 Power measurements

The previous section demonstrated the unique property of this chip to maintain its frequency response for different sample rates. In this section, actual measurements on the chip's power consumption are presented.

The power consumption of the chip is, as is the one of any CT DSP, input-dependent. It can widely vary across the input sample rates. The chip's leakage power is significantly reduced with respect to the previous voice-band CT DSP. This results in this chip consuming less total power, i.e. static + dynamic for small-rate inputs.

The filter's power consumption is input-dependent, as expected of CT DSPs. The expected power consumption is as follows:

$$\begin{aligned} & P = SR \cdot E_{sample} + P_{bias} \\ & \Rightarrow P = SR \cdot \left[\left(N_{cells/segment} \cdot E_{delay,cell} + E_{SRAM} \right) \cdot N_{segments} + E_{arithmetic} \right] + P_{bias} \ , \end{aligned} \tag{6.1}$$

where SR is the sample rate, $N_{cells/segment}$ is the total number of cells used in each delay segment, $N_{segments}$ is the number of FIR taps and $E_{delay,cell}$, E_{SRAM} and $E_{arithmetic}$ are the energy dissipations of one delay cell, one SRAM memory (for both read and write) and the combination of one multiplier and the adder for a single sample respectively. In this design, $E_{delay,cell} = 50$ fJ, $E_{SRAM} = 500$ fJ, $N_{segments} = 15$; the number of cells in each segment, $N_{cells/segment}$, varies depending on the desired frequency response, and the energy of the arithmetic blocks varies according to the sample

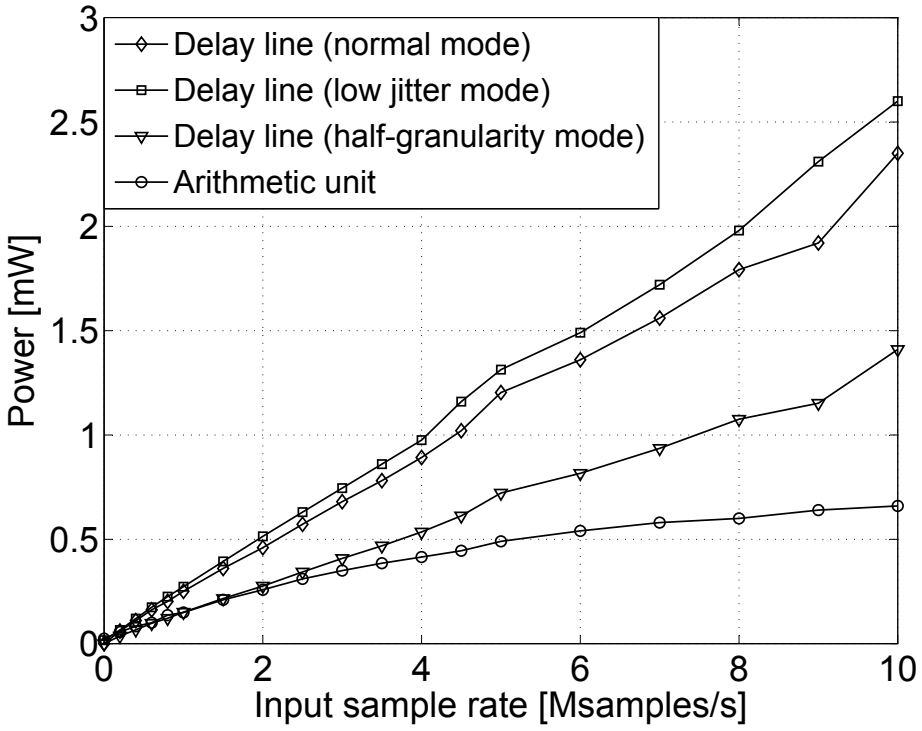

Figure 6.6: Power dissipation of the CT digital FIR key components vs. input sampling rate for a PCM 8 kHz input. The power of the delay line is presented for 3 different operation modes.

rate, as will be explained shortly.

Fig. 6.6 shows the breakdown of the filter's power to its key components, plotted versus the input sample rate. For this experiment, the delay line was initially tuned for a granularity fine enough to handle input rates up to 20 Msamples/s, hence a cell of 50 ns in each delay cell. This setting is enough to support the worst-case, i.e. maximum, input sample rate. Furthermore, the overall frequency response was set in the KHz region. This resulted in an average cell delay of 50 ns, with about 380 delay cells in each segment, for a total delay of 19 μs. The input sample rate in Fig. 6.6 was limited to half of its maximum value, i.e. 10 Mssamples/s, since the delay line in its "half- granularity mode" is only able to handle inputs up to half the maximum rate.

As expected from eq. 6.1, the power consumption of the delay line grows linearly with the input

sample rate. The "low-jitter" mode of the delay line increases the line's power by approximately 10%, but the resulting SNR is 3 dB higher. The low-jitter mode only increases the delay line's energy consumption by 10%, because the power consumed to charge the extra capacitor in the low-jitter mode in each delay cell (i.e. C2 in Fig. 5.4) is only 10% of the cell's total power consumption. Finally, the reduced granularity mode almost halves the line's power consumption, at the cost of the chip being able to handle inputs of only up to half the maximum rate, i.e. 10 Msamples/s.

The power of the arithmetic unit saturates for high sample rates; in such cases, e.g. input rates higher than 8 Msamples/s in Fig. 6.6, samples from multiple taps are concurrently inserted to the adder, leading to a single multi-way addition operation. For low-rate inputs, most samples are processed from the adder independently in distinct operations; the energy of the adder's control part is dissipated once per sample, as opposed to once for many samples, as is the case for high-rate input. As expected, the power of the arithmetic unit is smaller than that in [33]; this is due to the optimized adder structure (for power), through the use of RCA adders, at the cost of slightly increased adder cycle time.

For a given frequency response, the power consumption can be significantly optimized for low-sample-rate inputs. When the input is limited to low rates, the delay line can be programmed for a much coarser granularity, effectively using fewer cells, with greater delay each, for the same response. This brings down the power proportionally, as experimentally verified.

As an example, the delay line was re-programmed, this time for a maximum input sample rate of 5 Msamples/s, reduced by four times compared to the one used in the previous experiment. This results in the delay line being programmed for a granularity reduced by the same factor, i.e. one quarter of the delay cells of the previous example, each of four times more delay. The

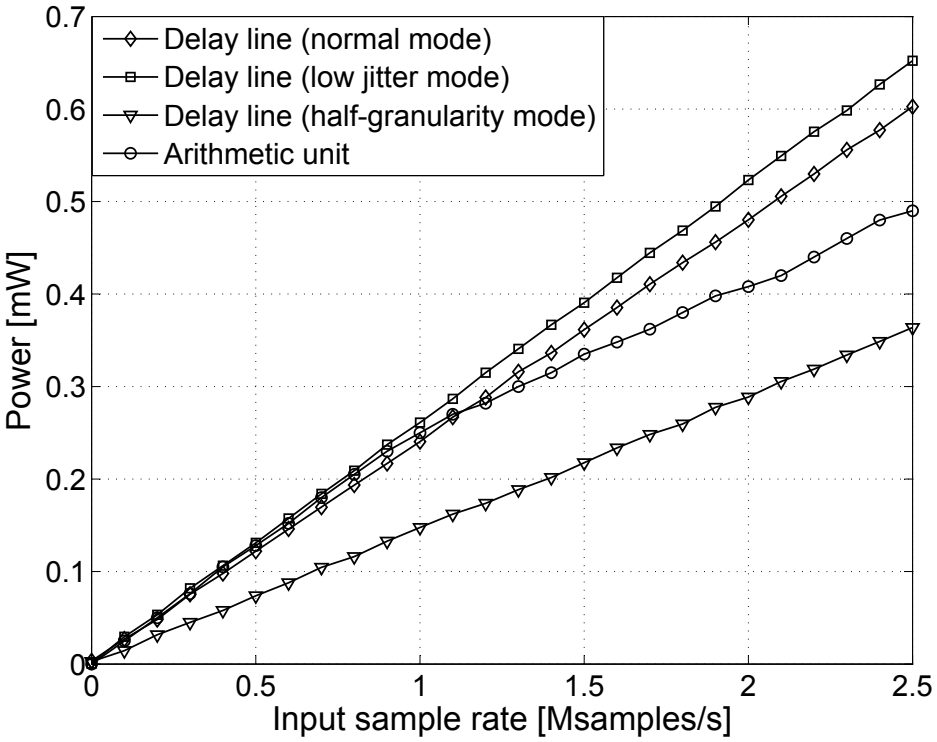

Figure 6.7: Power dissipation of the CT digital FIR key components vs. input sampling rate for a PCM 8 kHz input, using a coarse granularity for the delay line.

input sample rate is now swept up to half the new maximum input rate, i.e. 2.5 Msamples/s. The power consumption of the delay line, shown in Fig. 6.7 is four times smaller in all modes. The consumption of the arithmetic blocks (i.e. the multiplier and adder) remained the same, since their rate of activity remained intact and no modifications were made to them.

The leakage and bias power of the entire chip is only 40 μW, *reduced by a factor of* 6 *compared to [33].* This results from both the technology used, i.e. 130 nm vs. 90 nm in [33], as well as careful sizing and placement of high-V_T devices. The dynamic power is almost 50% larger compared to [33] for programming of the two designs, i.e. delay line granularity and segment delays. As a result, this chip consumes less overall power, static + dynamic than the one in [33] for inputs slower than 1 Msample/s.

6.5 Effect of automatic tuning

Fig. 6.8 evaluates the effect of the automated tuning. Such tuning effectively programs the delay of all FIR segments to very high precision. The chip's frequency response is shown at three different stages, starting from reset. Here the target was a low-pass response with a 14-kHz cutoff, periodic every 50 KHz, which translates to 20 μs segment delays. Initially the segment delays are not precisely equal, causing the response to lack large stop-band rejection. Furthermore, all segment delays are smaller than the desired value, leading to a scaled response, as shown in the top part of Fig. 6.8. The middle part of Fig. 6.8 shows the response after global tuning. The latter increases all cells' delay, directly scaling the response. The bottom part shows the final response, after local tuning of all 15 delay segments. Not only is the response fine-tuned to have the desired periodicity and cut-off frequency, but it also has much sharper notches and larger stop-band attenuation, suggesting good matching between the delay segments.

6.6 Additional measurements

In the previous sections, the key properties of the chip were demonstrated, i.e. the independence of frequency response from sample rate, as well as the input-dependent power consumption. A set of additional measurements are illustrated in this section, to demonstrate several additional interesting features: the inherent lack of aliasing of this chip, the effect of the various operation modes of the delay line, and the range of the responses that can be obtained using the automatic tuning.

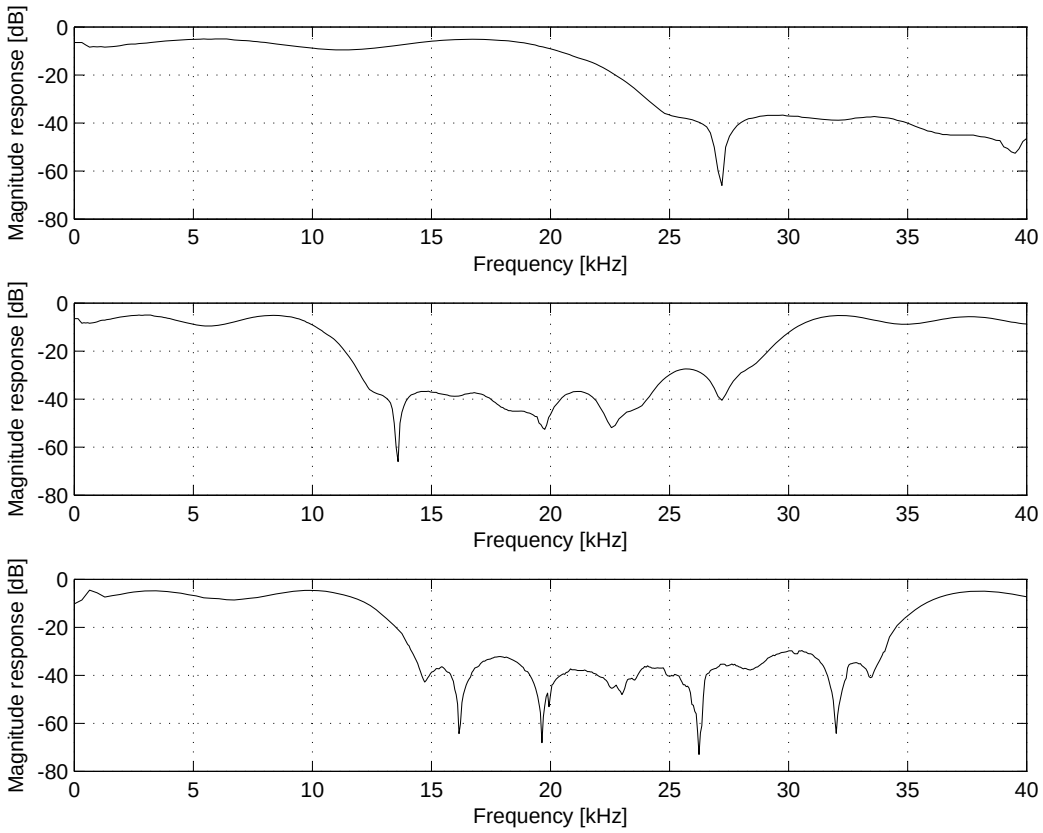

Figure 6.8: Demonstrating delay segment automatic tuning. The three plots show the frequency response (programmed low-pass) before tuning (top), after global tuning (middle), and after both global and local tuning (bottom). The responses were obtained using a PCM format, at 1 MHz sample rate.

6.6.1 Absence of aliasing

An important advantage of the new CT DSP chip is that it has *no inherent aliasing*. The lack of aliasing is a property common to all CT DSPs. Even when the input is sampled using a synchronous ADC, no aliasing is caused by the DSP itself.

Fig. 6.9 shows an associated experiment, for the frequency response shown in Fig. 6.4. A 42 kHz input is sampled at 500 KSamples/s and processed using a low-pass response, periodic every 50 kHz. Even though the location of the input is on the second lobe of the frequency response, it is not aliased back to the baseband, as would have been the case with any synchronous DSP system.

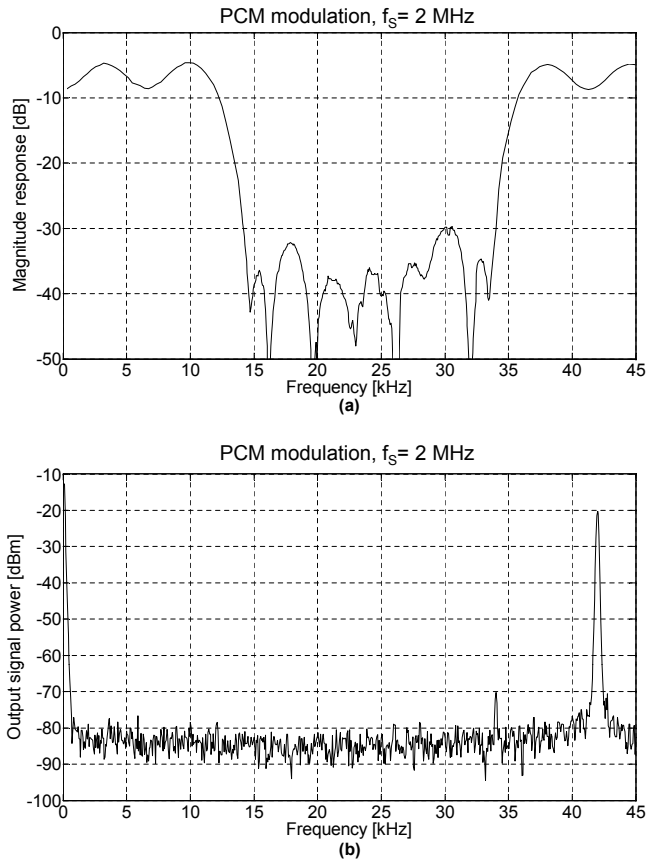

Figure 6.9: Demonstrating the inherent absence of aliasing in the CT DSP chip. The chip was configured to the frequency response shown in (a) and fed with a 42 kHz input, located in the 2nd lobe of the response's passband. Contrary to synchronous DSP systems, the input is not aliased back to the baseband. (RBW = 300 Hz)

6.6.2 Delay line operation modes

As expected, the chip can operate in various operation modes by using the digital bits controlling all cells of the delay line. The effect of the various power modes of the delay line on the output SNR is verified to match theoretical expectations.

Fig. 6.10 shows the output of the system for two configurations of the chip's delay line, the normal and low-jitter modes. In the latter, the jitter power of the delay line is reduced by a factor of 2· compared to the normal mode. As a result, the SNR in the low-jitter case is higher by 3 dB;

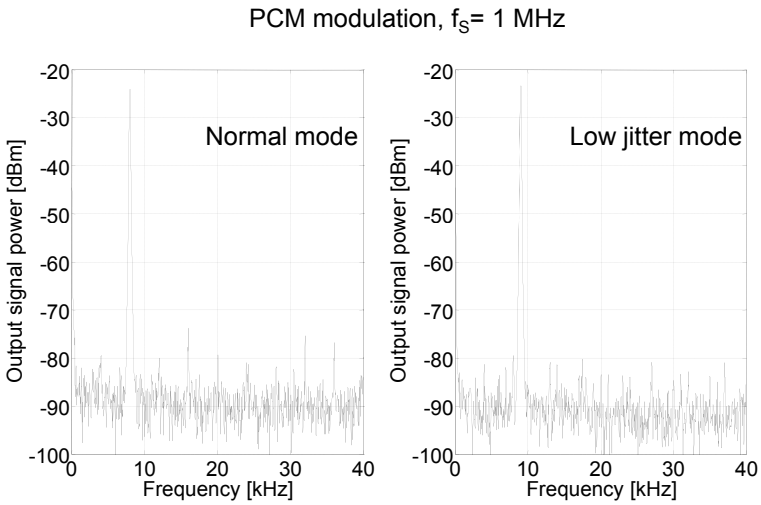

Figure 6.10: Different operation modes of the delay line and associated jitter. The output for a 8-kHz input is shown (using PCM format) for the normal and low-jitter modes of the delay line. (RBW = 300 Hz)

this is demonstrated through the reduction of the "noise-floor". The result for the half-granularity mode is equal to that of the normal mode, i.e. the SNR is 3 dB lower compared to the "low-jitter" mode.

6.6.3 Types and range of frequency responses

A key benefit of this chip is that its frequency response can programmed for various types and ranges. Fig. 6.11 shows sample band-pass, band-stop, high-pass and low-pass responses, obtained using a ΣΔ ADC. Each response was taken using two different sample rates, 1 and 5 Msamples/s. As in the case of Fig. 6.4, no internal adjustments were made each time the sample rate changed: each type of response remains almost intact for the different sample rates.

Fig. 6.12 illustrates the wide programmability of the chip, in terms of the periodicity of the frequency response. Responses in the range of 200 kHz (top) and 1 MHz (bottom) are shown, as obtained by using the automatic tuning. These two sample responses were obtained by using the

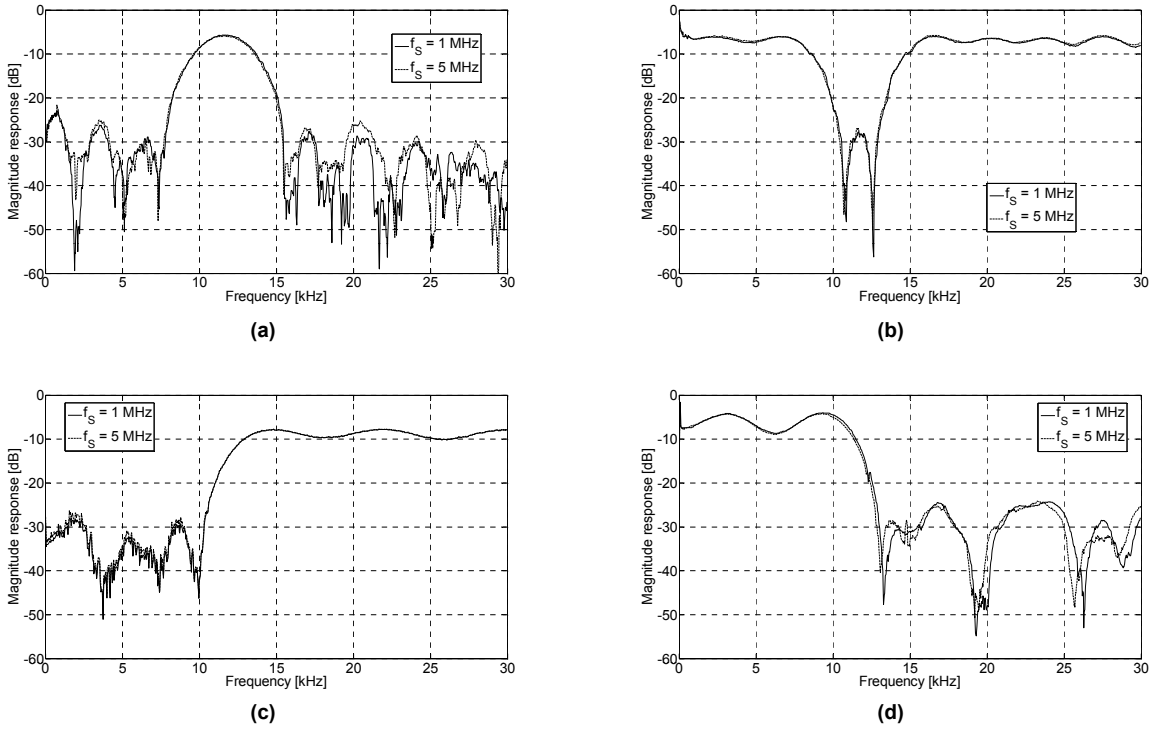

Figure 6.11: Demonstrating different types of frequency responses (a) band-pass, (b) band-stop, (c) high-pass, (d) low-pass. Measured with ΣΔ modulation, at two sample rates: 1 and 5 Msamples/s. RBW = 300 Hz)

automated tuning to change both the delays of all the delay cells inside the chip, as well as the numbers of cells in each segments.

6.7 Comparison to other DSP systems

In the previous sections, various measurements were presented for the implemented CT DSP. This section presents a comparison between DSP system using this chip, and state-of-the-art DSP systems, both CT and DT. Even though this chip is not optimized for performance or power, bur rather for multi-purpose usage, its efficiency is close to that of an optimized state-of-the-art DSP system.

Table 6.1 summarizes the performance of the ADC/DSP/DAC system containing the CT digital FIR filter. For all types and rates of inputs, the signal-to-noise and signal-to-distortion ratio (SNR

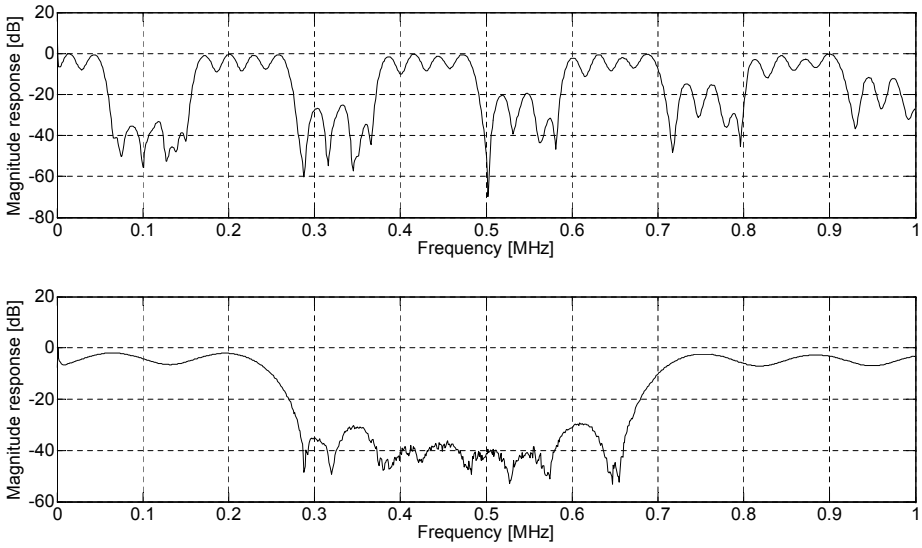

Figure 6.12: Demonstrating the range of the chip's frequency response. Shown are responses periodic every 100-kHz (top) and 1 MHz (bottom). Measured with PCM format and sample rate of 10 Msamples/s.

and SDR respectively) range from 43 to 52 dB, as measured for the low-pass configuration of Fig. 6.4 with maximum gain of 0 dB, over a band of 30 kHz. Unlike synchronous 8-bit systems, the SDR can exceed the theoretical limit of 50 dB without making any use of oversampling, as predicted theoretically in [4], [27].

Table 6.2 shows a comparison table between the ADC/DSP/DAC system using this chip and prior work for both CT [33], [34] and discrete-time (DT) [115] DSP systems. A figure of merit was used for comparison of the different designs, as follows:

$$\text{FOM} = \frac{P}{f_{MAX} \cdot 2^{\text{ENOB}} \cdot N} = (\frac{E_{sample}}{N}) \cdot (\frac{OS}{2^{ENOB}}) \tag{6.2}$$

, where P is the consumed total power, f_{MAX} the maximum signal frequency that can be processed, ENOB is the effective number of bits as determined from the output SNDR as $ENOB = \frac{SNDR-1.76}{6.02}$ and N is the filter order. As stated in the right part of 6.2, the FOM can also expressed as the product

Table 6.1: ADC/DSP/DAC performance table.

Input format	**PCM**	**PWM**		**ΣΔ**	
Type	**Sync**	**Sync**	**Async**	**Sync**	**Async**
Resolution [bits]	8	1	1	1	1
Min (max) sample rate [Msamples/s]	0.1 (10)	0.6 (2)	0.2 (1)	1 (20)	0.5 (2)
Power consumption for min (max) sample rate [mW]					
Normal mode	0.07 (3.1)	0.3 (0.78)	0.12 (0.43)	0.44 (5.2)	0.21 (0.77)
Low-jitter mode	0.07 (3.3)	0.32 (0.82)	0.14 (0.46)	0.48 (5.6)	0.26 (0.82)
Half granularity mode	0.06 (2)	0.22 (0.6)	0.12 (0.33)	0.33 (3.4)	0.18 (0.6)
Standby power [mW]	0.04				
Automatic tuning blocks' power (when active) [mW]	0.2				
In-band SNR [dB]					
Normal mode	51	40	43	41	40
Low-jitter mode	54	43	45	44	43
Half-granularity mode	51	40	43	41	40
Signal-to-distortion ratio [dB]*	47 – 52	42 – 45	44 – 50	43 – 45	42 – 44
IM3 distortion (full-scale sum of two tones in pass-band, 1 kHz apart) [dB]	-50				

* 2.5 kHz input, 13 kHz low-pass response, for sample rate range indicated

of two terms: the energy to handle a single sample, E_{sample}, divided by the number of taps, and the ratio of the digital format's over-sampling ratio (OS) divided by the resolution. This last ratio is indicative of the efficiency of a digital modulation (how many samples per cycle are required by the specific format, in order to achieve a certain resolution). The FOM of. 6.2, therefore, expresses the energy required for a given DSP system to process one period of the maximum-frequency signal, normalized to the number of taps and resolution. A smaller FOM indicates a more efficient DSP system. In all designs considered in Table 6.2, as well as the system using the new CTDSP chip, only the DSP core power was included for the calculation of the FOM.

For different formats the signal bandwidth f_{MAX} changes, as a result of the different sample

rates and OS ratio for each ADC used on the test setup. Each modulation results in a different minimum spacing between samples, and also in a different requirement for the filter's frequency response. The latter must be periodic with period at least $2f_{MAX}$, similar to the case in synchronous DSP systems. This affects the programming of the delay segments, i.e. how many delay cells need to be used to generate the required overall delay, changing E_{sample}. This, along with the efficiency of the modulation itself, causes different FOM values in each case; the FOM increases for over-sampled modulations.

The FOM for the system we present varies significantly across the ADC formats. When using PCM inputs, the system operates much closer to the Nyquist rate and process higher-bandwidth signals more efficiently, reducing the FOM. The result is a FOM better by a factor of 60 compared to the previous low-frequency CT DSP [33], and within a factor of 3 and 1.5 from state-of-the-art, GHz DT [115] and CT [34] DSP systems, respectively.

The primary reason why this chip's FOM cannot surpass the one for state-of-the-art DT and CT DSP designs, [115] and [34] respectively, is the use of a larger-feature-size technology. This increased the dynamic power consumption of both the delay cells and the arithmetic units, with respect to what we could have obtained by using a smaller-feature-size technology.

Compared to the state-of-the-art CT DSP [34], this chip incorporated asynchronous handshaking in the delay cells, which brought about a large power penalty. In [34], handshaking was eliminated due to the requirement for large speed of the delay cells. In the design in [34], the arithmetic hardware was implemented in an analog fashion, whereas in this design, a less energy-efficient but more flexible digital approach was used.

Table 6.2: Comparison of the ADC/DSP/DAC system using the designed chip to prior art in DT and CT DSP systems.

Parameter	**Schell**	**Kurchuk**	**Agrawal**	**This work**				
Process	90 nm	65 nm	32 nm	130 nm				
Supply voltage	1 V	1.2 V	1 V	1 V				
Clock	no	no	yes	no				
Resolution	8 bits	3 bits	3 bits	8 bits				
DSP core area	0.55 mm^2	0.07 mm^2	0.01 mm^2	5 mm^2				
Modulation	Δ	PCM	PCM	PCM	PWM		$\Sigma\Delta$	
Sync or async	Async	Async	Sync	Sync	Sync	Async	Sync	Async
Max. frequency	20 kHz	3.2 GHz	1.05 GHz	5 MHz	100 kHz	80 kHz	500 kHz	50 kHz
Max. power	3 mW	9.6 mW	24 mW	1.3 mW	0.8 mW	0.5 mW	1.8 mW	0.8 mW
Number of taps	16	6	4	16				
Energy/sample	280 pJ	40 fJ	15 fJ	126 pJ	290 pJ	310 pJ	185 pJ	410 pJ
FOM	3.3 pJ	30 fJ	15 fJ	45 fJ	1.4 pJ	1.2 pJ	88 fJ	2.8 pJ

6.8 Comparison to other reported results

The comparison of Table 6.2 does not account for the true advantage of the chi, which is its multi-format and multi-rate operation. Previous CT DSP systems [46], [33] and [34] were restricted to use asynchronous level-crossing-sampling ADCs. Furthermore, the approaches in [46] and [33] operated using a 1-bit datapath, coming from Δ-modulation encoding, with the potential hazard of a single-sample internal loss permanently affecting the entire system. The state-of-the-art CT DSP system [34] is a 3-bit system, very hard to extend to different bit-widths.

The true advantage of the new chip is its general-purpose usage, i.e. its ability to handle a wide variety of digital encodings, possibly with different sample rates each, even asynchronous or with varying sample rates, e.g. due to the Doppler effect. Such a property is presented for the first time, even among CT DSPs, and is inherently impossible to classical synchronous DSPs.

6.9 Summary

This chapter presented measurement results for the new CT DSP chip. The chip was used as part of an ADC/DSP/DAC system which, to the best of our knowledge, is the first system able to process using different rates and formats without adjustments.

The measurements highlighted the smooth way in which the chip can handle inputs using different modulations, from 0.1 to 20 Msamples/s sample rate. For certain cases, the SDR can exceed that of synchronous DSP systems. The wide programmability of the chip was also confirmed, through usage of the on-chip automatic tuning mechanism. As with all CT DSPs, this chip too has signal-dependent dynamic power consumption, as well as internal absence of aliasing. Through the reduction of the leakage power by a factor of 6, this chip consumes less overall power, i.e. static+dynamic, compared to the previous baseband CT DSP [33] for similar filter configurations, at input rates of 1 Msamples/s or below.

This chip serves as a generalization of the work in [33], both extending its usage to more types of inputs as well as providing a more general and scalable approach. Apart from efficiently handling more signals, it was demonstrated that through significant reduction in leakage power this chip consumes less total power than the one in [33] for low-rate inputs. Despite its much wider operation range, this work is still able to achieve a figure-of-merit close, that is close to the ones for state-of-the art CT [115] and CT [34] DSP systems, in particular by a factor of 1.5 and 3, respectively.

Chapter 7

A Methodology for Designing Real-Time Delay Lines with Dynamically-Adaptive Granularity

Chapters 4 to 6 presented the complete design and measurement results for a new CT DSP chip with multi-rate capability. This chapter focus on optimizing one key component: the *delay line*. The goal is to significantly decrease the line's average dynamic power, by modifying its granularity according to its input traffic. This enhancement is not only a good match for CT DSPs, but also for other applications which are constructed around delay lines.

In particular, this chapter introduces a complete methodology for designing reconfigurable delay lines, which dynamically adapt granularity to traffic, on-the-fly, without stalling or disturbing normal operation. Such delay lines are large real-time pipelines with many individual delay cells; we show how to change the number of cells in the delay line *dynamically* and according to the

input traffic. During sparser traffic, the system is reconfigured to the proper coarser-grain mode, thereby reducing total energy, and it reverts to fine-grain mode during denser traffic. In each case, overall delay is preserved.

The contribution of this chapter is the methodology to build such adaptive delay lines, presented for two complete examples: a *bi-modal* and a *tri-modal* line. The microarchitecture is constructed with several asynchronous controllers, used for a decomposed design solution allowing easy extensions and modifications. Simulations on the implementation of these lines indicate significant power savings, with respect to the baseline non-reconfigurable line, which can obtain 45.1% and 70.2% for the bi-modal and tri-modal line, respectively. Designs with larger number of granularity settings can further increase such savings.

This chapter is structured as follows. First a motivation for adaptive delay lines is presented in Sec. 7.1. The contributions of our technique will be presented in Sec. 7.2. Before going in detail into presenting the adaptive granularity system, the latter will be first shown in an overview in Sec. 7.3. The first example, for a bi-modal line, will be presented at its original version in Sec. 7.4. In Sec. 7.5, a subtle design bug in this version is highlighted, and corrected in Sec. 7.6. Extensions to support multiple granularity settings are introduced in Sec. 7.7, with a detailed example of the design of a tri-modal. Detailed simulation experiments are presented in Sec. 7.8. A new approach to generate test patterns for simulation of such delay lines is presented, together with a technical discussion of the effect of technology scaling on this design methodology and of future directions. Finally, Sec. 7.9 concludes this chapter.

7.1 Motivation for adaptive granularity

This section addresses the motivations behind adaptive granularity delay lines, as well as our contributions to the problem. We begin with the role of delay lines in modern digital systems, and proceed to describe the limitations of delay lines which call for adaptive granularity. This section concludes by sketching our contribution, which is an entire system for adaptive granularity management.

7.1.1 Role of delay lines

In modern integrated circuit (IC) technologies, real-time delay lines find a wide range of applications. Delay lines, which provide calibrated timing without the aid of a digital clock, are key components in PLLs/DLLs [116], oscillators with digital control [117], digital memories [118], wireless tranceivers [119], voltage converters [120] and microprocessors [121]. Furthermore, due to the reduced supply voltage headroom in modern IC technologies, and the trend toward shifting many analog and digital operations to the time domain, delay lines are also important components in certain analog-to-digital converters. For example, in a new ADC approach [122], delay lines replace conventional methods of discretizing an analog voltage using comparators. Finally, delay lines are the largest power component of CT DSPs [46], [33], [34], [36], around which delay lines will be presented; this is the case for many of the other applications mentioned above as well. Hence, optimizing the line's power has a critical impact on overall energy efficiency.

In CT DSPs, delay lines are used as part of the DSP core, whose versions to date implement CT digital FIR filters. The delay line is used to generate the multiple delayed copies of the input channel, as required for the FIR operation. Fig. 7.1 shows a typical case of a CT DSP input:

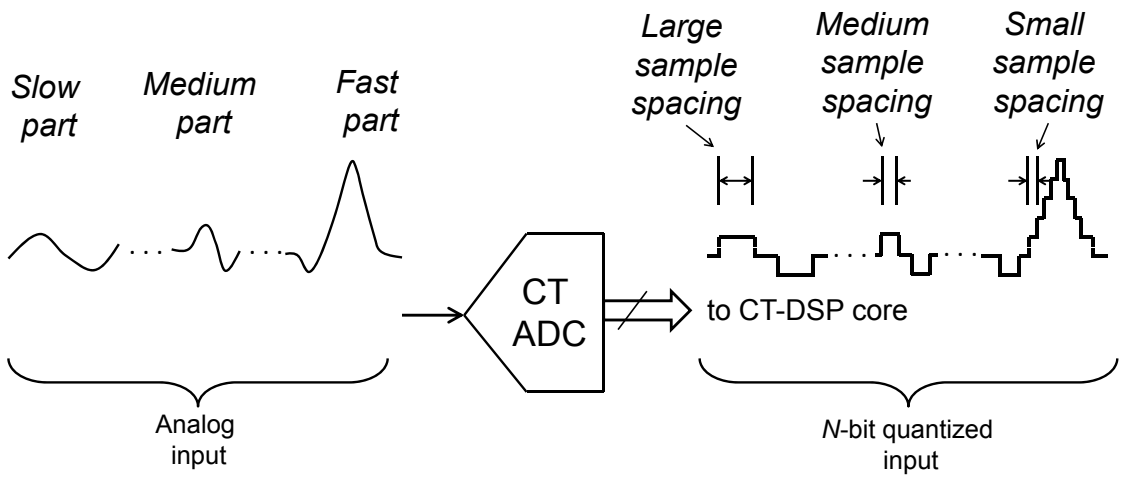

Figure 7.1: Traffic and event spacing in a CT DSP system.

inputs are mostly slow-varying, resulting in widely-spaced samples presented to the DSP core. The sample spacing decreases as either the analog input's amplitude or frequency grow larger. At very large input amplitude and frequencies, samples are spaced closely, resulting in high traffic. Likewise, intermediate cases lead to medium spacing. This dynamic nature of sample spacing, dominated by medium and low cases, will be exploited toward power optimization in the delay line. For a more detailed analysis of the traffic in a CT DSP, see [27].

7.1.2 Prior designs and limitations

Prior delay-line designs used in CT DSP's, such as the one presented in Chs. 4-6, are *non-adaptive* real-time pipelines, with a static (i.e. fixed) granularity, which is fine enough to handle input streams of a worst-case input rate [33], [29], [46]. However, supporting higher-rate input streams requires delay lines with finer-granularity, and proportionally larger energy consumption. In particular, *the total energy consumption to process a single sample* travelling through a delay line is *directly proportional to the number of delay cells,* i.e. the granularity of the pipeline. However,

CT DSP's, in the applications they are intended for such as audio and biomedical applications, receive input streams of significantly *varying rate,* where the inputs can be mostly composed of long periods of silence followed by short bursts. Furthermore, even during these input bursts, sample spacing can also vary significantly, from sparse to dense. While approaches have been proposed for dynamic recalibration and tuning of delay lines [123], there is no prior work on dynamic granularity optimization. A system which explores optimized granularity [28] has been proposed, but without dynamic adaptability: only one statically-selected granularity is implemented.

7.2 Contribution: adaptive granularity delay lines

The contribution of this chapter is a complete approach to support dynamic granularity reconfiguration in a pipelined delay line. In particular, the deliverable of this work is a systematic methodology to design a time-management and granularity-control unit for the delay line. The system monitors traffic in real time and reconfigures the entire line to the most energy-efficient setting that can handle its current input.

The proposed approach introduces two key control components. An *asynchronous digital traffic-monitoring unit* classifies incoming traffic and chooses the desired delay line granularity setting, and an *asynchronous micro-pipeline* then serially reconfigures the entire delay line. The traffic-monitoring unit compares incoming traffic to one or more pre-defined threshold rates and classifies the traffic accordingly. The delay line is then dynamically set to operate in the most energy-efficient granularity mode which is still able to support the current traffic. This strategy is shown at an abstract view in Fig. 7.2, for a tri-modal delay line: for each traffic case, i.e. for each case of sample-spacing, the adaptive system reconfigures to the most energy-efficient setting

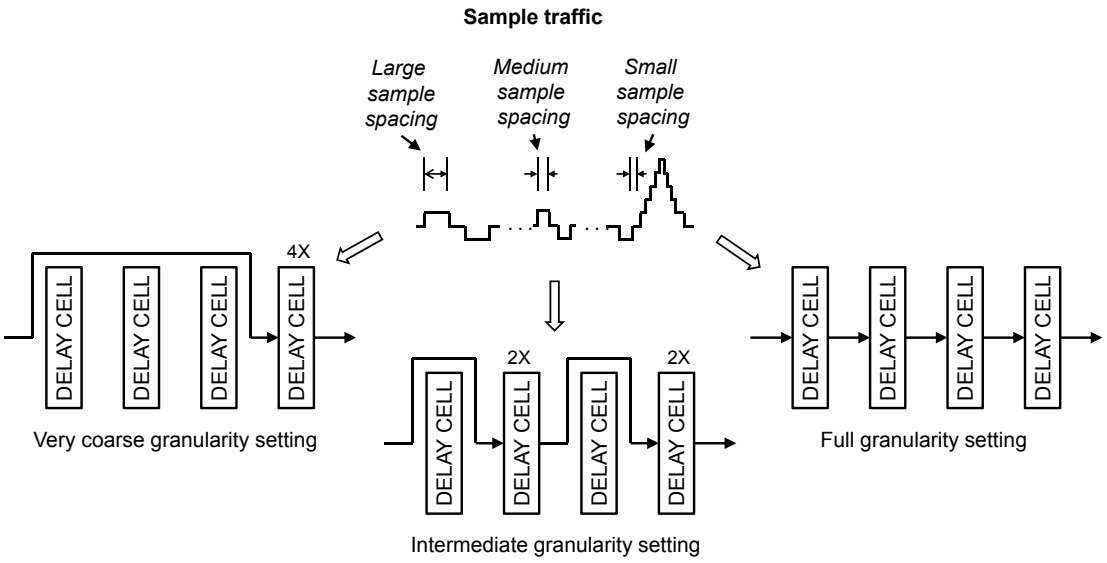

Figure 7.2: Abstract view of a tri-modal adaptive delay line, showing the three different types of granularity settings.

which can still support the traffic, i.e.without losing samples or causing sample congestion. As an example, when the sample spacing is more than four times the minimum spacing, then the line is set to the coarsest granularity, *only using one quarter of its cells where each now has four times the delay*; this setting preserves the overall line delay and can still support this slow traffic with significantly reduced energy consumption.

Three key challenges have to be tackled in this effort. Granularity reconfiguration must occur during normal system operation, without having to pause or shut down the system. Reconfiguration must occur safely and without overwriting samples already inside the delay line, which could cause temporary or permanent distortion to the CT DSP's output. Finally, the additional control components must be lightweight, both in terms of area and power, so that their own power does not cancel out the savings in datapath power.

Two complete design paradigms for adaptive delay lines are presented, with *bi-modal* and *tri-modal operation*, respectively. While fixed traffic thresholds are used for presentation, the

position of the thresholds can be easily modified using the design methodology of this work. The extension to include even more thresholds is also made clear through the two design examples we present. Lower thresholds allow the use of even coarser-grain settings for the delay line, leading to increased power savings for inputs that qualify as low-rate; the low-rate inputs would then achieve greater energy savings, but at the cost of less total traffic classified as low-rate. Furthermore, more thresholds enable the use of more granularity modes, hence a better and smoother adaptation of the overall power to the incoming traffic. This approach can also lead to increased power savings, at the cost of a larger complexity for traffic monitoring. Thus, a new class of design space toward energy optimization of a CT DSP system is proposed: given statistics of the system's expected traffic, the designer can set both the number and positions of the granularity-modification thresholds accordingly, effectively controlling the trade-off between energy savings in each mode and the duration of each operation phase, minimizing the overall system power.

7.3 Simplified view of an adaptive granularity delay line

This section reviews the overall adaptive delay line system at an abstract level, before going into technical detail in the next sections. The new system enables the adaptive granularity management of a delay line. It monitors input traffic at the left of the delay line, in real time, and reconfigures the rest of the delay line as traffic patterns change.

Fig. 7.3 shows a top-level view of the original version designed for a bi-modal adaptive delay line, presented in [37]. Even though this version is not final, and also includes a subtle bug which can cause system malfunction, it is presented here since it is intuitive and can help understand the underlying concepts better. The enhancements for the bug correction, as well as the support of

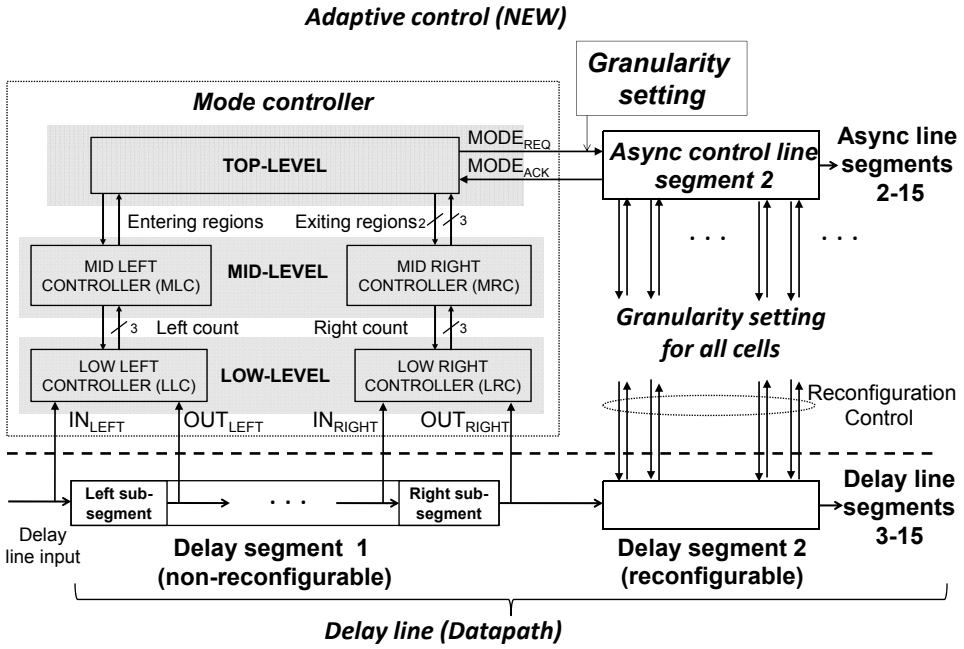

Figure 7.3: Proposed adaptive granularity system: overview of the original approach for bi-modal operation.

more operation modes [38], will follow in later sections. The structure for these versions is similar, so the one shown in Fig. 7.3 will be shown as reference for the system description.

Fig. 7.3 shows a top-level overview, divided into datapath and control. The datapath is a delay line with 15 segments, only 2 of which are shown. The control involves two blocks. The traffic-monitoring unit, *mode controller,* monitors traffic inside the first segment, which itself is non-reconfigurable and hardwired to always operate in fine-grain mode. It also chooses the mode of operation for segments 2-15, which is then serially passed down using an asynchronous micro-pipeline, the *control line.*

The mode controller monitors traffic in three levels. The lowest level observes the input and output of individual samples in two sub-segments (left and right) of the first delay segment. It reports their current occupancy, which implies information on the spacing between their consecutive samples. Based on this information, the mid level identifies each entire contiguous *region* of high-

traffic, i.e. multiple consecutive closely-spaced samples, entering or exiting segment 1. Finally, the top level counts the total number of high-traffic regions currently contained inside the first segment. Segments 2-15 are configured to coarse-grain mode only when the entire first delay segment has low-rate traffic, i.e. the top-level count is 0. Whenever the global count changes from zero to non-zero, or vice-versa, reconfiguration is applied and serially propagated to segments 2-15 via the control line at the proper instances and at a variable rate, depending on the current occupancy of the segments. As will be shown later, the multi-mode approach will simply involve the monitoring of more types of regions.

A key feature of the approach is that, even though the delay line is real-time, the *application* of reconfiguration is performed by means of a lightweight asynchronous control line. Although the control line operates asynchronously, and does not track the operating rate of the delay line and, in fact, has much lower latency, the two lines are carefully synchronized through handshaking so that the reconfiguration signal does not overtake samples in the delay line. The benefit of this approach is to entirely avoid a high-overhead real-time control delay line.

The mode controller introduces hysteresis in its traffic characterization, to avoid thrashing: rapid reconfiguration between modes. The requirement that the entire first segment must have zero high-traffic regions in order to reconfigure, guarantees that traffic will have to be below a certain threshold for a sufficient time before it can be characterized as "low". This requirement is equivalent to enforcing hysteresis in the monitoring strategy and avoids frequent energy-wasting reconfigurations during rapidly-varying inputs, in which case the line will pessimistically operate only in fine-grain mode. Note that such hysteresis is one-sided, since it only applies to categorizing "low" traffic. In contrast, the observation of any "high-traffic" region in the first delay segment immediately results in classification of the overall incoming traffic as high-rate.

The scalability and modularity of this monitoring approach comes from the very distinct features of the mode controller's levels. The lowest level only classifies each pair of consecutive samples as "widely" or "closely" spaced. The notion of "traffic threshold rate" is hardwired here, and here alone, by defining the maximum sample spacing for "closely-spaced" samples. Changing the traffic threshold rate, to explore the particular design space, is simple and involves only a slight modification of this level of the mode controller.

Two enhanced versions of the adaptive delay line are also presented, following the initial one: the first enhancement is used to solve a potential bug that we identified in the original approach, which can in rare cases lead the asynchronous traffic controller to failing. The second enhancement is the addition of more configuration modes, presented around a tri-modal version.

7.4 Bi-modal adaptive delay line: original approach

This section presents the key components of the original version for a bi-modal system in detail. The delay line is introduced first, followed by a bottom-up presentation of the mode controller and then the asynchronous control line. While a concrete system configuration is outlined, directions for extending the components for different settings is also included. We note here that this version is not final, since it includes a subtle bug in the mode controller. We then present the updated version of the system with the bug fix in a later section; the original version is included for pedagogical reasons.

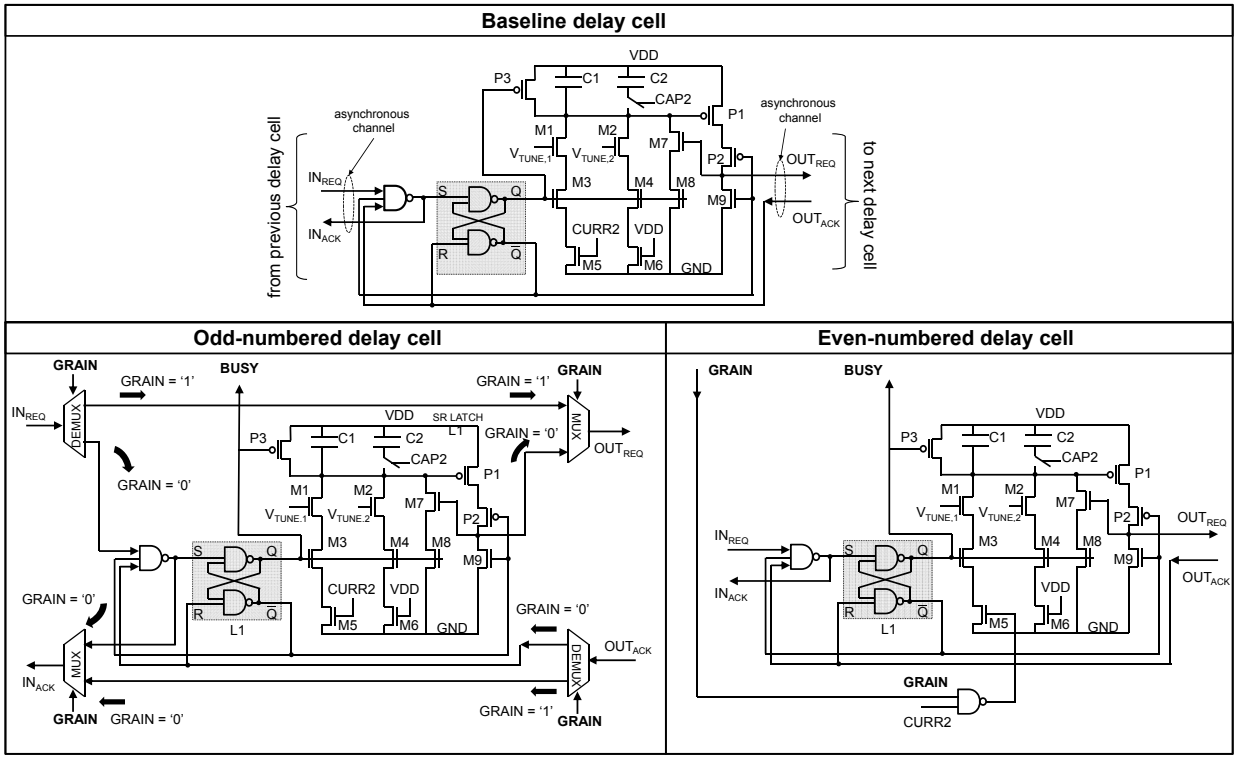

Figure 7.4: Baseline and bi-modal adaptive delay cells.

7.4.1 Delay line

The delay line is a foundational block of any CT DSP core, as well as many other calibrated systems. It is the only block common to both traditional CT DSP designs and the proposed adaptive design. It is composed of a large number of individual *delay cells.* First an overview of the baseline (i.e. non-adaptive) cell design is reviewed, followed by two new cells for use in the adaptive system.

The CT DSP line is a pipeline with 15 segments. Each segment is composed of 500 delay cells very similar to the design presented in [33]. A segment creates time-shifted versions of each input sample while preserving the time interval between consecutive samples. As discussed previously, the term "sample" will be used to refer to the timing signal of an *N*-bit sample, and not to the data bits. The dynamic granularity management only focuses on movement of timing signals via the

delay line and not on data movement and storage, which is not affected by out method. The rate of the incoming samples are limited to a maximum value, hence to a minimum spacing. In the context of CT DSPs, this limitation is imposed by the CT ADC. Delay cells need to have a delay at most equal to such a minimum spacing, to avoid over-writing of samples.

The baseline delay cell is used to implement a small increment of delay, based on the slow charging of a capacitor. It has already been presented in detail in Chap. 5. As shown in Fig. 7.4, each delay cell has one input and one output channel. The cell's operation is initiated when its predecessor (i.e. left neighbor) passes a new sample using an active-high input request. The cell immediately acknowledges the left interface, so the 4-phase transaction quickly completes on the input side, and concurrently starts its main operation. Using both its current sources (M1-2), a capacitor is slowly charged until a positive feedback block detects that the capacitor voltage exceeds a well-defined voltage threshold. At this point the feedback block rapidly completes the capacitor charge and passes the sample to the successor cell. Upon acknowledgment, the cell completes its operation and the capacitor is fully discharged, and then waits for the next sample. The status signal *BUSY* is safely asserted high during the entire active phase, reporting an occupied cell. For audio applications [33], delay cells have a delay in the order of $50-60$ ns, while delay segments have a delay in the order of 25 μs.

The delay cells and segments are tuned to compensate for process variations, as described in chapter 5. The tuning first adjusts the analog voltages $V_{TUNE1,2}$ so that a large group of training delay cells has a specified delay value. In this way, the average delay of the training cells is tuned to the desired value. The adjusted voltages $V_{TUNE1,2}$ are then transmitted globally to all delay cells, so that all cells have approximately the desired delay value. Due to layout mismatches across the same chip, delay of different cells on the same chip were observed to deviate up to $\pm 3\%$ from

the cell's nominal value. Finally, the number of delay cells within each segment is also tuned as in [33], so that the delay of all segments is close to the desired value. The resulting mismatch between delay segments after tuning was observed to be always less than $\pm 1\%$ of the specified delay.

The new reconfigurable delay cells build on these baseline cells, but with added instrumentation to provide status information and dynamic reconfiguration capability. The delay cells are grouped into even-odd pairs, as shown in the bottom part of Fig. 7.4. An even cell is bypassable and an odd cell can undergo a delay doubling by disabling one of its two current sources. The reconfigurable cells need two current sources, each of which is tuned as in the baseline case. A control signal, *MODE,* is used for reconfiguration: a 0 value puts the cell group in coarse-grain mode, where the pair only has the capacity to hold one sample, but preserves the same overall delay of a baseline two-cell group. Each cell's energy is delay-independent [33], thus in coarse-grain mode only one cell dissipates energy and therefore the delay line's overall energy consumption is almost halved. In the final 15-segment adaptive delay line, the first segment is non-reconfigurable and the remaining 14 segments are reconfigurable, as shown in Fig. 7.3. However, to ensure proper matching timing of all segments' delay, the first segment should also be implemented with reconfigurable delay pairs, but statically hard-wired to fine-grained mode.

Extension: The above strategy can easily be modified to support different coarse-grain settings. For example, $\frac{1}{3}$ of the nominal granularity can be used as a threshold by bypassing 2 delay cells in every group of 3 and increasing the delay of the remaining cell three-fold. To achieve a delay 3 times larger than the baseline cell, it suffices to provide different values for the delay cell tuning voltages $V_{TUNE1,2}$, so that by disabling the second current source the delay increases by a factor of three. There is no need to have three current sources.

7.4.2 Mode controller

The mode controller is the key component of the reconfigurable system. It monitors the start and end of delay line segment 1 and classifies the current input traffic. All three levels of the mode controller are presented in detail. The burst-mode (BM) specifications for all the machines will be provided in Appendix B; this section will only sketch their operation at an RTL level.

Low-level controller

Two low-level controllers (LC's) are used, on the left and right of the first delay segment, to monitor incoming and exiting traffic, respectively, as shown in Fig. 7.3. Each controller only has access to a very small sub-segment of segment 1. For each sub-segment, the corresponding controller reports *if there are 0, 1 or 2 samples within it.* In addition, each of the controllers has 2 input channels, the input and output of a sub-section, equal to 2 delay cells each. Each low-level controller has a single output channel to communicate with the mid level. Its input channels are single outputs from the individual delay cells, and their signal transitions are narrow non-persistent pulses. Its output channel is a 4-phase channel with a 1-of-3-hot request representing an event count.

In particular, each low controller, left (LLC) and right (LRC), are composed of three parts, as shown in Fig. 7.5. A protocol controller first converts its pulsed inputs to 4-phase handshaking. The requests on the two input channels are serialized, via a mutual exclusion (MUTEX) [113], to allow the asynchronous BM controller to process one at a time. Each low-level controller counts the current number of samples inside the corresponding sub-segment and reports it to the mid level. Samples entering or exiting each sub-segment cause a count "increment" or "decrement" operation, respectively. In this design, the count can only take the values 0, 1 and 2, and is explicitly encoded

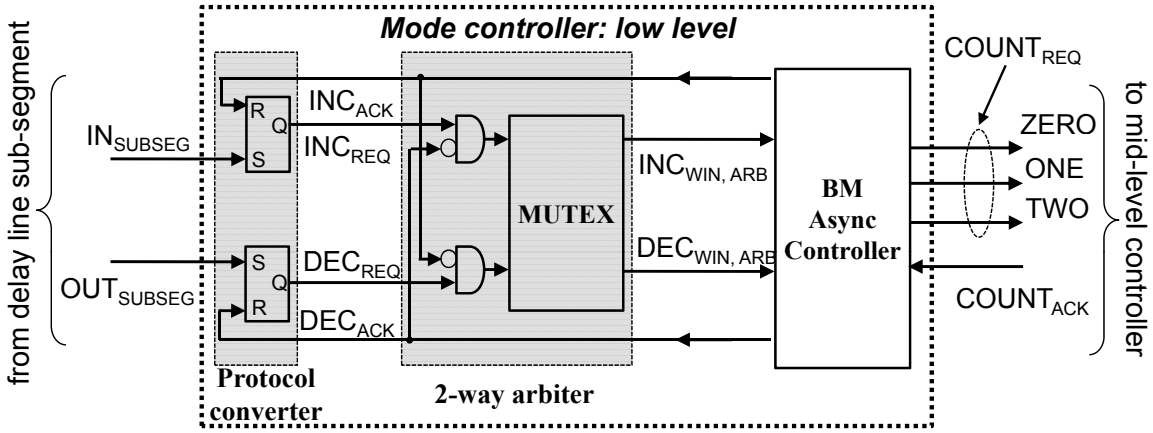

Figure 7.5: Mode controller low-level structure.

in the controller states.

This count implies important information about consecutive sample spacing. When two samples are spaced by more than a 2-cell delay, and the sub-segment is empty with count initially 0, the first sample will enter the sub-segment (i.e. incrementing the count to 1) and then exit (i.e. decrementing the count to 0), before the second sample enters. *Therefore, non-dense traffic results in counts alternating between 0 and 1.* In the opposite case, if the sub-segment is empty, with count initially 0, two samples spaced by less than 2 delay cells will both occupy the sub-segment at a given time, resulting in the count first increasing to 1 and then finally to 2, which identifies the start of dense traffic. *Hence, in dense traffic, counts alternate between 2 and 1.* In sum, a count of 0 always indicates widely-spaced samples, i.e. low traffic; a count of 2 always indicates closely-spaced samples, i.e. high traffic; and a count of 1 continues the current state.

Extension: The sub-segment length sets the notion of traffic threshold. By using more delay cells in a sub-segment, the controller can detect samples spaced by larger time amounts. Widely-spaced samples still keep the low-level count to 0 or 1, while closely-spaced samples increase the count to 1 or more. A count of 1 can belong to both cases.

Role of the arbiter: The arbiter protects the BM controller from receiving non-specified inputs. More specifically, the arbiter ensures that the BM controller *only* receives and processes one kind of sample at a time, entering or exiting, and then it takes enough time to stabilize before receiving the next sample. To ensure this, the arbiter forces the de-assertion of its losing channel (i.e. the one that did not win the arbitration) while the winning channel is being processed, as shown in Fig. 7.5. The loser channel is gated using the acknowledgment of the winner channel; this blocking is not lifted until after the winning channel has cleared up, indicated by the de-assertion of the mentioned acknowledgement. This prevents the loser channel from gaining adder access while the winner channel is still clearing up, hence the BM controller is still processing former.

The above operation has a timing constraint associated with it: when the BM controller acknowledges the winner channel, then the loser channel must be blocked (through the AND gate in Fig. 7.5) in a time smaller than the one taking the winner channel to de-assert its own request, i.e. going through an RS latch in the protocol converter, as well as another AND gate; this constraint in practice is easily satisfied.

Mid-level controller

The mid-level control unit observes the low-level counts and extracts higher-level information about entire entering and exiting *regions* in the first delay segment, where a region is a continuous run of samples with similar spacing. There are two types of regions, sparse (i.e. low traffic) and dense (i.e. high traffic). The left and right input streams are processed independently.

The mid level control (MC) consists of two asynchronous BM controllers, MLC and MRC as shown in Fig. 7.3. Each controller has a 4-phase input data channel coming from the corresponding

low-level controller, which represents the local sample count. Each communicates to the top level through its 4-phase output channel. The MLC identifies whenever a dense region enters the left sub-segment. The MRC, which has greater functionality, identifies whenever a dense region enters or exits the right sub-segment.

The MLC identifies the start of each high-traffic region by observing the fingerprint count sequence of the LLC, 0, 1, 2. Steady-state count patterns produced within low-traffic (0, 1, 0, 1, ...) or high-traffic (2, 1, 2, 1, ...) regions have no effect.

The MRC identifies and reports three distinct traffic events to the top level. First, it flags when a dense region exits the right sub-segment. The region-entrance from the MLC and the region-exit from the MRC will be used by the top level to produce a current total count of dense regions inside segment 1. The two additional events reported by the MRC are used for a power-optimization in the reconfiguration from coarse-grain to fine-grain mode. In particular, instead of reconfiguring the delay line, starting from segment 2, immediately when the first high-traffic region is flagged to enter segment 1, the controller *waits* until this region is nearly ready to exit segment 1, to prolong the duration of the coarse-grain mode as much as possible and maximize overall power savings. The entrance of a region in the right sub-segment, along with a special extra flag for a safe window (called $SLOT_{FINE-GRAIN}$), are the additional events sent to the top level, which in turn will identify and apply safe reconfiguration.

Top-level controller and reconfiguration policy

The top-level control unit counts the *total number of high-traffic regions* in the entire segment 1 and issues the reconfiguration control signal to the rest of the delay line.

Fig. 7.6 shows the structure and interfaces of the top level. This unit is connected to the two

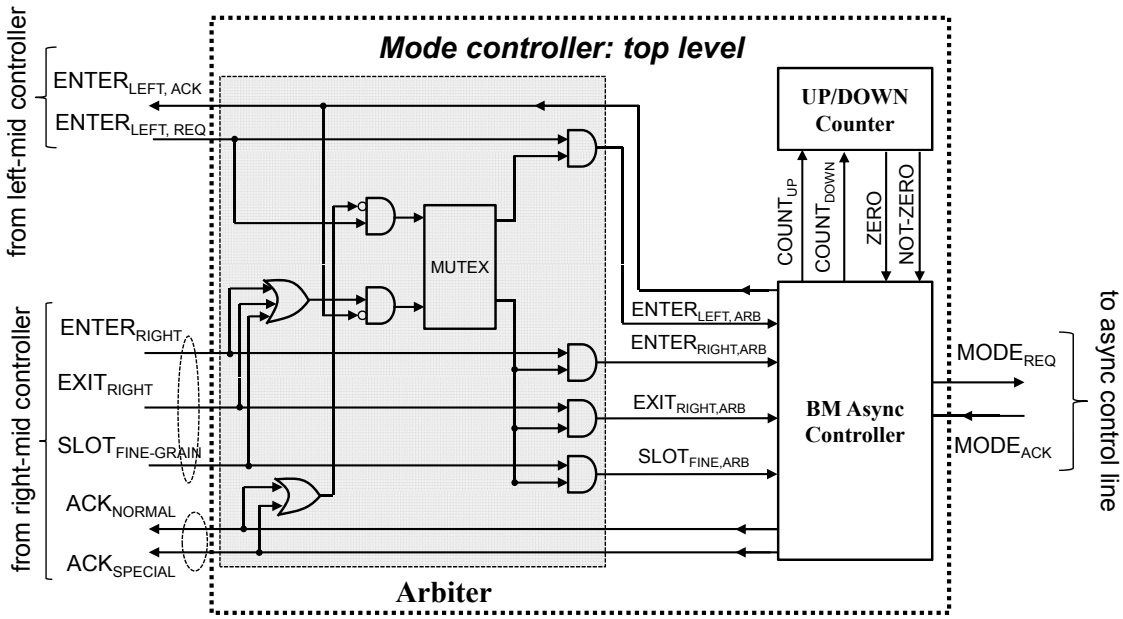

Figure 7.6: Mode controller top-level structure and interfaces for original bi-modal design.

mid-level controllers and has a 2-phase output channel to the async control line. It consists of 3 blocks. An arbiter serializes the two input channels, to allow the asynchronous BM controller to process either left or right input information, one at a time. The BM controller in turn issues increment or decrement operations to the UP/DOWN counter, which maintains the total region count, as well as reconfiguration control to the asynchronous control line. The UP/DOWN counter keeps count of high-traffic regions and reports to the BM controller if it is exactly zero or not. All interfaces internal to the top level are 4-phase.

Simulation Example: Sparse to Dense Traffic. A simple simulation of mode reconfiguration from sparse traffic (i.e. coarse-grain) to dense traffic (i.e. fine-grain) is now illustrated. Initially, segment 1 has no dense regions, so the top-level count is 0. Traffic in segment 2 is also sparse, so any samples inside segment 2 are spaced wide enough to never occupy neighboring cells. This feature is exploited in the design strategy. At some point, a high-traffic region enters the left sub-segment. This information is communicated from LLC to MLC and then to top level, resulting

in the overall dense region count incremented to 1. While reconfiguration could be applied at this point, it is stalled to prolong the coarse-grain operation until a mode change is absolutely required.

The dense region finally enters the right sub-segment, almost ready to enter segment 2. Its arrival is communicated from mid level (MRC) to top level; the latter now requests a safe reconfiguration window from the MRC, by asserting a special acknowledgment signal called $ACK_{SPECIAL}$, initiating the sparse-to-dense mode change. As part of the design strategy, the MRC waits until the first dense sample exits segment 1 and enters segment 2, which it communicates to the top level by asserting its special $SLOT_{FINE-GRAIN}$ signal, thus offering a clean time stamp which can serve as the requested window. At this point, it is *safe* for the top level to reconfigure the delay line through its $MODE$ output channel, as long as reconfiguration is immediately applied *in front of* this new sample in segment 2.

The hand-shaking for the sequence of events described here is variation of the 4-phase protocol, worth mentioning here. When the entrance of the dense region to the right sub-segment is reported from the MRC to the top level (as an active-high $ENTER_{RIGHT}$), the top responds with an active-high $ACK_{SPECIAL}$.The MRC de-asserts its request but the acknowledgement from the top is not de-asserted. When finally $SLOT_{FINE-GRAIN}$ is reported from the MRC to the top level, the latter de-asserts $ACK_{SPECIAL}$ to complete the second transaction. This protocol has little effect on the operation of the controllers; its only effect is saving some states in the specification of the two controllers, making them slightly easier to implement.

As soon as the first cell of segment 2 receives a new sample, and given the sparse traffic inside segment 2, it is certain that cell 3 of segment 2 is either empty, or that it is already safely occupied. No new sample is or will be entering cell 3 for a sufficient time. Reconfiguration control, issued to the third cell of segment 2 as shown in Fig. 7.7, will not be competing with a sample entering

cell 3. Given the design of the asynchronous control line, utilizing back-pressure from the delay line as discussed further below, it is guaranteed that reconfiguration does not overwrite the sample stream. Finally, the first two delay cells of segment 2 are made non-reconfigurable and operate in dense mode, in order to always support high traffic in this scenario.

Simulation Example: Dense to Sparse Traffic. The next simulation, from dense traffic to sparse traffic, is simpler. At some point, the first segment has only one remaining dense region, so the top-level dense region count is 1. When a new sparse-traffic region enters the left sub-segment, the MLC does not communicate any change to the top level, since it only identifies new dense regions. This sparse region finally reaches and enters the right sub-segment. No further information is communicated by mid level (MRC) to the top level. Finally, the dense region *exits* the first delay segment, which is reported from low (LRC) to mid (MRC) to top level. The top level updates its global count of dense regions back to 0, and the immediately sends a reconfiguration to the first cell of delay segment 2. In contrast to the previous simulation, this signal is applied *behind* the exiting dense sample in the second segment.

7.4.3 Asynchronous control line

The control line is an asynchronous micro-pipeline which serially passes mode control issued by the mode controller, to the delay line. Control moves down different paths to guarantee safe mode changes and at a variable rate.

The asynchronous control line's structure is shown in Fig. 7.7. There are four components: a *steady-state control cell,* a *fork module,* a *merge module* and a *protocol converter.* The steady-state cell propagates reconfiguration control in parallel with the delay line, and applies it, with tight

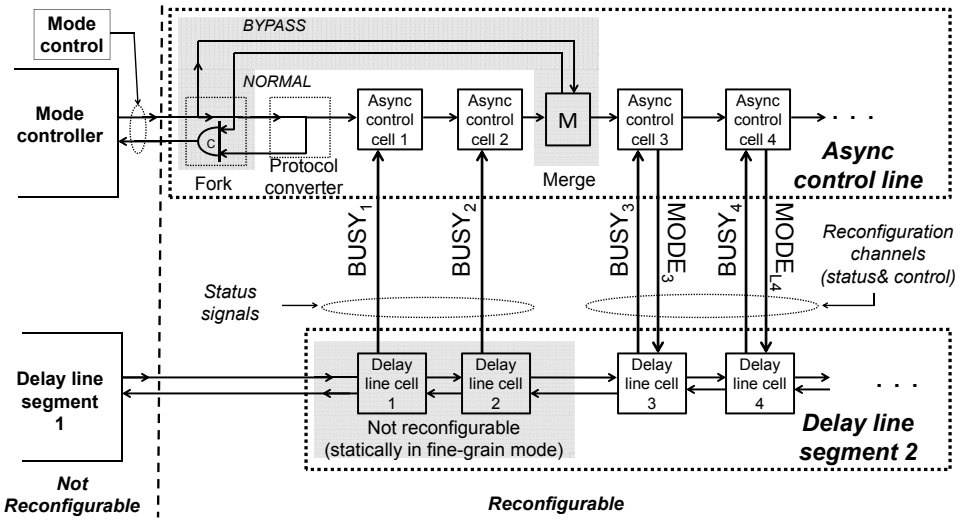

Figure 7.7: Asynchronous control line structure.

synchronization, in a serial fashion, proceeding from left to right through the delay line. It uses a 2-phase protocol with no acknowledgment signal. The fork module splits the reconfiguration control to apply it both at the start of the segment (cell 1) and to a later point (cell 3), to support the asymmetric application of reconfiguration for the two modes, outlined above. The merge module has two inputs and one output, and dynamically alternates between filtering each of its inputs, depending on the current mode. Effectively, since modes strictly alternate between sparse and dense, the corresponding control also alternates between bypassing the first two asynchronous control cells and not, depending on the required control line configuration. The protocol converter converts from 2-phase with acknowledgment (from the mode controller) to 2-phase without acknowledgment (to the steady-state cells).

The steady-state control cell is shown in more detail in Fig. 7.8. It receives an active-high status signal from its corresponding delay cell, called $BUSY$. Due to the carefully-developed reconfiguration strategy, new mode control will arrive at a cell either while the latter is empty and will remain so for a large time, or safely $after$ the latter has been made busy. In the former case,

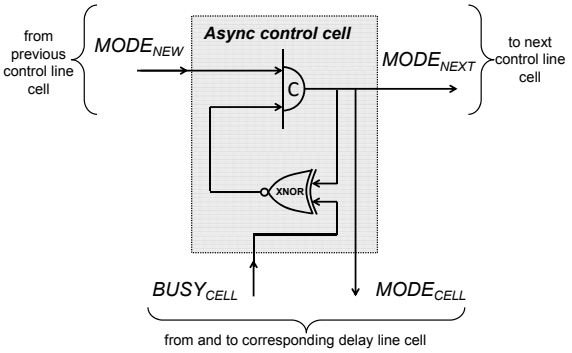

Figure 7.8: Asynchronous control steady-state cell.

the C-element of the control cell is already half-enabled for an input transition, so incoming control is immediately forwarded to the next cell and also applied to reconfigure the current delay cell. In the case of a busy delay cell, the C-element is disabled and control is throttled. After the delay cell has fully completed its operation, its *BUSY* status is de-asserted low and control is immediately applied and forwarded. Control is always passed down safely, never reconfiguring busy delay cells and at a variable rate, depending on the delay line's occupancy. In sum, even though the asynchronous control line is much faster than the delay line which it reconfigures, its control signal is always applied to inactive cells, and its propagation is synchronized to never overtake any earlier sample in the delay line.

7.5 MTBF-type bug in the mode controller

In this section we look into the possibility of a subtle failure in this adaptive delay line system due to the use of *double arbitration* inside the mode controller. The result can be a *mean-time-between-failure* (MTBF) that can cause system misalignment. Before addressing the bug in the next section, we begin with a thorough analysis here.

In more detail, the original design of the mode controller included the danger of characterizing

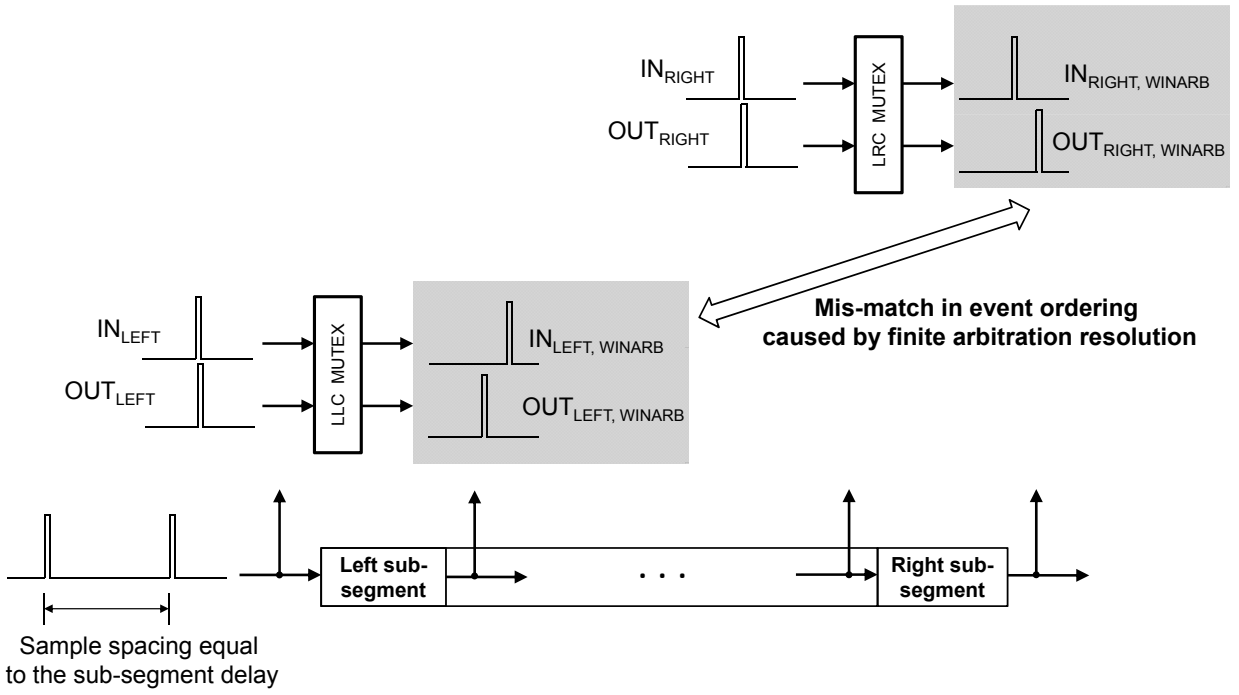

Figure 7.9: Description of the MTBF-type bug of the original approach for the adaptive delay line.

the same traffic in *two different ways*, at two different locations, i.e. the beginning and end of the first delay segment. The problem comes as the result of two things: the double evaluation of arbitration for the same traffic, once at the front and once at the end of the first segment, as well as the finite resolution of arbiters. The effect of this, i.e. the mis-match between the characterization of traffic at these two places, can cause temporary or permanent damage to the system's operation. The operation of the adaptive delay line, as has been made evident from the discussion in the previous section, is based on the fact that that the same traffic is equally characterized by the left and right parts of the first segment; breaking this assumption can lead to problems as we show next.

Fig. 7.9 illustrates this problem. For simplicity, assume that no traffic is present in the delay line initially. The top-level count of HIGH regions is zero and the line operates in coarse-grain mode. Next two samples, spaced approximately as much as the delay of one sub-segment, i.e. two delay

cells in this example, arrive at the line's input. First the earlier sample enters the sub-segment. When the earlier sample is exiting the left sub-segment, the later sample enters. The result is two closely-spaced samples presented to the MUTEX; after some delay, which can be large or in principle even un-bounded, the latter makes a binary decision as to which sample arrived first. In this example, the exiting sample was pronounced as the one which arrived to the MUTEX first. The LLC will process these two samples at the order depicted by the MUTEX. In this example, the LLC will increase its count from 0 (i.e. initial state), to 1 due to the first sample, to 0 since the first sample was pronounced to first exit the left sub-segment, then to 1 due to the later sample. No entering HIGH region will be detected.

After some time, these two samples will arrive at the right sub-segment. In this case too, two closely-spaced samples will be presented to the LRC arbiter. Due to finite resolution, or some path mismatch along the way, the entering sub-segment is pronounced as the one which arrived first, and *the two samples will be processed by the LRC with the opposite order compared to the LLC*. In this case, the LRC count will start from 0, and go monotonically to 1 and then 2, since two entering samples were sensed. This is a *mismatch* to the case at the left. Furthermore, as a result in this case a HIGH region will be detected by the mid controller at the right side, without one first being detected at the left. This is a non-specified scenario, which can cause even the mode controller's BM machines to enter illegal, i.e. non-specified, states.

The fundamental cause of this problem is the *dual sensing* of the same traffic in two different times/locations. It resembles the problem in clock-domain synchronization [24], where two receivers at one clock domain are trying to synchronize to the same transmitter operating at a different clock domain. If the two receivers in this example perform independent synchronization, then the resulting outputs might be different. This performance is measured by the "mean time

between failure" (MTBF), hence the name used to describe this type of bug in the adaptive delay line.

As we will show, the solution to the bug is also similar to the one used for clock-domain synchronization [24]. In the latter, a safe solution includes $first$ the synchronization between the domains at a $singlepoint$, and then distributing the result to the two receivers. This guarantees that the two receivers will get the same result. The solution of the MTBF bug in the adaptive delay line also includes sensing traffic only once, at the left side of the first segment, then re-use the decision made on traffic at the right side when required.

As a final note, it is pointed out that this bug is only exercised for samples spaced by an amount of time very close to the delay of the sub-segment. These cases include traffic which is marginally between HIGH and LOW traffic. In principle, and with a small time error, this traffic can safely be handled by both settings of the delay line. Therefore, even in the cases where the MTBF bug caused the controllers to enter wrong states, the delay line could still handle the traffic stream at that point. For this brief time interval, where traffic remains marginal between high and low, both granularity settings can handle it; if, however, the mode controller does enter a wrong state and high traffic follows, then the delay line will not be able to handle it. Such a case would constitute a system failure.

7.6 Bi-modal delay line with MTBF enhancement

To resolve the previously-mentioned bug, the structure of the mode controller is modified. The overall functionality of the controller, at an abstract level, is the same as before, but the way different pieces of information are extracted and used is different, in order to solve the MTBF-type

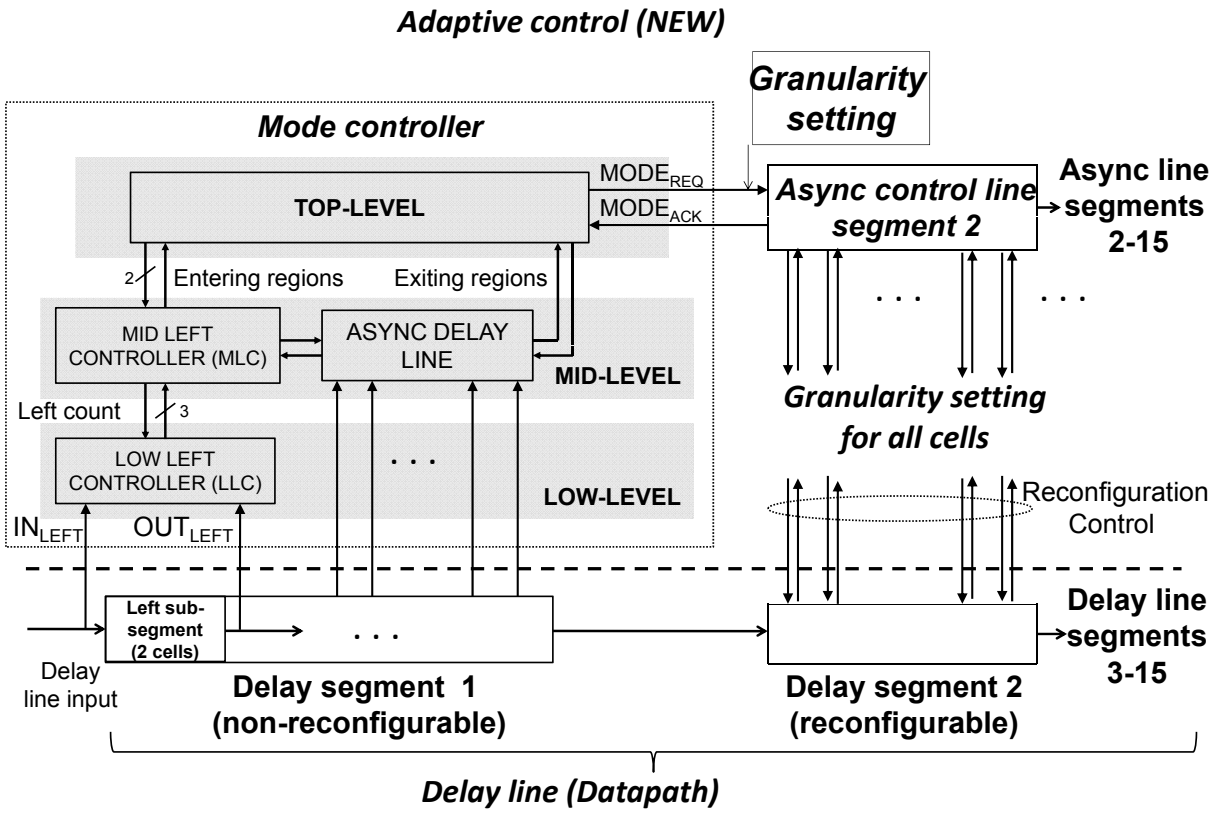

Figure 7.10: Proposed adaptive granularity system: overview of the original approach for bi-modal operation, with MTBF enhancement.

bug.

Fig. 7.10 shows the system overview of the new approach for the bi-modal system. The lowest level now observes the input and output of *only one sub-segment*, at the input of the first delay segment. Based on this information, the mid level identifies high-traffic regions entering or exiting segment 1. While entering regions are detected in real-time, just as in the case for the initial version, exiting regions are detected in advance. However, now, *exiting region information is delayed through an asynchronous delay line and used at a later time*, matching the time when the regions exit segment 1 and move to segment 2. Finally, the top level has a similar functionality, counting the total number of high-traffic regions currently inside the first segment. The asynchronous control line, as well as reconfiguration policy, remain intact, as does the overall structure of the delay

line.

The remainder of this section presents the modified components of the mode controller in this new and final bi-modal approach. Since the delay line and low controller are the same in this case, with the difference that only one low controller is used in this version, the discussion begins with the mid level of the mode controller.

7.6.1 Mode controller

Mid-level controller

The new mid-level now *only* looks at the left-low controller count, and *extracts the same information as the entire mid controller of the previous version*, i.e. entering and exiting high regions.

The mid level controller consists of one asynchronous BM controller (MLC) and an asynchronous delay line, as shown in Fig. 7.10. The BM controller derives region information from the low-level stream, and the asynchronous line is used to delay a subset of the extracted events, acting as a buffer so that certain information can be used later than the time they were issued.

The BM controller MLC has a 4-phase input data channel coming from the corresponding low-level controller, which represents the local sample count, and communicates to the top level through its 4-phase output channel. It can identify both the start and the end of a high region inside the left sub-segment, by observing the fingerprint count sequences of the LLC, 0, 1, 2 and 2, 1, 0 respectively. Steady-state count patterns produced within low-traffic (0, 1, 0, 1, ...) or high-traffic (2, 1, 2, 1, ...) regions have no effect.

Three distinct events are reported by the MLC. One is sent directly the top level, and two are deferred to the right, undergoing a delay from the asynchronous line to be used at a proper later

point. The entrance of a dense region in segment 1 is directly flagged to the top level. The exit of a dense region from the sub-segment is sent to the asynchronous line, travelling down the line synchronized with the exiting region, so that it can be finally sent to the top level when the dense region fully exits segment 1. The last event is used for a power-optimization in the reconfiguration from coarse-grain to fine-grain mode. In particular, instead of reconfiguring the delay line, starting from segment 2 , immediately when the first high-traffic region is flagged to enter segment 1, the controller *waits* until this region is nearly ready to exit segment 1, to prolong the duration of the coarse-grain mode as much as possible and maximize overall power savings. This extra flag for a safe mode change is again detected well in advance and deferred to a later time point, being used finally by the top level to apply safe reconfiguration when high traffic enters segment 2.

The asynchronous delay line is a variant of the asynchronous control line used for granularity control application, presented later in this section. The line is a light-weight pipeline, but provides a fixed latency to the mid level events by operating in tight synchronization with the real-time datapath delay line. Unline the asynchronous control line, it does *not apply any control* to the delay line, but *only receives status information*. The latter is used for synchronization of the two lines, allowing the asynchronous line to have the same delay as the datapath one.

The key difference between the new methodology and previous work (Sec. 7.4 and [37]) is that the system only monitors traffic at *a single point* inside segment 1. In [37] traffic was monitored at both the input and output of the first segment, using 2 low and 2 mid controllers, one set for each side. This original approach had the potential danger for a different characterization of the same traffic at the two locations, caused by potentially different result in the arbiters of the two independent low controllers. Such a differentiation is a direct result of the arbitration process and can not be completely avoided.

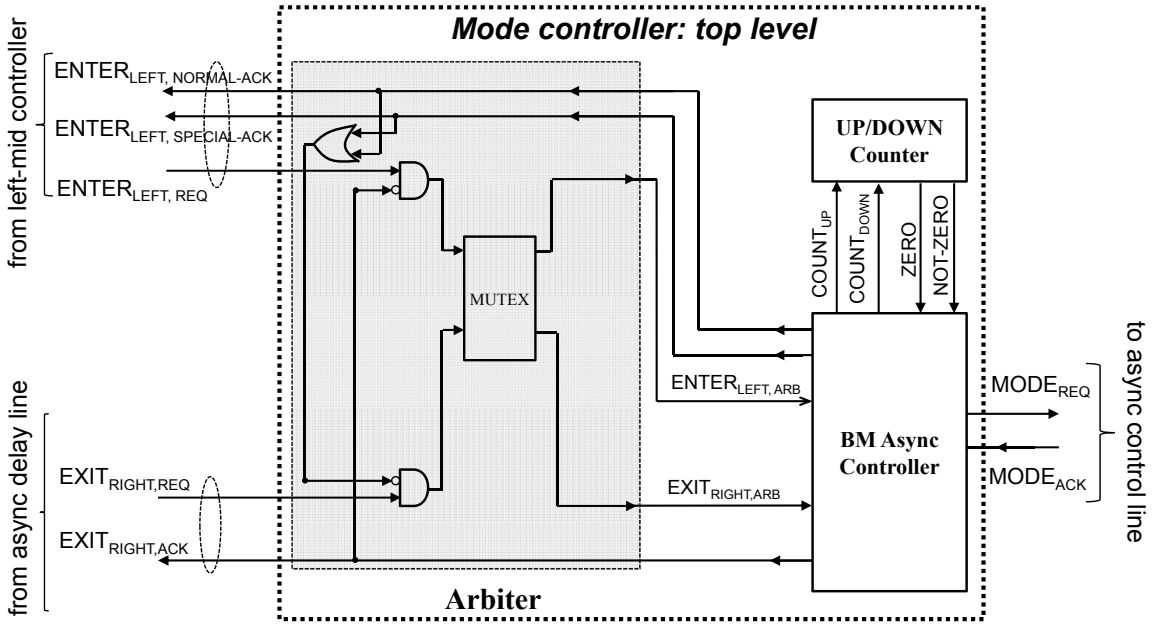

Figure 7.11: Mode controller top-level structure and interfaces for original bi-modal design.

This mis-match can cause temporary or permanent failure to the system's operation. By using a single observation point, a single decision on traffic is made once and no potential for mis-match is present. In the updated methodology all system-level events are extracted in advance, exploring the key property of the delay line that traffic at different points inside the line are time-shifted copies of one another. Therefore traffic at the end of segment 1, which was monitored in the earlier design by another low controller, is identical to that at the beginning of the segment with a proper time delay and certain properties of the traffic are allowed to be extracted in advance. Such events are properly deferred and used at later times; this approach still provides the same system-level behavior for the traffic controller, now clean of any arbitration problems, which can still occur even in the case of a delay line providing perfect time translation.

Top-level controller and reconfiguration policy

The top-level control is a modified version of the top-level controller shown previously in Sec. 7.4. It performs the same functionality, counting the top-level number of high regions in the first segment and issuing reconfiguration control, but it is slightly modified to accommodate the different way it receives information from the mid level.

Fig. 7.11 shows the structure and interfaces of the top level. This unit is connected to the left mid controller and the output of the mid-level asynchronous delay line inside segment 1, and has the same 2-phase output channel to the asynchronous control line. It consists of the same 3 blocks: an arbiter, a BM machine and an UP/DOWN counter. The arbiter and BM machine are slightly modified, compared to the original version. The operation of the controller is explained using the same set of simulations as before.

Simulation Example: Sparse to Dense Traffic. Initially, segment 1 has no dense regions, so the top-level count is 0, and traffic inside segment 2 is sparse. At some point, a high-traffic region enters the left sub-segment. This information is communicated from LLC to MLC and then to top level, resulting in the overall dense region count incremented to 1. While reconfiguration could be applied at this point, it is stalled to prolong the coarse-grain operation until a mode change is absolutely required, i.e. when the first sample of the first dense region is about to enter segment 2.

Instead of monitoring the entrance of the high region in segment 2, this event is detected in advance from the MLC. In particular, the top level responds to the entrance of the first dense region via a special acknowledgment, $ENTER_{LEFT,\ SPECIAL\ ack}$, effectively requesting from the MLC a safe reconfiguration window. The MLC waits until the first dense sample exits the left subsegment. Due to the exact time translation provided by the delay line, this a time-shifted earlier

copy of the entrance of the first dense sample in segment 2. This time slot information is sent to the asynchronous delay line, where it is aligned with the first dense sample and moves down toward segment 2. When the dense sample finally exits segment 1 and moves to segment 2, the time slot is sent to the top controller and used for reconfiguration to dense mode.

At this point, it is *safe* for the top level to reconfigure the delay line through its *MODE* output channel, as long as reconfiguration is immediately applied *in front of* this new sample in segment 2. As soon as the first cell of segment 2 receives a new sample, and given the sparse traffic inside segment 2, it is certain that cell 3 of segment 2 is either empty, or that it is already safely occupied. No new sample is or will be entering cell 3 for a sufficient time. Reconfiguration control, issued to the third cell of segment 2 as shown in Fig. 7.7, will not be competing with a sample entering cell 3. Given the design of the asynchronous control line, utilizing back-pressure from the delay line as discussed further below, it is guaranteed that reconfiguration does not overwrite the sample stream. Finally, the first two delay cells of segment 2 are made non-reconfigurable and operate in dense mode, in order to always support high traffic in this scenario.

Simulation Example: Dense to Sparse Traffic. At some point, the first segment has only one remaining dense region, so the top-level dense region count is 1. The time at which this dense region had exited the left sub-segment was identified by the MLC and is travelling down the asynchronous delay line, in tight synchronization with the last sample of the dense region. No new dense regions enter segment 1 from the left until this dense region exits segment 1. Finally, the dense region *exits* the first delay segment, allowing the "exit" event travelling alongside the region in the asynchronous line to move to the top control level. The top level updates its global count of dense regions back to 0, and the immediately sends a reconfiguration to the first cell of delay segment 2. In contrast to the previous simulation, this signal is applied *behind* the exiting dense

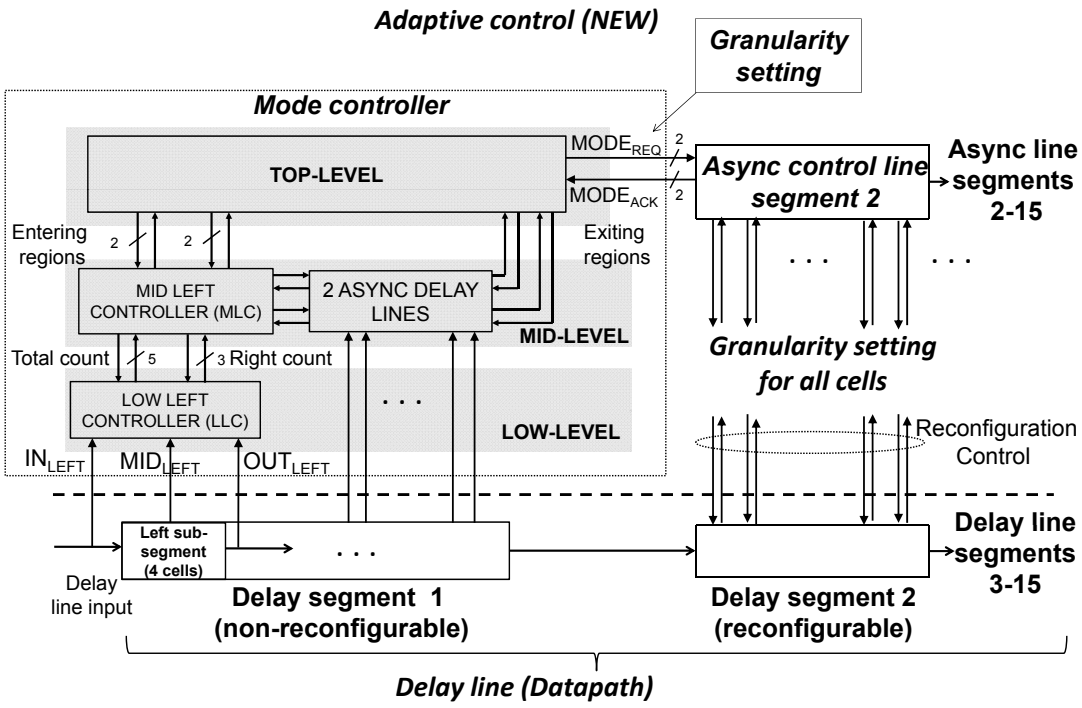

Figure 7.12: Proposed system with 3-mode traffic detection and granularity: overview.

sample in the second segment.

This concludes the presentation of the bi-modal adaptive delay line, with the enhancement for solving the MTBF-type problem. The asynchronous control line is identical in this version. The remaining part of the design methodology is the extension to multiple configuration modes, which follows.

7.7 Methodology extension for multiple granularity settings: a tri-modal delay line example

This section shows the methodology to extend an adaptive-granularity delay line to support multiple configuration modes. By using additional configuration modes, two major benefits are obtained: larger power savings and smoother transition between configurations. A multi-modal adap-

tive delay line can be easily extended from the bi-modal case described in detail earlier. In particular, a system which detects and uses more traffic modes can be built using multiple instantiations of the bi-modal system, and properly combining the individual results.

As a concrete example, a tri-modal adaptive delay line system is presented. A system overview is shown in Fig. 7.12. As in the bi-modal case, traffic is monitored in the first delay segment by the mode controller, however now observing three states. In addition, the remaining segments are correspondingly set to one of 3 granularity settings through an asynchronous control line.

Instead of using a monolithic approach for a 3-way traffic classification, two different and independent 2-way classifications are performed on incoming traffic. This results to each of the 3 levels of the new mode controller consisting of 2 independent parts, one for each bi-modal traffic decision. Ultimately traffic is compared to two thresholds, one half and one quarter of the maximum rate in this example, and depending on the 2 results, a final decision on traffic is made between "high", "medium" and "low". The delay line then operates in full-, half- or quarter-granularity setting respectively. This section presents how each of the tri-modal system's key components is extended from the bi-modal design.

7.7.1 Delay line

The delay line is a modified version of the bi-modal line, supporting three distinct operation modes. Delay cells are now ordered into groups of four, i.e. into *delay cell quads*, and each quad is built using three distinct types of adaptive delay cells. The interface and protocol, i.e. hand-shaking, of the cells is identical to the bi-modal case, and is not presented again.

Fig. 7.13 shows the three types of delay cells in each cell quad. All are similar extensions of

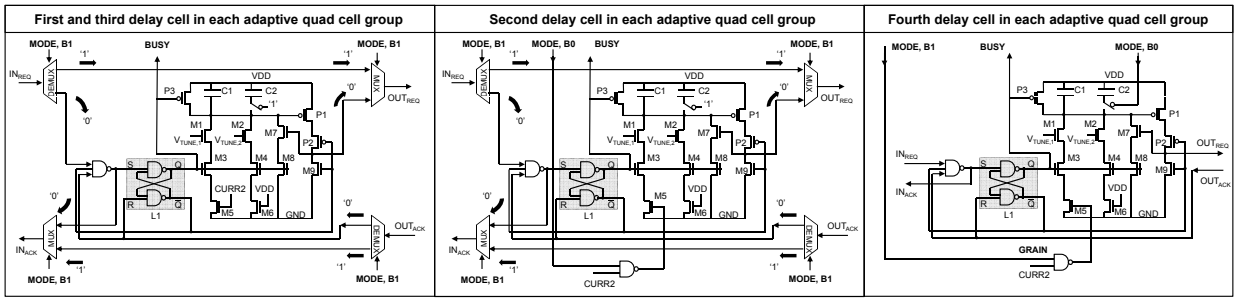

Figure 7.13: Adaptive delay cells for the 3-mode adaptive delay line.

the baseline cell of Fig. 7.4, but now receive 2 control bits $MODE,B1$ and $MODE,B0$ to control their operation. The first type of adaptive cell is identical to the "even" bi-modal cell. It is used twice in each quad, in the first and third position, and can either operate providing a single unit of delay or be bypassed. The second adaptive cell can either give a single delay unit, a double delay by only using one current source instead of two, and also zero delay by being fully bypassed using MUXes and DEMUXes. The final cell, used in the fourth position in each quad, can give a single delay, a double delay, as well as a quadruple delay by using the combination of only one current source and two charging capacitors.

Each quad cell group can thus operate in three different granularity modes to safely handle different traffic types. During very low traffic the quad can be set to its coarser mode ($MODE,B1 = 1$, $MODE,B0 = 1$), where the first three cells are bypassed and the fourth operates with $4X$ the delay. In the medium granularity mode ($MODE,B1 = 1$, $MODE,B0 = 0$), the first and third cells are bypassed and the remaining have a $2X$ delay. During high traffic the finest-grain mode must be used ($MODE,B1 = 0$, $MODE,B0 = 0$), where all cells are used with a single-unit delay. As in the bi-modal case, the overall group delay is kept constant for all settings, but the group's energy consumption is decreased during low or medium traffic by almost $4X$ and $2X$ respectively. Each setting is designed to safely support one type of traffic with the minimum energy drain.

7.7.2 Mode controller

The mode controller performs the 3-way classification of the line's input traffic in the first segment. It consists of *two bi-modal controller instantiations*, therefore its structure is de-composed into two independent parts. At an abstract level, two different and independent operations are first performed, by making two binary decisions on traffic. The latter is characterized as either "high" or "not-high", and as either "low" or "not-low". By combining the two binary decisions, a final classification is made, characterizing the traffic in segment 1 as "high", "medium" or "low".

Low-level controller

The low controller level monitors the incoming traffic at the left of delay segment 1. This time traffic is observed inside a quad group of delay cells. The low controller has 3 input channels, coming from the input and output of the quad group, as well as the mid-point of the group. Two output channels are used to communicate to the two mid controller parts. The low controller reports two distinct count of samples: for both the entire group, with possible values 0, 1, 2, 3 and 4, as well as for the right half of the group with possible values 0, 1 and 2.

The low controller is shown in Fig. 7.14 to consist of 3 parts. A protocol converter and 3-way arbiter are direct extensions of the ones used in the bi-modal case. Two asynchronous controllers are now used to keep and report the two desired counts. Samples entering/exiting the group will cause an increment/decrement operation to the total sample count. The count of samples in the right half group will be also be decremented from samples exiting the group, but will be incremented from samples at the group mid-point, which effectively move from the left to the right half of the group.

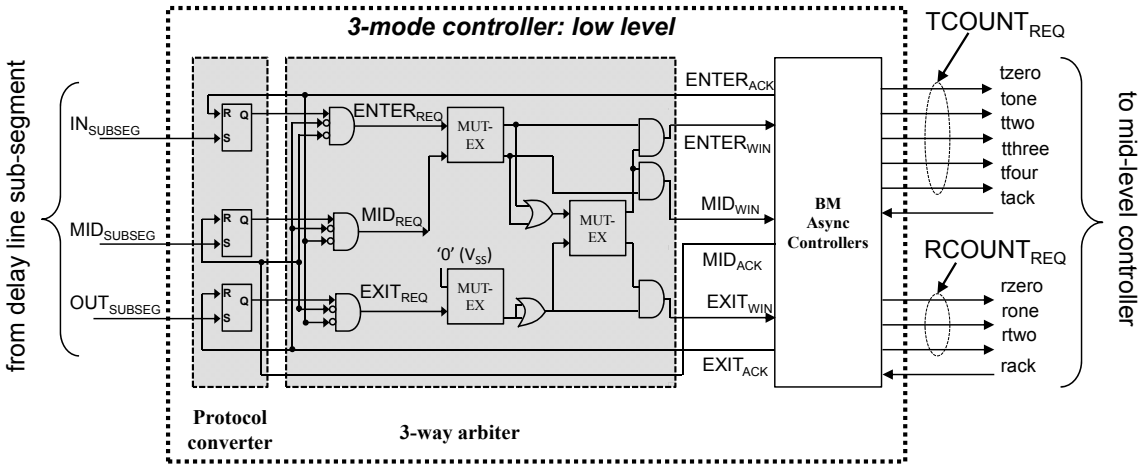

Figure 7.14: Mode controller low-level structure for 3-mode traffic detection.

The two derived sample counts can be used toward local traffic classifications. The way the right count is used is identical to the bi-modal case, as presented previously. The total count can be used in a similar fashion to discriminate between low-traffic samples, spaced by 4 or more time units, and samples with smaller spacing. The latter can either be high- or medium- traffic samples.

Fig. 7.15 shows how the two asynchronous BM controllers are combined. The sample information, i.e. sample entering, exiting or in the middle of the quad cell group, is forked into both controllers. The acknowledgements from both controllers are then combined using C-elements, effectively synchronizing the two controllers. Only when both controllers are done will the combined acknowledgement be forwarded to the left environment. This ensures the safe operation of the controllers, by avoiding the danger of completing the hand-shake on one controller before the other has completed processing. An example of such a case is the following: the $ENTER_{WIN}$ event is sent to both controllers, only the "top" controller uses it to update its count and transmit that to the mid level, while the "right" controller quickly acknowledges at its input channel. Releasing the $ENTER_{WIN}$ channel at this point would compromise the operation of the top controller, which is still processing. By waiting until the top controller acknowledges, the two controllers are effec-

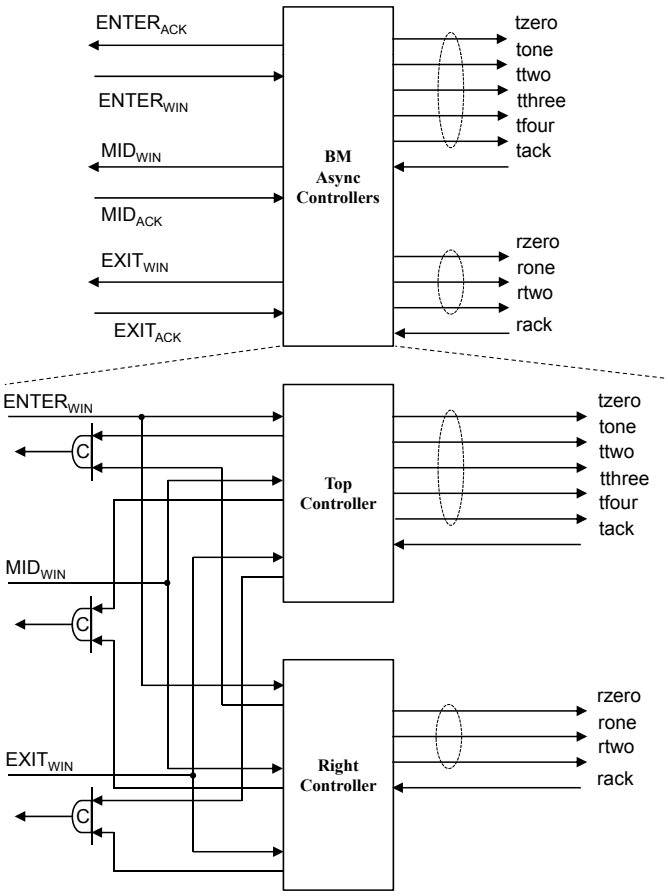

Figure 7.15: Break-down of low level BM controllers, showing the synchronization between the two sub-controllers.

tively synchronized and the transaction at the input channel of the controllers is completed when it is guaranteed that both of them are safely done.

Role of the arbiter: The arbiter protects the BM controllers, as in the bi-modal case, ensuring that only one sample is processed at a time. This design is a direct extension of the bi-modal one, so the same properties and timing constraints apply here too. An additional timing constraint is also applicable here: both MUTEX elements associated with one sample must safely clear up (i.e. de-assert their outputs) once an input channel is blocked, before the winner channel of the arbiter completes its transaction. This ensures that new samples are presented to the BM controller safely after the current sample is done being processed. Intermediate MUTEX outputs not being fully

de-asserted for the losing channels present the danger of new transactions to the BM controllers being attempted by the arbiter before the current transaction has been completed. As a final note, the bottom-left MUTEX in Fig. 7.14 was added for symmetry, i.e. to equalize the input-output path delay for all three inputs.

Mid-level controller

The mid level controller processes the low level information. Similarly to the bi-modal case, the low controller outputs are used to extract higher-level information on traffic *regions*.

There are two independent mid controllers, each processing one low count and identifying one type of region. Each mid controller consists of a BM controller and an asynchronous delay line identical to the bi-modal case. The *region* information derived by the mid level is either sent immediately to the top level or is delayed through the asynchronous delay line, effectively sent to the top after a time delay. The first mid controller processes the right low count; it is identical to the bi-modal mid controller and identifies the high-traffic regions entering and exiting segment 1.

The second mid controller is an extended version of the first controller, processing the total low count. It identifies regions of samples spaced by less than 4 delay cells; since this case corresponds to both high and medium traffic, these regions will be named "medium-high" regions. Entrance and exit of these regions are identified by the fingerprint count sequences 0, 1, 2 and 2, 1, 0 respectively, much like the bi-modal case. Steady-state input patterns of 0, 1, 0, 1, ... during low traffic, or continuous runs of counts larger or equal to 1 during medium or high traffic do not lead to any outputs. Entering region information is sent to the top controller, and both the exiting regions and the special-case reconfiguration slot for each region type, as described for the bi-modal case, is sent to the asynchronous delay lines.

The special-case reconfiguration event is sent to the right, i.e. asynchronous control line when the *first* sample of the *first* medium-high region exits the left sub-segment. Information on which region is the first one is *only* provided by the top level. This, along with the fingerprint sequences for entering and exiting of medium-high regions, are *design invariants* preserved from the bi-modal case, which will also apply in the extensions to even more settings.

In particular, the extension for the special reconfiguration signal has a small effect on the operation of the second mid controller (the one detecting medium-high regions). In the bi-modal case, the special signal was the only option to follow the entrance of the new high region, since the former was detected as a new count of 1 following a count sequence of $0,1,2$. In the case for the second mid controller of the tri-modal case, the exit of the first medium-high sample from the left sub-segment is indicated by a count decrement in the total count of samples in the sub-segment. Given the invariant requirement for the entrance of a medium-high region $(0,1,2)$, this decrement can come from various paths: $0,1,2,1$, or $0,1,2,3,2$, or finally $0,1,2,3,43$. The above slightly increases the complexity of this mid controller.

Each mid controller has its own asynchronous delay line. Both are identical copies of the one used for the bi-modal design, and synchronize to the datapath delay line. Despite their common synchronization, the two lines operate independently.

Top-level controller

The top level keeps a count for both types of monitored traffic regions. It consists of two parts, one for high and one for medium-high traffic regions. Each part keeps an independent count and, based on that, makes an independent binary decision on traffic. The two binary decisions are finally combined for a 3-way final decision on the incoming traffic.

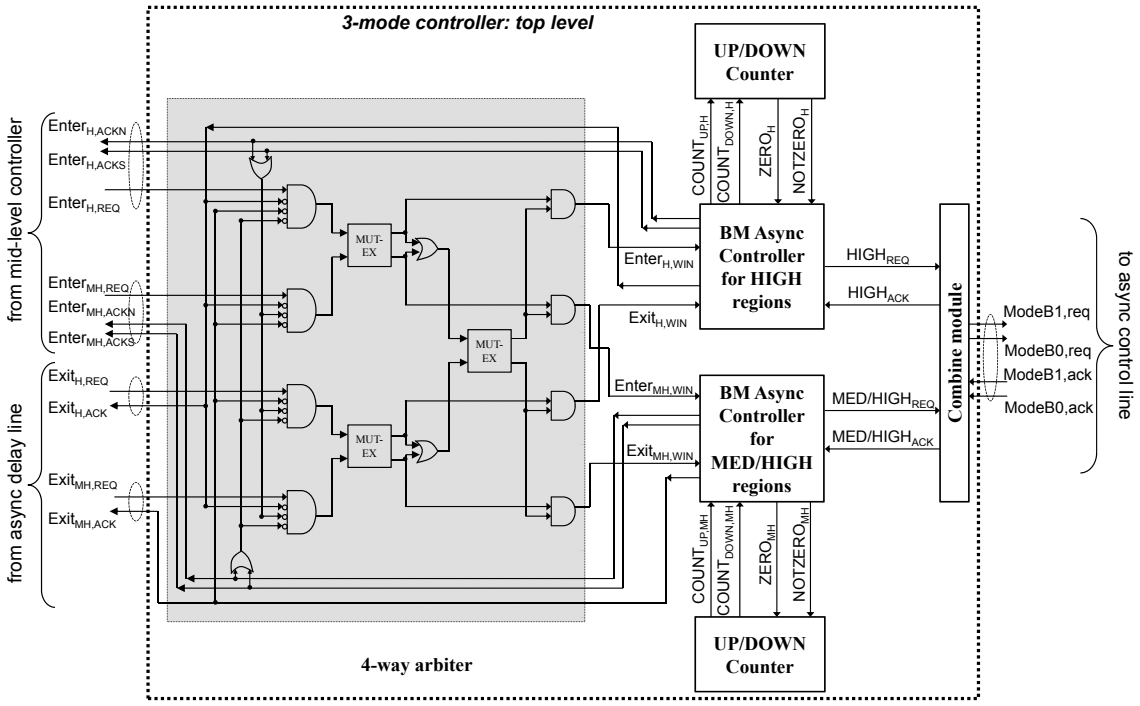

Figure 7.16: Mode controller for 3-mode traffic detection: top-level.

Fig. 7.16 shows the top controller structure. The top level has four input channels, two from the two parts of the mid controller and two from the two asynchronous delay lines. The single output channel of the top controller is the final delay line granularity control, going to the asynchronous control line. A 4-way arbiter serializes the input channels, so that either left or right information is processed from one BM controller, for either high or medium-high regions. Two separate BM controllers and UP-DOWN counters, identical to the ones in the bi-modal design, keep the two counts of high and medium-high regions inside segment 1. The operation and used signaling of each part of the controller individually is the same as the one described in the bi-modal case through detailed simulations. Depending on the count of each type of regions, each BM controller communicates its binary traffic decision to a "combine" module, which combines the two binary decisions and sends the final 3-way granularity mode to the asynchronous control line.

7.7.3 Asynchronous control line

The multi-modal control line passes down the granularity setting to all the delay segments. It is a direct extension of the bi-modal control line for a 2-bit granularity control passed down the line. Similarly to the bi-modal design, control moves serially down the control line at a variable speed, depending on the traffic inside the delay line.

The structure of the control line is similar to Fig. 7.7, with the proper enhancement for a 2-bit control. There are now two protocol converters and fork-merge modules, one for each control bit. The steady-state control cell is also enhanced for a larger bit width. Different control settings move down different paths, either skipping the first 2 cells of the second delay segment or not, ensuring safe reconfiguration for the entire line. Control settings putting the delay line into a granularity mode finer than the previous state always bypass the first 2 control cells, to ensure that reconfiguration is applied ahead of the incoming traffic. Control toward a coarser granularity never bypasses any control cells, effectively being applied behind exiting traffic.

The new steady-state control cell consists of two instantiations of the single-bit version of Fig. 7.8. Both instantiations communicate with a single delay line cell and receive the same status signal $BUSY$. Control moves to the next control cell immediately for the case of a quiescent delay cell, i.e. when $BUSY = 0$, and is temporarily stalled in the case of a busy cell ($BUSY = 1$) until the delay cell completes its operation.

7.8 Simulation results

Sections 7.6 and 7.7 presented two complete design paradigms:a bi-modal system (with fixed MTBF problem) and a tri-modal system. Simulation results are now presented for the two adaptive

systems, in comparison to a baseline non-adaptive system. Simulations indicated significant power savings, up to 70% compared to the baseline version; such benefits can further extend for more configuration modes. The bi-modal asynchronous controllers were also found (through a layout implementation) to be light-weight in terms of both power consumption and total area. Since the tri-modal controller has number of gates comparable to the bi-modal case, the size of the tri-modal controllers is also expected to be small compared to the datapath delay line.

Both baseline and adaptive systems were designed in a 0.13 μm IBM CMOS technology. Each system included a 15-segment delay line, designed for a segment delay of 25 μs, and operated with a nominal sypply of 1 V. The baseline and bi-modal adaptive version were also fully laid out; the tri-modal adaptive version was not laid out due to time constraints. The systems were implemented using custom-made cells, and all routing and placement was custom. The asynchronous adaptive control unit and control line are designed entirely using combinational gates and flip-flops, as well as with a few analog mutual-exclusion (MUTEX) elements. Conservative time-domain simulations were performed on all systems using Cadence Spectre tools. The baseline and bi-modal adaptive systems were simulated using their post-layout extracted designs. The tri-modal design was simulated using the schematic-level designs, and power results were then adjusted to include the effect of layout parasitic capacitances, as observed for other designs in the same technology kit.

The three systems were compared for their average power consumption, calculated as the integral of the total current drained from the supply, divided by the simulation time and multiplied with the supply voltage. The correctness of the designs was verified by extensive simulations of individual blocks and of the entire systems.

Simulation Traces. Figs. 7.17 and 7.18 present detailed snapshots for the bi-modal adaptive system, showing important signals associated with fine-grain and coarse-grain mode changes, re-

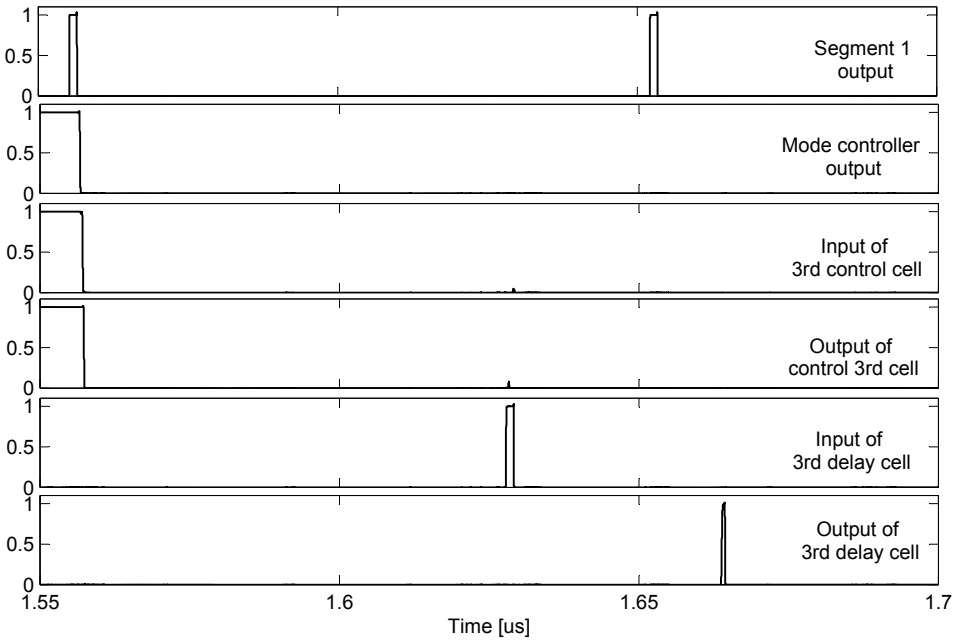

Figure 7.17: Simulation snapshot for bi-modal adaptive delay line: fine-grain mode change.

spectively. They focus on the end of delay segment 1 and the start of delay and control lines for delay segment 2. In particular, the 3rd control and delay cells of the second delay segment, which are the first ones which can be reconfigured.

Fig. 7.17 assumes initially a coarse-grain mode. A new dense region enters segment 1 (not shown), and the first event illustrated is the first sample of this region exiting segment 1 and entering segment 2. The mode controller immediately issues a reconfiguration request to fine-grain mode, encoded as an active-low event in its output to the control line. This event is then forwarded to the third control cell. In this particular simulation, the third delay cell happens to be unoccupied (which is not always the case), so the third cell is immediately reconfigured. The control event is then passed down to the next control cell. If the third delay cell were busy when control arrived at the third control cell, control would be stalled until the data sample had safely exited. In this way, the asynchronous control signal is applied only *behind* the data sample, and never overtakes it. The result is a safe and lightweight synchronization between the asynchronous control line and

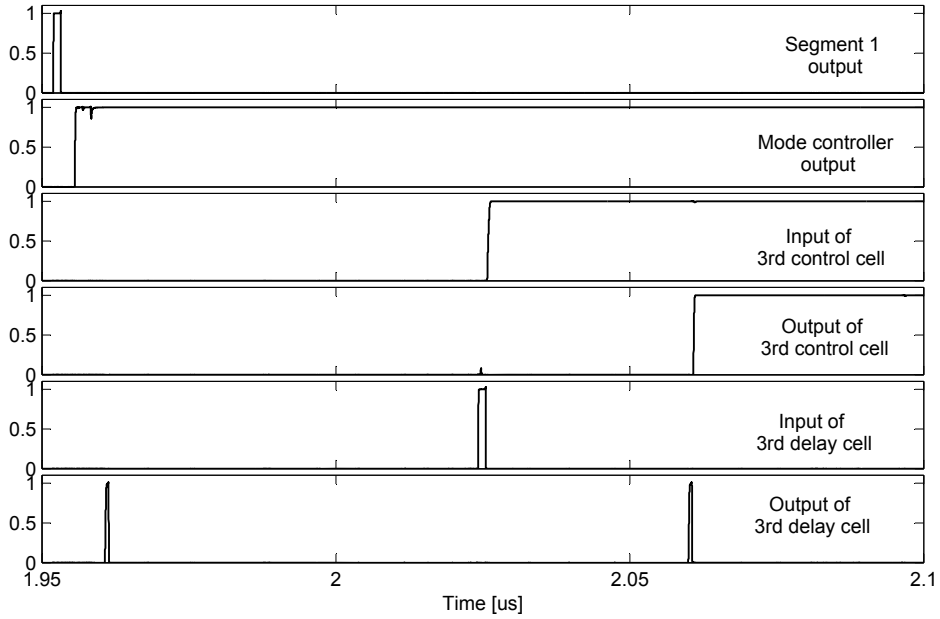

Figure 7.18: Simulation snapshot for bi-modal adaptive delay line: coarse-grain mode change.

the delay line. Much later, a new sample will enter the third delay cell.

Fig. 7.18 assumes initially a fine-grain mode, and the top-level count (of segment 1 dense regions) is 1, i.e. a single dense region inside segment 1. The last sample in the single dense region inside segment 1 is first shown to exit segment 1, signifying that the entire dense region exited. Immediately, the mode controller issues reconfiguration control for coarse-mode, behind the mentioned data sample, to the first control cell. In this case, the control signal always moves down slowly and at the rate of the delay line, being stalled by the last dense sample. The latter first arrives at the third data cell after some time, emptying delay cell 2 and allowing the control to move through control cell 2 to the input of control cell 3. Again, the reconfiguration control signal is stalled due to back-pressure, synchronized to wait for the current sample to exit the cell. The data sample first exits the third delay cell, to allow the control signal to move to the next cell.

Similar signaling and behavior was observed for the case of the tri-modal adaptive delay line. Granularity control was always applied to each delay cell safely and with large time margins com-

pared to the next sample entering the particular delay cell. The variable rate of control application was also observed, with identical properties as the bi-modal design.

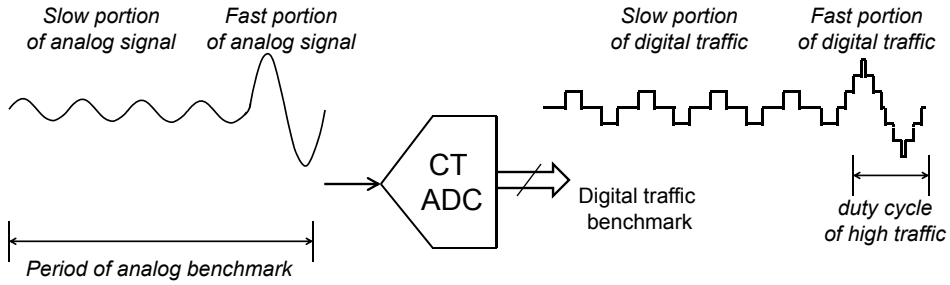

Figure 7.19: Method for synthesizing test benchmarks for delay line evaluation.

Dynamic Power Evaluation. All systems were simulated with various input patterns, to explore the dynamic operation of reconfiguration for different scenarios and to evaluate the resulting system power.

Traffic patterns were synthesized to emulate realistic cases in a CT DSP. Each scenario included a distinct synthetic analog signal of different amplitude and frequency, consisting of a fast and slow portion. The synthetic analog signals were then fed to a model of a CT ADC, which converts them to the digital traffic used for the delay line simulations, as shown in Fig. 7.19. The duty cycle of the high traffic phase was limited to 30% of total traffic, to simulate the realistic quiescent or slow parts of an actual CT DSP environment in typical applications (see [27]- [33] for details). Different frequencies and amplitudes of the analog input result in different average sample rates of the digital traffic. Furthermore, the slow part of *some* benchmarks was slow enough to exercise the very coarse mode in the tri-modal design, while in other cases the slow part of the traffic was interpreted as medium traffic in the tri-modal system. The "duty cycle" of the fast portion of the traffic, together with the average rates for high and low traffic, and the periodicity cycle, together constitute a sufficient set of parameters to characterize each resulting digital pattern.

Input traffic					*Average dynamic power [uW]*										
Pattern	Avg. high-traffic rate [MHz]	Avg. low-traffic rate [MHz]	High-traffic duty cycle [%]	Traffic period [us]	Baseline (non-adaptive)	Bi-modal adaptive					Tri-modal adaptive				
						Delay line	Mode Controller	Async control line	Total	Power reduction [%]	Delay line	Mode Controller	Async control line	Total	Power reduction [%]
1	12	4	20	12	2100	2100	15	0	2115	-0.7	2100	18	0	2118	-0.9
2	15	5	30	15	3190	3190	20	0	3210	-0.6	3190	22	0	3212	-0.7
3	18	4	2	26	1590	880	4	10	894	43.8	480	8	10	498	68.7
4	15	5	5	50	2090	1250	8	5	1263	39.6	1250	16	5	1271	39.2
5	13	4	5	100	1740	980	7	3	990	43.1	560	13	5	578	66.8
6	12	3	5	200	1290	720	6	2	728	43.6	465	10	4	479	62.9
7	11	2	10	300	1180	790	5	1	796	32.5	582	8	2	592	49.8
8	--	2	0	1000	750	405	4	0	409	45.5	210	7	0	217	71.1

Figure 7.20: Average power consumption for adaptive systems: varying input patterns.

Fig. 7.20 lists each pattern's parameters, as well a power comparison between the designed systems and the baseline system, a non-adaptive delay line. The power consumption of the adaptive systems are broken down to their three main components, i.e. the delay line, mode controller and asynchronous control line. When consecutive high-traffic regions are spaced less than the delay of a single delay segment, as in patterns 1 and 2, both adaptive systems never reconfigure their granularity and pessimistically operate in fine-grain mode due to hysteresis. In these cases, no power reduction is offered with respect to the baseline solution; some small penalty is also paid due to the usage of the mode controller as an extra power component. However, a typical CT DSP system will mostly operate in low-traffic environments. In such cases, where high-traffic parts are spaced more than the delay of 1 segment, the adaptive systems do reconfigure to coarse-grain mode during low traffic and segments 2-15 operate with reduced dynamic power.

In the case of the bi-modal system, the resulting dynamic power is almost half compared to the baseline case. The tri-modal system offers even further power reductions, in the case where the traffic during the slow phase is sparse enough to trigger the lowest granularity setting of the tri-modal system. In this case the latter will operate with almost one quarter of the baseline's dynamic power. In the remaining cases the tri-modal system will simply operate in its "medium"

configuration, with power equal to the bi-modal coarse-grain setting. The power savings depend on the relative duration of the high-and-low-traffic periods, and range from 32.5% to 45.5% for the bi-modal case and 39.2% to 71.1% for the tri-modal design. The power consumption of the control blocks is, as expected, *minimal*.

Fig. 7.21 shows a more in-depth power comparison between the baseline and proposed design for one typical CT DSP input pattern, pattern 3, with varying delay line size. The overall trend is that power savings improves with increased number of segments, because the contribution of the non-reconfigurable first segment in the new design becomes smaller. For a line size with a greater number of segments, so that the contribution of segment 1 is small enough, the relative power savings do not increase much with the delay line size. The entire delay line operates mostly in reduced-granularity mode, except for small parts when high traffic appears. The operation setting is different for the two adaptive designs, since the traffic is mostly slow enough so that the tri-modal system enters its coarser-grain setting. An overall power reduction for this input pattern of 43.8% and 68.7% is obtained for a 15-segment delay line, for the bi- and tri- modal designs respectively.

Area Evaluation. Fig. 7.22 compares the total area of the designs. The adaptive system's mode controller occupies extra area, which is 10% / 20% of a single delay segment for the bi- /tri- modal system. Therefore, its area overhead for the case of a 15-segment delay line is less than 1.5% and 3%. The area of the async control line is about 15% and 30% of the delay line for the two designs. However, the control line in both cases is able to completely fit under unoccupied space in the delay line's layout, under wires that held analog and digital programming signals, which was not used in the baseline design. The control line's area *overhead* is, therefore, considered to be effectively zero.

Discussion: Comparative Benefits of Adaptivity. While these power results show significant

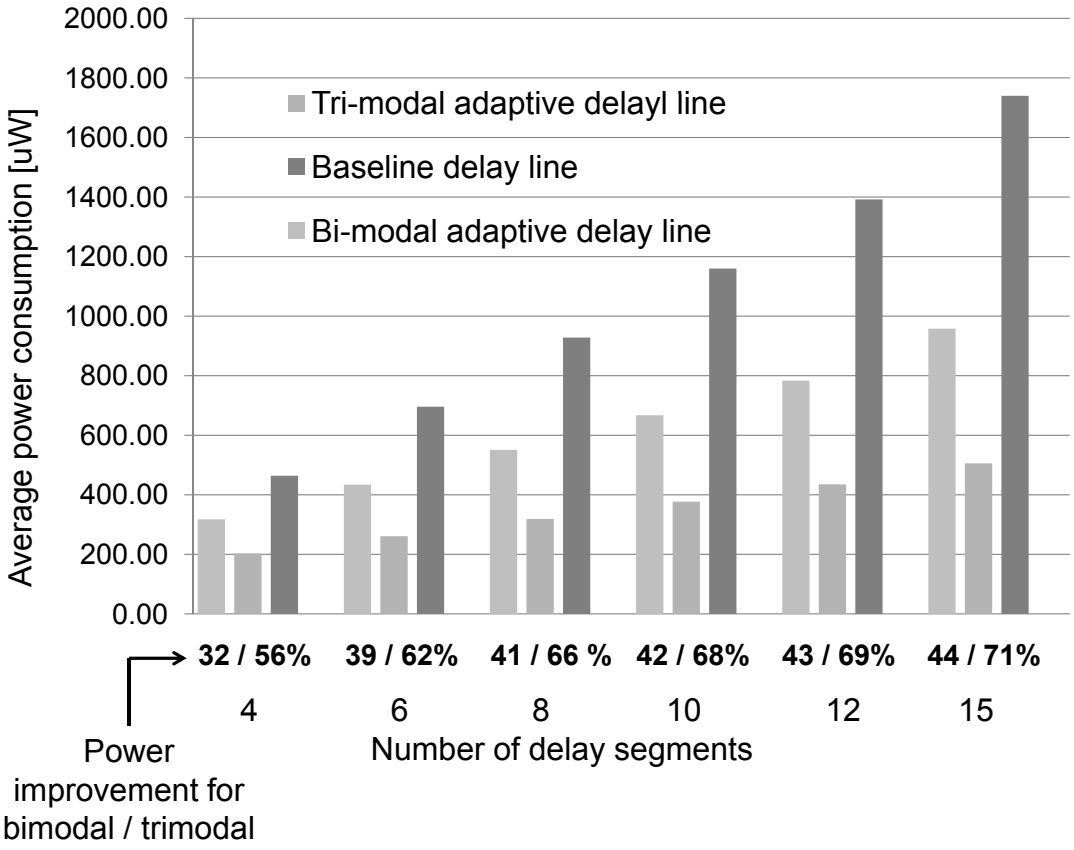

Figure 7.21: Average power consumption: varying delay line size for input pattern 3.

improvements over the baseline design, they are not claimed as the absolute best in the literature. Absolute power results for a digital delay line depends on the target technology. The current technology, 130 nm, is fairly conservative compared to some published designs. We have normalized the simulated dynamic power and compared it to prior designs in 90 nm [33], by using both design kits and performing a detailed evaluation of the dynamic power for the same design block (i.e. a delay cell). The ratio of these dynamic powers (evaluated here as $\frac{2}{3}$) is an estimate of how dynamic power scales for this particular design from 130 to 90nm. After normalizing our design performance by this ratio, the proposed bi-modal adaptive system is estimated to dominate the prior design by 40% for almost all input traffic, while the tri-modal system is estimated to dominate by 70%.

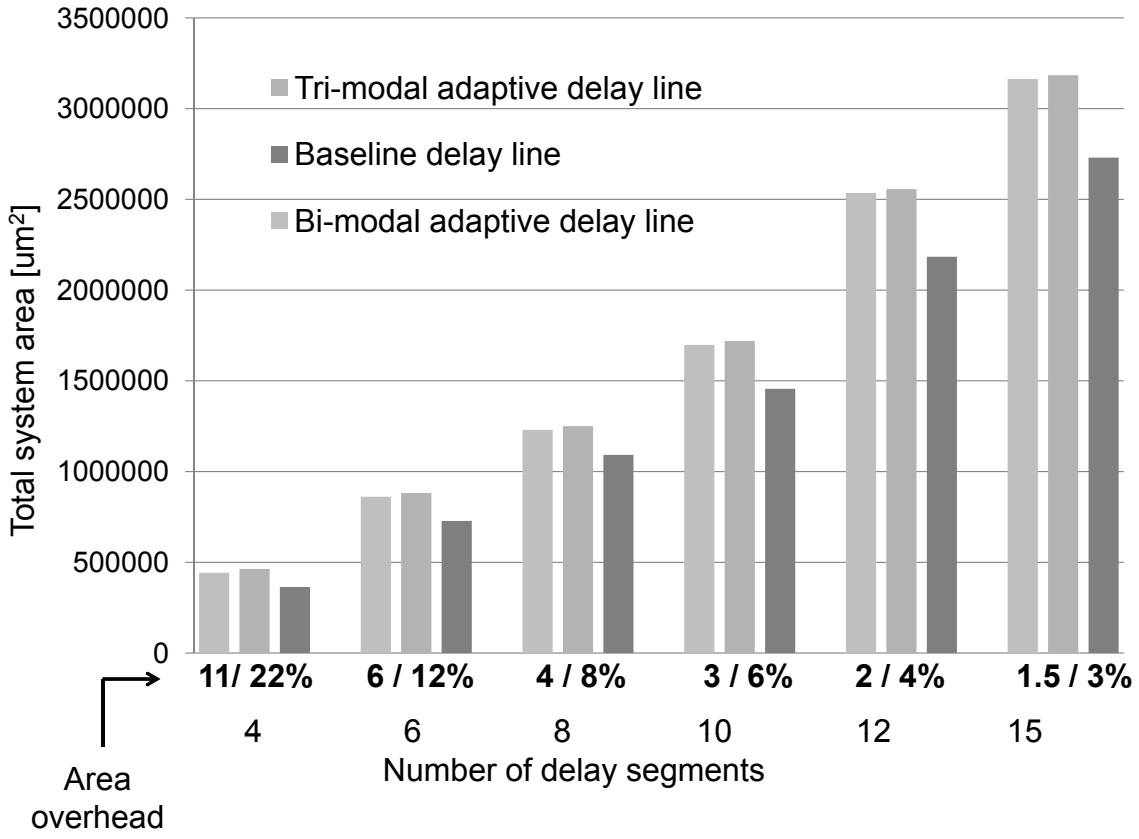

Figure 7.22: Total area: varying delay line size.

Effect of technology scaling. The effect of reducing a CT DSP's dynamic power to almost half or one quarter from the adaptive system is expected to remain almost intact, even in more aggressive technologies with smaller transistors.

Regardless of the technology, we still anticipate that the delay line will dominate the overall dynamic power of a CT DSP system. And, since it will mostly operate the line at halved or quartered granularity, depending on traffic, i.e. bypassing half or 75% of the cells, the adaptive system is therefore expected to offer nearly identical power improvements regardless of the technology.

The additional control components required for the adaptive system are also expected to continue being lightweight, both in terms of area and of dynamic power. The extra control blocks, i.e. the mode controller and asynchronous control line, are only composed of a small number of gates

compared to the CT DSP system. Therefore, their relative area and dynamic power will remain minimal.

7.9 Summary

This chapter introduced a systematic design methodology for dynamically reconfigurable delay lines. Two complete design paradigms were extensively presented, for a delay line operating in two and three granularity modes. We introduced two asynchronous components: the mode controller which can classify traffic in one of many different modes, and the asynchronous control line which can apply granularity control to the delay line safely, quickly and efficiently by using novel tight synchronization to the line.

Simulations of a complete design example for a continuous-time DSP demonstrated indicative power benefits of up to 71%, which can be potentially further enhanced in environments with relaxed traffic statistics. A physical layout demonstrated the minimal overhead of the mode controller and control line.

We introduced two independent design variables in this work, the number and position of traffic thresholds for traffic classifications and the line's operation modes. These constitute a new design space for the power optimization of delay lines; by adjusting the adaptive operation of the line to the statistics of the input traffic in any application, significant power savings can be obtained.

Chapter 8

Companding DSPs for MPEG-Encoded Signals

Companding (*comp*ressing - exp*anding*) DSPs can be considered as one more instance of adaptive DSPs. They effectively adapt their dynamic range to their input, so that the input best fits to the full scale of the adapted dynamic range. As a result, they reduce the effect of quantization at their outputs for small inputs. A companding DSP [42], [124] is known to offer a signal-to-noise-ratio (SNR) which is relatively flat with respect to the input amplitude.

This chapter presents a companding DSP design for MPEG-encoded signals. Such a DSP offers increased SNR while processing low-amplitude signals, compared to a classical DSP. Furthermore, by performing processing before MPEG decoding, the companding DSP minimizes the effect of quantization noise by exploiting the noise masking MPEG property. The net result is a DSP system which combines low-complexity processing, which uses simple arithmetic operations, with audio quality comparable to high-resolution systems (as verified by listening tests).

This chapter begins with an introduction to the MPEG standard, focusing on MPEG1-Layer II

which is the exact audio encoding standard used in this implementation. Following a discussion of motivation for this research, which is low-power MPEG DSPs with high dynamic range, we will present a simplified view of the proposed system. Section 8.3 focuses on the system's implementation, as well as the core of the companding DSP: this includes the modifications to the DSP required to incorporate the companding principle, as well as the extractor of the required envelopes for processing. Section 8.4 presents the results of our implementation, followed by qualitative discussion in Section 8.5. Section 8.6 summarizes this chapter.

8.1 Overview of the MPEG1 standard

The MPEG standard family is the most widely used set of standards for audio and video compression and encoding. They provide perceptually loss-less encoding. The encoded signal does *not* precisely equal the actual signal (audio or video), but listening tests have shown that the difference is not perceivable by the human eye or ear, for audio and video respectively. A very good tutorial on MPEG and its various "layers" can be found in [125]. This chapter deals with audio processing, so the following discussion will be limited to the audio aspect of MPEG signals; however, similar considerations and signal processing methodologies can be applied to video. Finally, this discussion and following implementation is presented around the MPEG1 standard, so "MPEG" hereby refers to "MPEG1".

MPEG encodes audio in three layered formats, layers I-III, where Layer III is the well known "mp3" format. The basic concept around MPEG and its excellent coding properties is the well-known phenomenon of "frequency-domain masking" (or simply "masking"). Listening tests have verified that an audio tone at one frequency makes all frequency components of a certain amplitude

or weaker inaudible, in the vicinity of that tone. This property of the human ear is exploited for efficient audio compression.

MPEG achieves excellent encoding and compression characteristics. It encodes audio in a layered, multi-step approach. Starting with a sampled digital signal in raw (i.e. uncompressed) format, the encoder first passes the input through a "filter-bank" of 32 pre-defined narrowband filters, which split the audio band ($0 - 22$ kHz) into 32 non-overlapping bands. Each filtered version of the signal is then sub-sampled by the same factor, i.e. 32. Within each band then, and for every N samples, the encoder identifies the largest component and filters out all components which are smaller by this by a certain threshold. Essentially, in the presence of large-amplitude samples in one sub-band a coarser resolution is used to quantize the samples.

The sample window N and the threshold for component filtering are set according to the desired encoding compression. Layers I and II are based on the same basic principle, but differ in encoding resolution, with Layer II being more aggressive, using coarser sample resolution and thus achieving better compression. Layer III works on top of Layer II, performing an DCT (direct cosine transform) within each band and then encoding the frequency-domain results, using the same approach to quantize the obtained frequency components. Finally, all samples, time-domain (for layers I and II) or frequency-domain (layer III) are first normalized before being stored: all samples within one frame, i.e. N samples, are first divided by one of 64 pre-defined numbers called "scale factors", so that the maximum sample in each frame is close to 1 in absolute. The information stored for each frame and sub-band are the normalized samples, in floating-point format, as well as a 6-bit number pointing to the used scale factor.

The decoder procedure is complementary to the encoding. The decoder for Layers I and II first de-normalizes the samples, i.e. multiplies the normalized sample with the corresponding scale

factor. Then it up-samples by 32 and filters each sub-band properly, so that all frequency-domain aliases created by the encoder sub-sampling are cancelled out through careful choice of the filter-banks. Then all sub-bands are summed to form the final audio output, alias free and with the original signal recovered. The total noise components, arithmetic and round-off, are at least 95 dB below the signal, as set by the standard, measured to be inaudible to the human ear.

8.1.1 Requirement for low-energy DSPs

Given the wide usage of the MPEG standard, there is much interest in processing MPEG audio, even before the decoding process. Using an internal floating-point arithmetic format, high-fidelity processing of MPEG audio was conventionally performed in floating-point [126]. However many recent applications, mostly portable, cannot afford floating-point arithmetic for audio processing. The reason for this is either the lack of a floating-point unit or the demand to multiplex the latter with other, more demanding applications (like video processing), which puts audio at a lower priority. Finally, the energy consumption of a floating-point unit is considerable; it can quickly drain a battery. Most portable systems, therefore, can not use floating-point arithmetic for audio processing.

Fixed-point arithmetic and processing is used as an alternative to floating-point in many systems. Performing fixed-point processing is much desirable. This is due to both the existence of multiple fixed-point units in most systems, which minimizes the possibility of congestion, as well as the reduced latency and energy consumption of fixed-point operations. MPEG audio processing is an application that will highly benefit from replacing floating with fixed point. However, conventional fixed-point processing, performed after decoding imposes the fundamental limitation of

reduced accuracy (i.e. SNR) for small-amplitude signals. The processing precision, due to arithmetic and quantization noise, is limited in fixed-point operations, so the SNR degrades for smaller signal amplitude. This is the exact opposite from floating-point operation, which has much higher precision for all signal amplitudes.

The remaining chapter presents companding DSPs that perform MPEG processing before decoding and in fixed-point. The DSP shown in this chapter operate on MPEG signals encoded in the sub-band domain, i.e. before MPEG decoding. While these are not the first proposed sub-band processors, previous implementations [127], [128], [129] were limited to floating-point operation. The companding technique increases the output SNR for small amplitudes, by suppressing the quantization noise for the case of small signals. Use of this technique, where separate processing happens in each narrow sub-band, confines the quantization noise to always be close to large signals in the frequency domain. The net result is that large quantization noise only appears close to large signals, and is made inaudible due to frequency masking.

The fidelity of the output of a companding MPEG DSP is comparable to that of floating-point DSPs. Through sample clips that are available on a website [130], it can be verified that a companding MPEG DSP has almost similar performance to a floating-point one, despite the large difference in the two systems' SNR and processing precision.

8.2 Companding DSPs for MPEG audio: simplified view

The companding DSP technique was introduced in [42], and its two versions, syllabic and instantaneous, were presented in [131] and [132], respectively. The technique involves an ADC/DSP/DAC system with adaptive resolution, which has the same overall behavior as a classical and linear-time-

invariant (LTI) ADC/DSP/DAC prototype system. The companding system scales its analog input with a varying factor before the ADC, enlarging small inputs before quantization and effectively *comp*ressing the system's input dynamic range. The system's output is restored to its proper level after the DAC using the same principle of scaling with a proper number, therefore exp*anding* the output dynamic range. To enlarge and restore, companding DSPs use appropriate "envelopes" to divide and multiply the input and output of the system respectively. The DSP core of the companding system is not the same as the prototype's DSP core, but it is a properly modified version so that the companding system's overall response, including the effect of the varying input and output scaling, is the same as the prototype's. The companding DSP core uses similar envelopes for the system's internal states, which are used to enlarge the latter. The resulting DSP core is shown in [42] to be non-LTI.

In this work, we process MPEG audio *prior* to de-normalization by using the syllabic companding DSP technique, as shown in Fig. 8.1. This techniques processes the compressed input in each of the 32 sub-bands, along with corresponding input scale factors used in lieu of the companding envelopes, and yields compressed output, along with corresponding output scale factors. Each sub-band is processed through a separate DSP. The remaining MPEG decoding then continues to provide the filtered signal at its output. The companding technique requires time-domain compressed signals, as well as corresponding scale factors. The highest MPEG layer for which this is the case is MPEG 1-Layer II (MP2), and this is the standard we used in this work. However, the techniques presented in this work can be similarly applied to any standard in which audio is encoded in the form of normalized time-domain subband samples and corresponding time-domain scale factors; we chose MP2 since it is used for many applications, including digital-video-broadcasting (DVB) and DVD players.

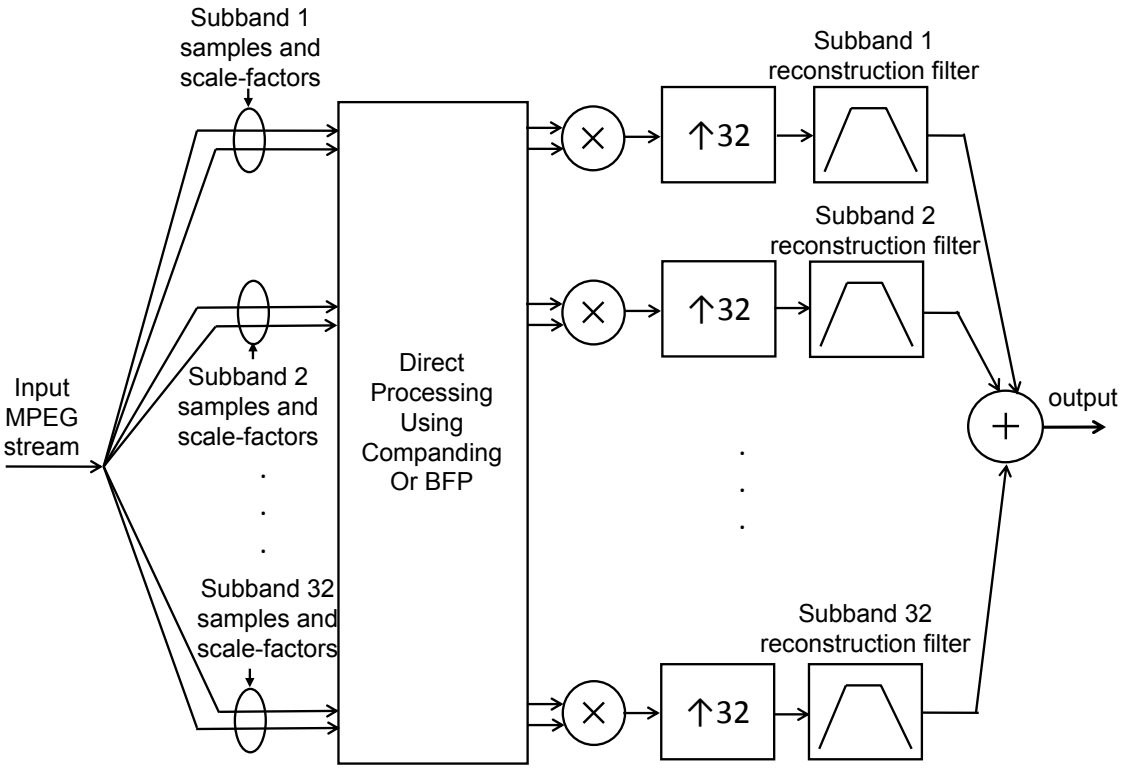

Figure 8.1: Direct processing of MPEG-audio. Subband samples and corresponding scale factors are efficiently processed before denormalization by using syllabic companding processors.

8.3 Companding MPEG processors: detailed description

Each of the 32 companding DSPs operates along the basic companding principle, which is described shortly here. Each companding DSP replaces a prototype LTI DSP, where the latter has a single input and output and K states, and is described described in state space as follows:

$$\begin{aligned} x_i(n+1) &= \alpha_{i1} \cdot x_1(n) + ... + \alpha_{iK} \cdot x_K(n) + b_i \cdot u(n), \quad 1 \leq i \leq K \\ y(n) &= c_1 \cdot x_1(n) + ... + c_K \cdot x_K(n) + d \cdot u(n) \end{aligned} \tag{8.1}$$

, where the constant coefficient matrices $A = \{a_{ij}\}$, $B = \{b_i\}$, $C = \{c_i\}$ and $D = \{d\}$ are KxK, $Kx1$, $1xK$ and $1x1$ respectively. Companding DSPs introduce externally applied envelopes $e_u(n)$, $e_{x_i}(n)$ and $e_y(n)$ for the input, states and output respectively and normalized input, output and states $\hat{u}(n)$, $\hat{y}(n)$, and $\hat{x}_i(n)$, such that:

$$\begin{aligned} \hat{u}(n) &= \frac{u(n)}{e_u(n)} \\ \hat{y}(n) &= \frac{y(n)}{e_y(n)} \\ \hat{x}_i(n) &= \frac{x_i(n)}{e_{x_i}(n)}, \quad 1 \leq i \leq K \end{aligned} \tag{8.2}$$

Therefore, by substituting 8.2 into 8.4, the companding DSP's equations become:

$$\begin{aligned} \hat{x}_i(n+1) &= \alpha_{i1} \cdot \frac{e_{x_1}(n)}{e_{x_i}(n+1)} \cdot \hat{x}_1(n) + ... \alpha_{iK} \cdot \frac{e_{x_K}(n)}{e_{x_i}(n+1)} \cdot \hat{x}_K(n) + b \cdot \frac{e_u(n)}{e_{x_i}(n+1)} \cdot \hat{u}(n), \quad 1 \leq i \leq K \\ \hat{y}(n) &= c_1 \cdot \frac{e_{x_1}(n)}{e_y(n)} \cdot \hat{x}_1(n) + ... + c_K \cdot \frac{e_{x_K}(n)}{e_y(n)} \cdot \hat{x}_K(n) + d \cdot \frac{e_u(n)}{e_y(n)} \cdot \hat{u}(n) \end{aligned} \tag{8.3}$$

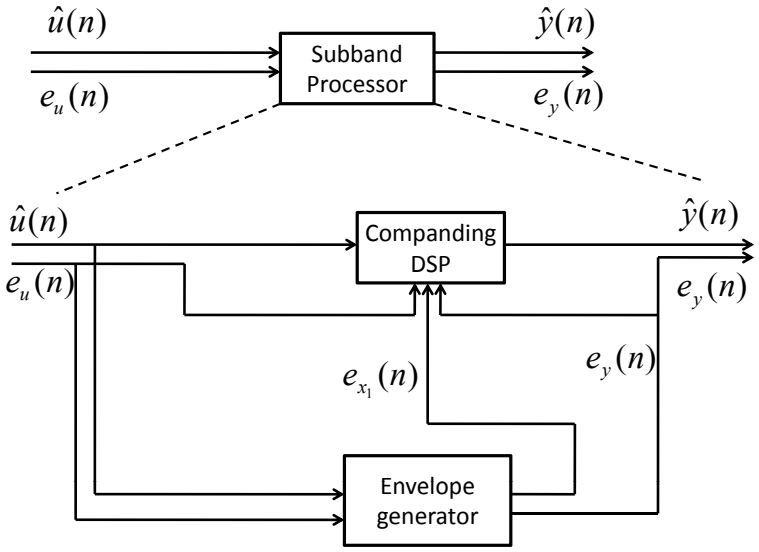

Figure 8.2: A companding subband processor. For this case study, the processor block of Fig. 8.1 is composed of 32 identical copies of this subband processor, with the i^{th} processor taking as input only the i^{th} stream of subband samples and corresponding scale factors.

The effect of companding can qualitatively be explained with a simple example, which assumes an overall all-pass DSP. When the input $u(n)$ to the companding DSP system is small, the resulting input envelope $e_u(n)$ will be small, and so will be the state and output envelopes $e_{x_i}(n)$, $e_y(n)$. The resulting normalized input $\hat{u}(n)$, quantized by the ADC will be large. At the DSP system output, i.e. after the DAC, the normalized (large) output will be scaled with the small output envelope. The result is that both the (large) normalized output and the arithmetic and quantization noise will be suppressed by the same factor for small signals. The resulting SNR is still large in this case. This is in full contrast with classical DSP systems, where the noise is never suppressed, and the SNR is degraded for small inputs.

A sub-band processor is shown in Fig. 8.2. It processes the pair of normalized input and input scale factor for the pair of normalized output and scale factor. It consists of a companding DSP, as described by eqs. 8.3, as well as an appropriate block to generate the required e-controls. Two

different versions are presented for the *e*-control generators.

In the case of MPEG signals, the prototype DSP and sub-band DSPs operate at different sample rates; each DSP for the 32 MPEG bands operates at a rate smaller than that of the prototype DSP by a factor of 32. This is due to the fact that each MPEG band is sub-sampled during MPEG encoding. In [127], the method for deriving the sub-band DSPs from the prototype DSP is thoroughly described. In the special case where the prototype DSP has an identical effect for all the sub-bands, the individual DSPs are themselves identical and are a sub-sampled version of the prototype DSP. A detailed example of such a case is presented now.

8.3.1 Implementation of a companding reverberator

A reverberator is an all-pass system, simulating the "echo" effect. This example is also used in the earlier works on companding [124], [133], [132]. In this example, we use the following state equations:

$$\begin{aligned} x_1(n+1) &= -0.8 \cdot x_L(n) + 0.2 \cdot u(n) \\ x_i(n+1) &= x_{i-1}(n), \quad 2 \leq i \leq L \\ y(n) &= 1.8 \cdot x_L(n) + 0.8 \cdot u(n) \end{aligned} \tag{8.4}$$

where $L = 2048$ and the sampling rate for the input $u(n)$, output $y(n)$, and states $x_i(n)$ of the prototype is $f_S = 44.1$ kHz. Since the reverberator has the same effect on all the MPEG sub-bands, all the sub-band processors are identical and described by the state equations:

$$\begin{aligned}
\hat{x}_1(n+1) &= -0.8 \cdot \frac{e_{x_K}(n)}{e_{x_1}(n+1)} \cdot \hat{x}_K(n) + 0.2 \cdot \frac{e_u(n)}{e_{x_1}(n+1)} \cdot \hat{u}(n) \\
\hat{x}_i(n+1) &= \hat{x}_{i-1}(n), \quad 2 \leq i \leq K \\
\hat{y}(n) &= 1.8 \cdot \frac{e_{x_K}(n)}{e_y(n)} \cdot \hat{x}_K(n) + 0.8 \cdot \frac{e_u(n)}{e_y(n)} \cdot \hat{u}(n)
\end{aligned} \tag{8.5}$$

,with $K = 2048/32 = 64$ and $e_{x_K}(n) = e_{x_1}(n-K+1)$.

The e-controls are constrained to be integer powers of 2, so that the ratios in (8.5) are efficiently implemented as subtractions of (integer) base-2 logarithms, and multiplying by the ratios is efficiently implemented with arithmetic bit-shift. Information about the input envelope for each subband is provided in MPEG-audio in the form of a signal scale-factor. From this, the $e_u(n)$ control signal is generated via a lookup table (LUT). The LUT has a 14-bit input: the 8-bit normalized input sample, concatenated with its corresponding 6-bit scale-factor index. The LUT outputs a 4-bit integer corresponding to the base-2 logarithm of the lowest integer power of 2 greater than the scale-factor, and a new 8-bit compressed subband sample corresponding to this power-of-2 scale factor. The new 8-bit sample is used as $\hat{u}(n)$ in (8.5), while the power-of-2 scale factor is used as $e_u(n)$ in (8.5). As shown in [131], the remaining e-controls should be chosen to correspond, at least roughly, to the envelopes of the corresponding signals in the prototype, in order to maximize the dynamic range of the subband processor, and minimize the quantization distortion. Thus, as seen in Fig. 8.2, we use an "Envelope generator" block to obtain the remaining e-controls required by the companding DSP. In [131], a replica DSP was used to calculate the remaining required e-controls. We could do this here as well, using 32 low-resolution fixed-point implementations of the subband-prototype. However, implementing the replica DSPs adds significant overhead, so

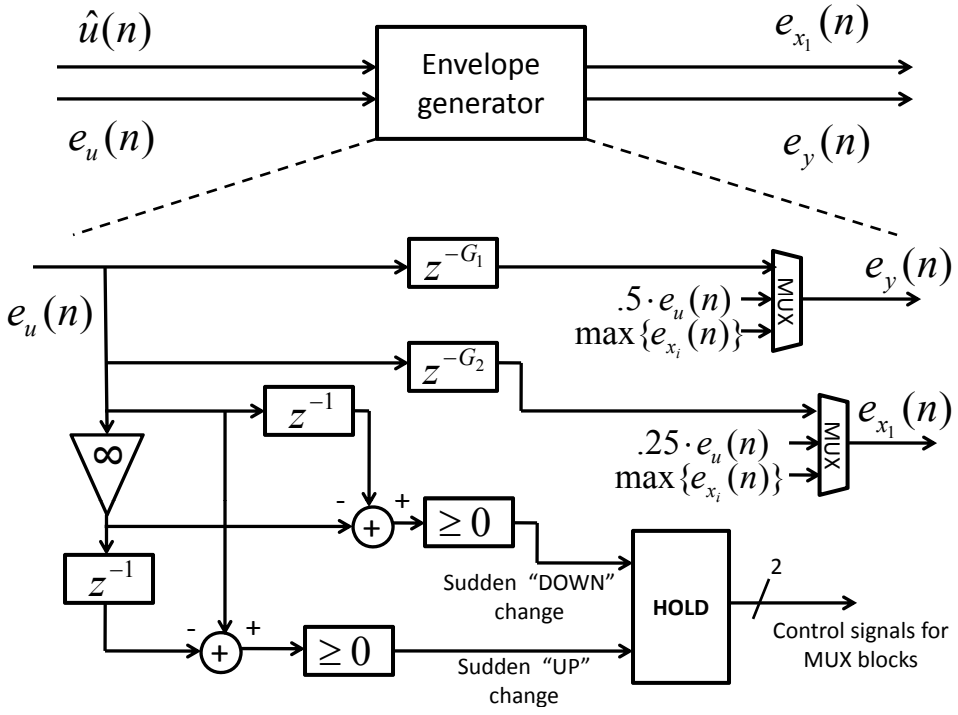

Figure 8.3: Subband processor without a replica DSP for the case of an all-pass reverberator.

we have devised a more efficient technique for estimating the remaining e-controls. Our algorithm, shown in block diagram format in Fig. 8.3, takes advantage of the narrowband nature of the subbands, and its operation principle described in detail in the following.

When a signal $u(n)$, narrowband around a frequency ω_1, is processed with an LTI filter, one can approximate $u(n)$ with a single tone at frequency ω_1, so that the output is roughly $\tilde{y}(n) = A_1 \cdot u(n-n_1)$, where A_1 is the magnitude of the filter's transfer function at frequency ω_1, and n_1 is the group delay of the filter, rounded to the nearest integer, at frequency ω_1. Thus, the envelope of $y(n)$, $e_y(n)$, can be approximated with $A_1 \cdot e_u(n-n_1)$. Similar results hold for the filter states.

The above discussion only applies when there is no sudden change in the input, $u(n)$, since until the system resettles after the sudden change, it cannot be viewed as above. We have determined empirically that abrupt changes in $u(n)$ are indicated by changes of more than a factor of 8 between

consecutive values of $e_u(n)$ in (8.2). When no such change is detected, we can consider the subband signal to be narrowband. For our subband-prototypes, all input-state and input-output transfer functions are normalized such that their maxima are at 0 dB, so $A_1 = 1$. Thus, in (8.5), we can approximate the output envelope of the companding DSP's output, $e_y(n)$, by $e_u(n-G_1)$ and the first state's envelope, $e_{x_1}(n)$, by $e_u(n-G_2)$, where G_1 and G_2 are the corresponding group delays, rounded to the nearest integer.

The magnitude of the transfer function from the subband prototype's input, $u(n)$, to its K^{th} state, $x_K(n)$, ranges from -15 dB to 0 dB in this example. When there have been no recent abrupt input envelope changes, $e_u(n)$ and $e_{x_K}(n)$ can differ by at most one order of magnitude. When there are abrupt input envelope changes, $e_u(n)$ will temporarily be either much larger or much smaller than $e_{x_K}(n)$. In the subband prototypes, given by (8.4), but with L replaced by $K = \frac{L}{32}$, we see that $x_1(n+1)$ and $y(n)$ are both composed of two components: one depending on the input, $u(n)$, and the other on the K^{th} state, $x_K(n)$. When there is an abrupt input envelope change, one or the other component will dominate in the envelopes of $x_1(n+1)$ and $y(n)$, allowing us to use simple approximations for these envelopes. Specifically, for sudden increases in $e_u(n)$, $e_u(n)$ temporarily becomes significantly larger than $e_{x_K}(n)$, so in (8.5), we can approximate $e_y(n)$ as $.8 \cdot e_u(n)$, and $e_{x_1}(n)$ as $.2 \cdot e_u(n)$. Since we always use exact integer powers of 2 for $e_u(n)$, and we also want $e_y(n)$ and $e_{x_1}(n)$ to be exact integer powers of 2, we further approximate $e_y(n)$ as $.5 \cdot e_u(n)$ and $e_{x_1}(n)$ as $.25 \cdot e_u(n)$. This also results in a far simpler implementation, as $e_y(n)$ and $e_{x_1}(n)$ can be easily computed from $e_u(n)$ by simply subtracting 1 or 2, respectively, from the integer power of 2 stored for $e_u(n)$. These assignments must be carried for at least G_1 samples, after which the envelopes can again be estimated via the group delays, until a new abrupt input jump is detected. Similarly, for sudden decreases in $e_u(n)$, both $e_y(n)$ and $e_{x_1}(n)$ can be approximated as $\max\{e_{x_i}(n)\}$ until a

new abrupt input jump is detected.

The above described functionality is shown in Fig. 8.3. Even though only low-resolution fixed-point operations, along with a minimal amount of extra hardware, are used in the implementation described above, its performance will be seen to yield high output SNR over a large input dynamic range, and excellent perceived audio quality.

8.4 Implementation results

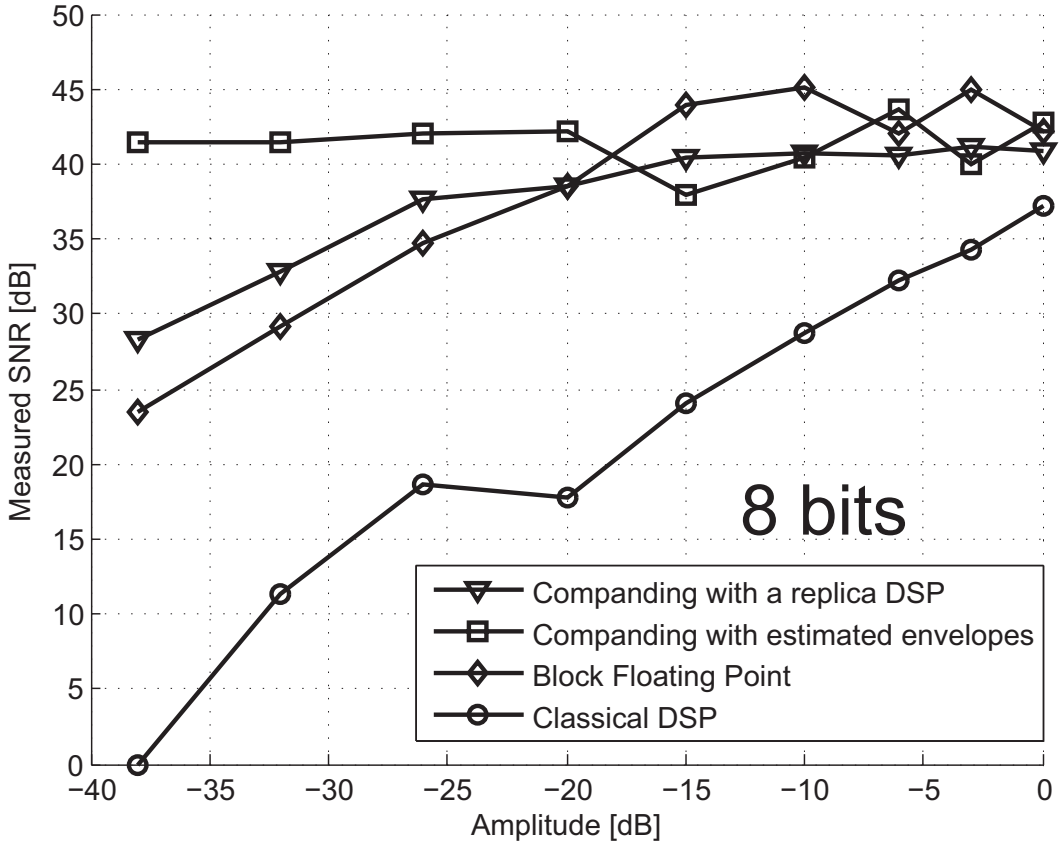

Figure 8.4: SNR comparison for a 500 Hz input tone.

The system discussed above was implemented in C and simulated with both pure-tone and speech inputs, and was presented in [44]. Fig. 8.4 shows the SNR for all systems when their inputs

are a 500 Hz encoded tone. Block-floating-point [134]- [135], an existing technique for providing larger dynamic range using just fixed-point arithmetic, was also included and implemented for comparison purposes. All systems operate in 8-bit, fixed-point arithmetic, meaning that they use 8-bit registers and multipliers, and 16-bit accumulators, adders, subtracters and shifters. As shown, the SNR at the output of the companding and BFP systems is very close to the full-scale SNR over a large input dynamic range (DR); such is not the case for the 8-bit classical system. Thus, for a fixed target SNR, the companding system can provide a much larger DR than a classical system using the same number of bits.

A typical output spectrum for a companding MPEG DSP is shown in Fig. 8.5. This example corresponds to a full-scale 500-Hz input tone to the all-pass reverberator described previously. The resulting SNR is 43 dB. However, as is made evident from Fig. 8.5, the output spectrum is vastly different from that of a classical fixed-point DSP. The majority of the quantization noise occupies the first MPEG sub-bands, to which the input signal was encoded. The noise at the remaining bands is suppressed to a very large extent, with some quantization noise present at the second sub-band (since some small part of the signal was encoded there as well, due to the overlapping nature of the MPEG encoding filter-banks).

The noise at each band is suppressed by the portion of the signal encoded into that particular band, denoted by the sub-bands's scale factor. This is in full agreement with the operation of companding DSP, as expected by eqs. 8.3. This process is controlled by the MPEG scale factors, which denormalize the output of each band after the DSP. Sub-bands which contain little or no signal will be denormalized, i.e. multiplied, by a small output scale factor, and effectively undergo a suppression for their quantization noise. Only sub-bands containing large signals, like the first sub-band in this experiment, will have the 8-bit quantization noise present inside them.

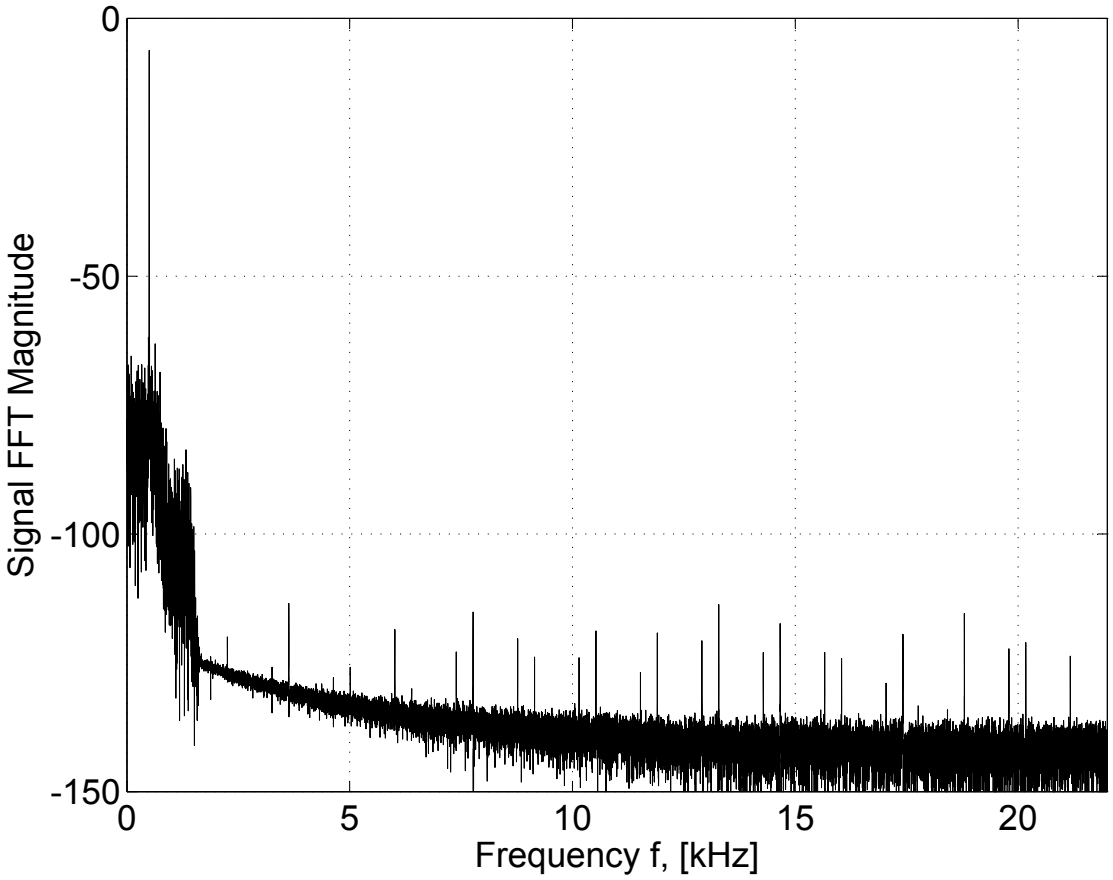

Figure 8.5: Spectrum of the companding MPEG DSP (with "guessing" envelope approach) for a full-scale 500 Hz input tone and all-pass reverberator DSP. All the large quantization noise (due to fixed-point, limited precision processing) is masked by the large output tone.

8.5 Discussion: SNR and audio fidelity in MPEG processors

Fig. 8.4 alone does not fully characterize the performance of the system. As shown via the example of Fig. 8.5, companding MPEG DSPs have different performance than classical MPEG DSPs. While the two systems have similar overall SNR, the frequency placement of quantization noise results in much higher performance in companding MPEG DSPs due to noise masking. Quantifying the performance of companding DSPs (e.g. comparing an 8-bit companding MPEG DSP to a 12-bit classical DSP) is no easy task, since it takes many different listening tests, with hundreds of

participants and carefully controlled procedures. Such characterization was not within the scope of this work.

Finally, it is important to note that SNR tests (involving static, sinusoidal signals, effectively fixed-envelope inputs) are not enough to describe such systems. Especially given the dynamic approaches for real-time envelope, i.e. magnitude, estimation, tests involving signals with varying envelopes are also important, to estimate the accuracy of envelope calculations. As such, we also fed the companding MPEG system with audio signals, including speech and music signals. Sample audio clips have been posted on a web site [130]; the use of only 8 bits allows us to audibly demonstrate the noise reduction achievable using the techniques presented, which would not be the case if more bits had been used, as the noise in all cases would then be largely imperceptible, due to limitations of the medium. Listening tests confirmed that the quantization noise of the companding system is significantly reduced relative to that of the classical DSP, due to the higher SNRs shown in Fig. 8.4 and the masking properties of the MPEG-audio reconstruction filter bank.

8.6 Summary

This chapter presented two design approaches for a processor of MPEG-encoded signals before the decoding process. By combining companding and sub-band processing, the requirement for high-precision processors was relaxed. Companding enabled significant SNR improvement for small-amplitude signals, while processing in the sub-band domain and prior to denormalization served a dual purpose. Besides the ease in obtaining input envelopes, necessary for companding, sub-band processing ensured the close proximity of large distortion and noise to large signals in the frequency domain. The net effect is that the overall audio fidelity of the obtained outputs was much

higher than that of classical, fixed-point DSPs, as confirmed by listening tests. Using only fixed-point arithmetic internally, the MPEG DSP architectures operate with smaller latency and overall energy drain. Two different processor architectures were presented, one following the classical companding theory [131] and the other using a heuristic envelope extractor for further arithmetic optimizations.

Chapter 9

Directions for Future Work

This thesis presented design methodologies for CT and companding DSPs using asynchronous and adaptive techniques. A number of interesting future paths open up, following up on the work of this thesis.

The processing technique for LCS-encoded samples is still at an early stage, with much more work remaining to be done. Most work has to revolve around optimizing the solution of the large linear system (eqns. 3.11) to solve the interpolating coefficients. A real-time solution for the inversion of this matrix will be a decisive step toward a real-time implementation for this class of DSPs; such a real-time implementation can lead so silicon implementations. One more path toward a real-time implementation can also be the exploration of different, more efficient types of interpolation.

CT DSPs also have much room for improvement, despite the few silicon prototypes that have been demonstrated. By using the design techniques presented in Chapter 4, it is hoped that CT DSPs of higher resolution and bit-width can be fabricated, effectively providing solutions for higher-quality processors, for which synchronous implementations are currently the single avail-

able solution. Furthermore, it is hoped that the asynchronous methodologies that we presented we presented can lead to continous-time IIR filters; the latter have their own set of design challenges, one of them being the error accumulation due to adder congestion in the feedback path. IIR filters require a much smaller number of delay segments than FIR filters for achieving the same rejection in the stop-band; they can alleviate the large area requirements of current FIR implementations.

Combining concepts from the two classes of processors, CT and LCS, can also be used to tackle one more drawback of CT DSPs, which is the very high data rate at the output of the processor. By doing some real-time output interpolation, while still maintaining the CT nature of the processor, one may be able to alleviate the requirement for a very-high-speed DAC.

CT DSPs can also benefit from the concept of an adaptive delay line. A silicon implementation, with or without the surrounding hardware to construct a CT DSP can help demonstrate the benefits shown by simulations. Powering down the part of the delay line which is made dynamically inactive will also reduce leakage power. The latter is, given the large size of delay lines, an important issue; reducing it by factors of two, four or even higher will have a significant impact on any CT DSP's energy consumption. A careful examination of statistics of traffic in application examples can lead to suggestions on the adaptive settings for these domains.

Finally, a few suggestions on future work can also be made for companding DSPs. First, the heuristic methods we presented for the envelope extractions can be further analyzed in terms of their effects on signal quality, reporting the associated trade-offs. Second, by further exploiting the properties of the encoded signals, further reductions to the complexity of the DSP can be obtained: an example is to look at the sub-band locations of the signal components in the encoded signals, and possibly disabling some of the sub-band processors.

Appendix A

Comparison and Jitter Analysis for Serial and Parallel Continuous-Time Delay Lines

In this chapter we compare two topologies that can be used in the design of a real-time digital delay line. More specifically, the objective is to design "delay segments" (defined as a part of the delay line, e.g. located between two adjacent taps of an FIR or IIR transversal structure). The two approaches we examine are the serial and parallel, as they have been defined in Chap. 4. In this chapter, we will first derive an analytic expression for the mean square variation of the delay of these two structures. This mean square variation is equal to the jitter (delay cell standard deviation) squared. Then, we will compare the two structures for jitter, area and dynamic power.

The following conditions apply to the traffic that these topologies can process:

- The minimum spacing between consecutive samples is T_{MIN}
- The average spacing of input samples is T_{AVG}
- The segment's delay should be T_{SEG}

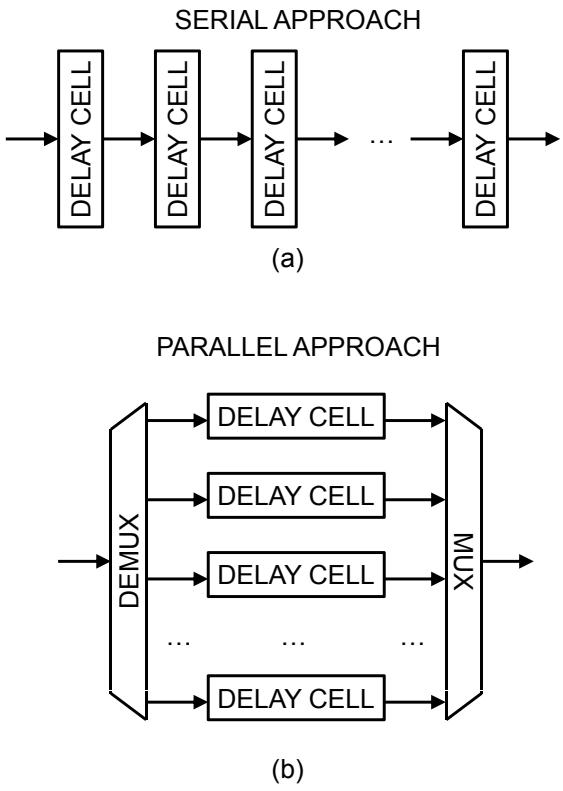

Figure A.1: Approaches for a CT DSP delay line: (a) serial and (b) parallel.

We examine the following two topologies to build the delay segment (shown in Fig. A.1):

- *Serial/pipelined* [33], [29], [36]: A delay segment consists of a serial interconnection of "delay cells", as in Fig. A.1(a). Each delay cell has a delay T_{MIN}; each segment has $N_{cells,s}$ cells and thus a total delay of $T_{SEG} = N_{cells,s} \cdot T_{MIN}$.

- *Parallel/cyclic* [28]: A delay segment consists of a parallel connection of delay cells, as in A.1(b), and a means of multiplexing between the inputs and outputs of the delay cells in a cyclic manner. The first incoming event to the delay segment is sent to and delayed by the first delay cell, the second incoming event to the second delay cell etc. (and all the delay cells' outputs are multiplexed to form the segment's output stream). A delay segment has $N_{cells,p}$ cells, and each of the delay cells has a delay of T_{SEG} (therefore each incoming event

undergoes a total delay of T_{SEG}, via a single delay cell, different for each sample).

The granularity (number of cells for each segment) in the two topologies is determined in different ways. In the serial approach, the factors which set the granularity are the minimum sample spacing and total segment delay; in the parallel approach the granularity is set through the total segment delay and average sample spacing.

A.1 Assumptions

A number of concrete assumptions will be first made, to help the following analysis and comparison.

The first assumption is quite obvious, namely that on average the incoming input event spacing is larger than the minimum one:

$$T_{AVG} > T_{MIN} \tag{A.1}$$

In particular, the input stream to any CT-DSP will by default (and due to the way the CT-ADC generates the input samples) consist of varying-density traffic; some input samples will be closely spaced (with spacing as little as T_{MIN}) but the spacing for some of the samples following those will be larger. The spacing between input samples will expand and shrink in a smooth manner.

T_{MIN} and T_{AVG} must be calculated for the particular ADC used at the input of the delay line. For a level-crossing CT-ADC with 8-bit resolution processing up to 20 kHz input signals, $T_{MIN} = 60$ ns and $T_{AVG} = 200$ ns .

We also assume that the delay cells have a design similar to the one in [33], [36]. A simplified

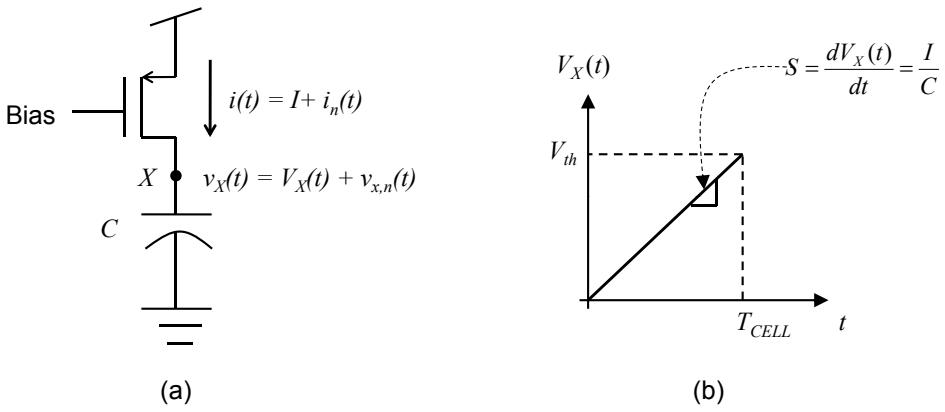

Figure A.2: Delay cell approximation for jitter analysis.

view of the delay cell is shown in Fig. A.2(a). Each delay cell consists of a "datapath" part and a "control" part. Only the datapath is shown in Fig. A.2, and consists of a capacitor (C), a controlled current source ($I(t)$) to charge C and a positive feedback structure, which is not shown in Fig. A.2. The current source has a nominal value I and a noise component $i_n(t)$. The control part of the segment includes the remaining gates and transistors (besides the current source and capacitor) in Fig. 5.4. Contrary to the case in Fig. 5.4, where a n-MOS transistor charges a capacitor referenced to the supply, in the simplified view used for jitter analysis (shown in Fig. A.2) a p-MOS device charges a capacitor referenced to ground; this structural difference has no impact on the jitter of the cell (besides the slightly different noise of the p-MOS and n-MOS devices which we ignore).

Fig. A.2 shows the voltage on node X, i.e. the voltage across capacitor C, while the cell is active: initially $v_X(0) = 0$ and when the cell starts operating it pulls X toward the supply voltage. In the absence of noise, i.e. if $i_n(t) = 0$, the voltage on node X increases linearly, at a constant slope or rate $S = \frac{dV_X(t)}{dt} = \frac{I}{C}$. When the voltage on node X becomes equal to a threshold voltage V_{th}, the charging operation is completed and the delay cell passes the sample to its successor cell and quickly discharges C, i.e. pulls node X back to ground. The voltage V_{th} is actually the

threshold voltage of a positive-feedback, thyristor-type circuit, which is shown in Fig. 5.4 in Chap. 5. This threshold voltage is assumed to be equal for all cells and with zero variance for the sake of simplicity. To ensure that V_{th} has small variation in practice across the different cells, the size of the corresponding device (P1 in Fig. 5.4) is four times larger than the minimum-size device.

The delay of each cell is equal to

$$T_{CELL} = V_{th} \cdot \frac{C}{I} \tag{A.2}$$

As a reference point, we name $C0$ the value of the capacitor C that ensures that $T = T_{MIN}$ and that the jitter of the serial delay segment has a given, reference value. It will be shown (see next section), that the capacitance required in the parallel approach for the same jitter is directly related to C_0.

The energy consumed by each cell for one operation (i.e. delaying a single event), E_{event}, can be broken down to:

$$E_{event} = E_{datapath,C_0} \cdot \frac{C}{C_0} + E_{control} \tag{A.3}$$

The first part in the right hand side of eq. A.3 refers to the datapath energy. The latter is proportional to the value of the charging capacitor C. $E_{datapath,C_0}$ is the datapath energy when $C = C_0$, i.e. when the delay cell has a delay T_{MIN} and the entire serial delay segment has the reference jitter. $E_{control}$ is a fixed energy amount (independent of jitter); it only depends on the delay cell topology and protocol.

A similar formula can be given for the area of a delay cell:

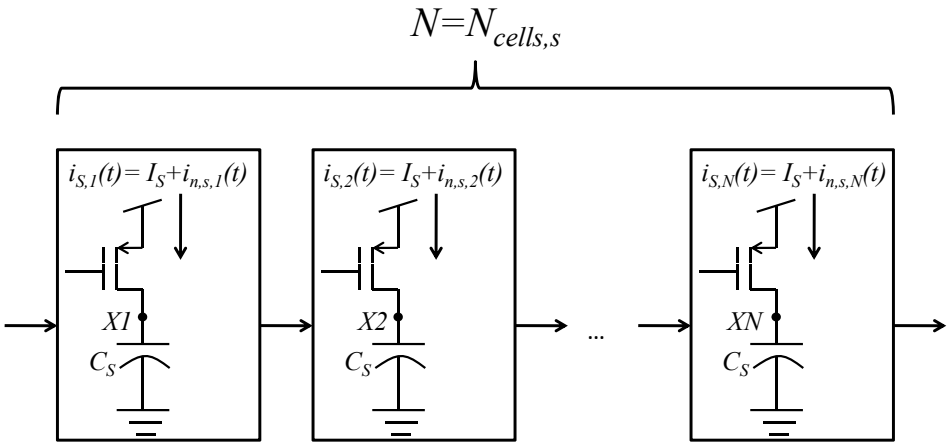

Figure A.3: Delay line serial approach, as a cascade of delay cells.

$$A_{cell} = A_{datapath,C_0} \cdot \frac{C}{C_0} + A_{control} \tag{A.4}$$

A.2 Jitter analysis

The jitter power (or variance in delay) of a delay segment (and, thus, of the entire delay line) is a very important parameter of delay cells, since its presence results in noise in a CT-DSP's output. In this section we will derive expressions for the delay variance of a serial and a parallel delay segment implementation, and find out under which condition the variance for the two approaches is the same.

A.2.1 Serial approach

In the serial approach, each cell contains (see Fig. A.3) a capacitor with value C_S and a pMOS transistor (acting as a current source) with a nominal current I_S. The nominal delay of the cell is $T_{CELL,s} = \frac{V_{th} \cdot C_S}{I_S}$.

The pMOS transistor has a total current $i_S(t)$:

$$i_S(t) = I_S + i_{n,s}(t) \tag{A.5}$$

, where $i_{n,s}(t)$ is the noise current which results in variance in the cell's delay; $i_{n,s}(t)$ is assumed to be wide-sense-stationary white noise with a power-spectral density $S_{iw,s}$. The assumption of infinite-bandwidth white noise simplifies some of the resulting equations. It has been attempted to do the subsequent analysis with a band-limited noise as well, and the results did approach the results using this assumption. In a practical case, there will be limitations to the bandwidth due to the finite speed of the circuitry involved. The flicker ($\frac{1}{f}$) noise of the devices is ignored.

If $i_{n,s}(t) = 0$, then the during the cell's active phase the voltage on node X would be an ideal ramp-up voltage, starting from value 0 at $t = 0$ and taking the value V_{th} exactly at $t = T_{CELL,s}$ (see Fig. A.2), at which point the cell's operation would complete (with zero time variance). In other words, node X would be charged in this case from 0 to V_{th} at a constant slope $S = \frac{I_S}{C_S}$, as shown in Fig. A.2(b).

The presence of the noise current $i_{n,s}(t)$ will result in the voltage on node X, $V_x(t)$, having a noise component, $v_{x,n}(t)$, in addition to the expected ramp voltage $V_X(t)$:

$$v_X(t) = V_X(t) + v_{x,n}(t) \tag{A.6}$$

The voltage noise component will result in a noise component, $\tau_{cell,s,n}(t)$, in the cell's delay:

$$\tau_{CELL,s}(t) = T_{CELL,s} + \tau_{cell,s,n}(t) \tag{A.7}$$

A delay segment has $N_{cells,s}$ cells, serially connected, as shown in Fig. A.3. This figure is very abstract, but indicates that all cells contain a capacitor C_S and a current source of nominal value I_S, as well as the fact that all current noise sources are completely independent. The nominal delay of each cell is $T_{CELL,s} = \frac{V_{th} \cdot C_S}{I_S}$, resulting in a nominal segment delay of $T_{SEG,s} = N_{cells,s} \cdot T_{CELL,s}$.

Due to all noise current sources in Fig. A.3, the segment's delay will contain an additional noise component $\tau_{seg,s,n}(t)$, resulting in an overall segment delay $\tau_{SEG,s}(t)$:

$$\tau_{SEG,s}(t) = T_{SEG,s} + \tau_{seg,s,n}(t) \tag{A.8}$$

The ultimate goal is an expression for the variance of $\tau_{seg,s,n}(t)$. We start by finding the variance of the delay of one cell. This problem is the same as the very-well-known "brownian motion with drift" problem in stochastic calculus [136]; here we will only sketch some key results (after some simplifications to avoid lengthy equations). We will denote the variance of variable Y as $Var[Y]$.

Referring back to Fig. A.2 and to eqn. A.6, during the active phase of a cell's operation, the voltage on node X [1], $v_X(t)$, is equal to:

$$\begin{aligned} v_X(t) &= \frac{1}{C_S} \cdot \int_0^t i_S(\lambda) d\lambda = V_X(t) + v_{x,n}(t) \\ &= \frac{I_S}{C_S} \cdot t + \frac{1}{C_S} \cdot \int_0^t i_{n,s}(\lambda) d\lambda \end{aligned} \tag{A.9}$$

The first method to calculate the variance of $v_{x,n}(t)$ at any given time t consists of applying the definition of the variance on the integral-based expression for $v_{x,n}(t)$ in eqn. A.9, then using the properties of the white noise current. In more detail, we are using the fundamental property that

[1] Node X can be any of the nodes Xi in Fig. A.3

all noise instances are completely independent.

We start by using the ergodic nature of $i_{n,s}(t)$, hence $v_{x,n}(t)$, which allows us to calculate the variance (mean square value) of $v_{x,n}(t)$ as:

$$Var[v_{x,n}(t)] = \overline{v_{x,n}(t)^2} = E[v_{x,n}(t)] \tag{A.10}$$

, where E[] denotes the expected value. Therefore:

$$\begin{aligned} Var[v_{x,n}(t)] = E[v_{x,n}^2(t)] = \cdot E[v_{x,n}(t) \cdot v_{x,n}(t)] = \\ = \frac{1}{C_S^2} \cdot E[\int_0^t i_{n,s}(\lambda_1) d\lambda_1 \cdot \int_0^t i_{n,s}(\lambda_2) d\lambda_2] = \\ = \frac{1}{C_S^2} \cdot E[\int_0^t \int_0^t i_{n,s}(\lambda_1) i_{n,s}(\lambda_2) d\lambda_1 d\lambda_2] = \\ = \frac{1}{C_S^2} \cdot \int_0^t \int_0^t E[i_{n,s}(\lambda_1) i_{n,s}(\lambda_2)] d\lambda_2 d\lambda_1 = \end{aligned} \tag{A.11}$$

We now use the key property of white noise, that all white noise samples are independent random variables, which is expressed as:

$$E[i_{n,s}(t_1) \cdot i_{n,s}(t_2)] = S_{iw,s} \cdot \delta(t_1 - t_2) \tag{A.12}$$

, where $\delta(t)$ is the Dirac function and $S_{iw,s}$ is the noise PSD. Substituting eqn. A.12 into eqn. A.11, we get:

$$Var[v_{x,n}(t)] = \frac{1}{C_S^2} \cdot \int_0^t \int_0^t S_{iw,s} \delta(\lambda_1 - \lambda_2) d\lambda_2 d\lambda_1 = \frac{1}{C_S^2} \cdot \int_0^t S_{iw,s} d\lambda_1 = \frac{t}{C_S^2} \cdot S_{iw,s} \tag{A.13}$$

This is the famous result of Brownian motion, the variance of the motion increases linearly with time. The result of eqn. A.13 agrees with theory, both qualitatively and quantitatively. A much more intuitive approach, though much less rigorous and based on many assumptions can be found here [2] .

Going back to eqn. A.9, the noise voltage $v_{x,n}(t)$ is added to a ramp voltage to form the total

[2] As an alternative approach to calculate the variance of the voltage on node X, $Var[v_{x,n}(t)]$ as a function of time, we first approximate the integral of eqn. A.9 as a finite sum (using small time steps), then calculate the variance using this sum and finally take the limit as the time step approaches zero. This approach is much less rigorous, however much more intuitive. We, therefore, approximate the continuous nature of the charging process of duration t as a series of an integer amount N_{steps} of small steps Δt, where:

$$N_{steps} = \frac{t}{\Delta t}$$

Since we now have a discrete-time process, we assume that the noise current PSD is now band-limited to only extend up to the maximum bandwidth of the discrete-time system. Therefore, we assume that the noise is band-limited to B, where:

$$B = \frac{1}{2\Delta t}$$

The assumption of a band-limited noise allows discretization of time without any problems caused by the aliasing of noise. As Δt goes to zero, we have the case of a perfect white noise with infinite bandwidth. Now we can write the noise voltage on node X, $v_{x,n}(t)$ in eqn. A.9, as:

$$v_{x,n}(t) = \sum_{k=1}^{N_{steps}} \frac{1}{C_S} \cdot i_{n,s}(k) \cdot \Delta t$$

, where $i_{n,s}(k)$ denotes the sample of the noise current at the k-th step. Taking the variance of both sides:

$$Var[v_{x,n}(t)] = Var[\sum_{k=1}^{N_{steps}} \frac{1}{C_S} \cdot i_{n,s}(k) \cdot \Delta t]$$

All N_{steps} noise current values are considered independent and with identical variances, therefore $Var[i_{n,s}(k)] = Var[i_{n,s}]$ for all k and :

$$Var[v_{x,n}(t)] = N_{steps} \cdot Var[\frac{1}{C_S} \cdot i_{n,s}(k) \cdot \Delta t] = N_{steps} \cdot \frac{\Delta t^2}{C_S^2} \cdot Var[i_{n,s}]$$

We recognize that the variance of the noise current can be calculated as the integral of the noise PSD over frequency:

$$Var[i_{n,s}] = \int_{-\infty}^{+\infty} S_{iw,s} df = \int_{-B}^{+B} S_{iw,s} df = 2 \cdot B \cdot S_{iw,s}$$

Using using the eqs. above, we end up with:

$$Var[v_{x,n}(t)] = N_{steps} \cdot \frac{\Delta t^2}{C_S^2} \cdot Var[i_{n,s}] = \frac{t}{\Delta t} \cdot \frac{\Delta t^2}{C_S^2} \cdot 2 \cdot \frac{1}{2\Delta t} \cdot S_{iw,s} = t \cdot \frac{S_{iw,s}}{C_S^2}$$

, which is the same result as before.

voltage on node X. At times when the total voltage is equal to V_{th}, typically the noise component is much smaller than the ramp component. Therefore, a small noise voltage δv will change the time at which the total voltage crosses the threshold by a small time amount, equal to

$$\delta t = \frac{\delta v}{\frac{dv_X(t)}{dt}} \tag{A.14}$$

, where $\frac{dv_X(t)}{dt}$ is the local slope of the total node voltage close to the point where $v_X(t) = V_{th}$. To simplify things, we consider that the slope of the total voltage is dominated by the slope of the ramp voltage S, therefore $\frac{dv_X(t)}{dt} \approx S$. This assumption may not always be correct, since (in principle) the slope of the noise component may be very large at times; however, in practical case, the capacitor (which integrates the noise) will limit the slope of $v_{x,n}(t)$.

$$Var[\delta t] = \frac{Var[\delta v]}{S^2} \tag{A.15}$$

Note that the above may not always hold, making the analysis very complicated. In [137], the author addresses the effect of the noise's slope even when the latter is not dominated by the ramp's slope; it is found that in most cases the noise slope has small effects.

By substituting $\delta v = v_{x,n}(t)$ and $t = \tau_{CELL,s}$ and $S = \frac{I_S}{C_S}$ in A.15, and using the result of eqn. A.13, we get:

$$Var[\tau_{cell,s,n}] = \frac{Var[v_{x,n}(t)]|_{(t=T_{CELL,s})}}{(\frac{I_S}{C_S})^2} = T_{CELL,s} \cdot \frac{S_{iw,s}}{I_S^2} \tag{A.16}$$

Referring now to Fig. A.3, and remembering that all the cells' noise sources are uncorrelated, we calculate the variance in the segment's delay. For $N_{cells,s}$ uncorrelated random variables, the

individual variances add up to give the total one:

$$\begin{aligned} Var[\tau_{seg,s,n}] &= N_{cells,s} \cdot Var[\tau_{cell,s,n}] = \\ &= N_{cells,s} \cdot T_{CELL,s} \cdot \frac{S_{iw,s}}{I_S^2} = T_{SEG,s} \cdot \frac{S_{iw,s}}{I_S^2} \end{aligned} \tag{A.17}$$

Eq. A.17 has been verified both through MATLAB simulations and through transient-noise simulations of an existing serial delay-segment implementation in Cadence.

A.2.2 Parallel approach

In the parallel approach, shown abstractly in Fig. A.4, each segment consists of $N_{cells,p}$ cells, each of which holds a capacitor C_P and a current source with nominal value I_P, and has a nominal delay equal to $T_{CELL,p} = \frac{V_{th} \cdot C_P}{I_P}$. The segment delay for this case $T_{SEG,p}$ is equal to $T_{CELL,p}$, since each event only goes through one delay cell from the input to the output of the segment.

$$T_{SEG,p} = T_{CELL,p} \tag{A.18}$$

All noise components are (similar to the serial case) WSS white-noise sources with PSD equal to $S_{iw,p}$ and are all considered uncorrelated.

For each of the $N_{cells,p}$ cells of the parallel approach, eqs. A.5-A.16 will hold after replacing all 's' (and 'S') subscripts with 'p' (and 'P' respectively), indicating design variables (current, capacitance, noise etc.) in the parallel approach. Each delay cell will have a delay variance $Var[\tau_{cell,p,n}]$, equal to:

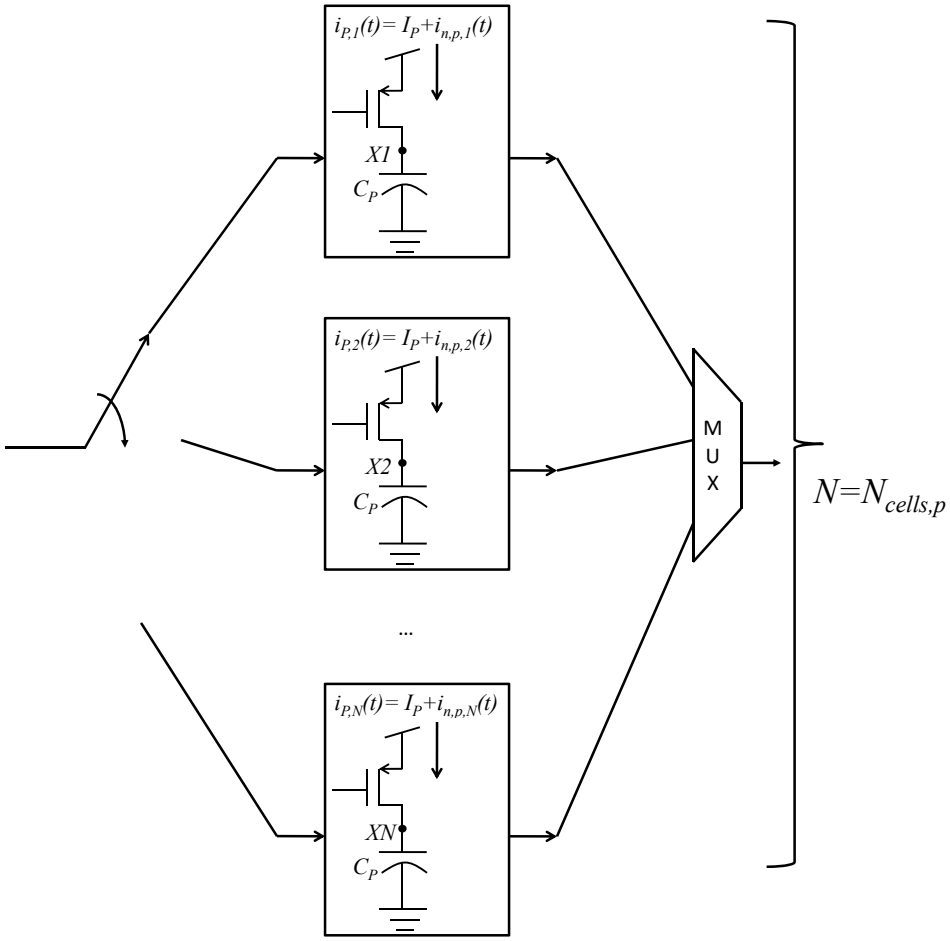

Figure A.4: Delay line parallel approach.

$$Var[\tau_{cell,p,n}] = T_{CELL,p} \cdot \frac{S_{iw,p}}{I_P^2} \tag{A.19}$$

Using eq. A.18:

$$Var[\tau_{seg,p,n}] = T_{SEG,p} \cdot \frac{S_{iw,p}}{I_P^2} \tag{A.20}$$

A.2.3 Comparison

Finally, consider the requirement for a serial and parallel design with the same delay ($T_{SEG,s} = T_{SEG,p}$) and the same variance (hence jitter as well). Using eqs. (A.17-A.20), we get:

$$\begin{aligned} &Var[\tau_{seg,s,n}] = Var[\tau_{seg,p,n}] \Rightarrow \\ &T_{SEG,s} \cdot \frac{S_{iw,s}}{I_S^2} = T_{SEG,p} \cdot \frac{S_{iw,p}}{I_P^2} \Rightarrow \\ &\frac{S_{iw,s}}{I_S^2} = \frac{S_{iw,p}}{I_P^2} \end{aligned} \tag{A.21}$$

Assuming that all current sources are identically biased, i.e. they all operate in strong inversion, we recognize that the function $\frac{S_{iw,s}}{I_S^2}$, i.e. PSD of the noise current normalized to the square of the current, is a one-to-one function of the nominal current, so the following applies:

$$\frac{S_{iw,s}}{I_S^2} = \frac{S_{iw,p}}{I_P^2} \Longleftrightarrow I_S = I_P \tag{A.22}$$

, so eqn. A.21 translates to

$$I_S = I_P \tag{A.23}$$

We, therefore, reach the conclusion that the demand for equal delay and jitter in a serial and parallel delay segment translates to the demand that all delay cells, in both the parallel and serial approach, hold the same charging current. This argument does not hold for the total charging current in the entire serial and parallel segments, since (as we will see in Sec. A.3) the serial and parallel segments do not have the same total number of cells.

As for capacitances, we begin by calling C_0 the value of C_S for which a certain jitter specifica-

tion (for the entire segment) is met in the serial approach. It is clear that C_0 is computed indirectly, one first calculates the required current I_S to meet the jitter requirement using eq. A.17 and then uses eq. A.2 to calculate C_0.

$$C_S = \frac{T_{CELL} \cdot I_S}{V_{th}} = C_0 \tag{A.24}$$

Each cell in the serial approach will hold a capacitor with value $C_S = C_0$. Each cell in the parallel approach must hold a larger capacitor C_P, in order to achieve a larger delay (the delay of the entire segment).

$$C_P = \frac{T_{SEG} \cdot I_P}{V_{th}} = \frac{N_{cells,s} \cdot T_{CELL} \cdot I_P}{V_{th}} = N_{cells,s} \cdot C_0 \tag{A.25}$$

It is clear that each cell in the parallel approach requires a capacitor equal to the *total* capacitance in the entire segment in the serial approach.

Translating that to a per-segment comparison, we note that one parallel segment has $N_{cells,p}$ delay cells, each of which has capacitor $N_{cells,s} \cdot C_0$ according to eqn. A.25. Therefore, one parallel segment has $N_{cells,p}$ times the total capacitance of the serial segment! Given that $N_{cells,p}$ can be in the order of 100 or more, this is a significant area drawback of the parallel approach.

A.3 Comparison of the two topologies

We now compare the two approaches in terms of three of their aspects: event capacity (to be defined shortly), energy and area. The comparisons are made for a given segment delay T_{SEG} for the two approaches, as well as given minimum and average spacing between input samples (T_{MIN}

and T_{AVG} respectively).

A.3.1 Event capacity

"Event capacity" (a term equivalent to "granularity" in [33]), refers to the number of cells per segment required in each approach. The number of cells is literally a measure of event capacity, since it indicates how many events the segment can concurrently hold and process at any time.

Serial approach

What defines the number of cells, $N_{cells,s}$, in the serial approach? Since the latter is a pipeline, we want each stage of the pipeline (i.e. each delay cell) to have finished processing one event by the time the next one arrives (to avoid congestion). Given this, we want the delay of each cell to be (at most equal to) T_{MIN}. Now given the target T_{SEG}, we easily conclude that:

$$N_{cells,s} = \frac{T_{SEG}}{T_{MIN}} \tag{A.26}$$

More precisely, since the result of the above may not be an integer, the number of required cells is the next closest integer to $\frac{T_{SEG}}{T_{MIN}}$. Note that this is a "worse-than-worst-case" design. The pipeline can be expected not to be completely occupied (given the environmental constraint that $T_{AVG} > T_{MIN}$). As an example imagine the case of a level-crossing A/D processing a sinusoidal input and feeding the corresponding sampling instances to the delay segment: each half-period of the sinusoid will contain only one pair of min-spaced events followed by many more events spaced by more than T_{MIN}. Given this, some events in each segment will be closely-spaced enough to occupy neighboring cells, while other events will be so largely-spaced that several delay cells in-

between will be left temporarily un-occupied. This overdesign is only done to respect the demand for handling a pair (in every 200 or so events) of minimum-spaced events!

Parallel approach

What defines the number of cells in the parallel approach? The way to determine this is to ask the following question: "What needs to happen in order for one sample to try and enter a delay cell which is already occupied?". Simply, the following must happen: starting from initial rest (no cell is occupied), one event (say the 1st) enters the first delay cell (where it is delayed by T_{SEG}), the next one enters the 2nd delay cell, etc., the($N_{cells,p}$)-th event enters the last cell and then the ($N_{cells,p}+1$)-th event tries to enter the first cell but the latter is occupied. In order for this not to happen, all the above-mentioned events must be done within a time frame less than T_{SEG} (so that the 1st cell is still occupied):

$$T_{SEG} \leq S(1,2) + S(2,3) + ... + S(N_{cells,p}, N_{cells,p}+1) \tag{A.27}$$

, where $S(i,i+1)$ denotes the time spacing between the i-th and $(i+1)$-th events. We can rewrite the above sum as $N_{cells,p} \cdot \overline{S(i,i+1)}$, where $\overline{S(i,i+1)}$ denotes the average value of event spacing:

$$\begin{aligned} & T_{SEG} \leq N_{cells,p} \cdot \overline{S(i,i+1)} \\ & \Rightarrow T_{SEG} \leq N_{cells,p} \cdot T_{AVG} \\ & \Rightarrow N_{cells,p} \geq \frac{T_{SEG}}{T_{AVG}} \end{aligned} \tag{A.28}$$

Note that:

- Compared to the previous approach, violating this condition does not affect the overall segment delay. Even if a segment is designed with less cells, all events that enter the segment will still undergo a time delay of T_{SEG}. If the corresponding condition (see eq. A.26) for the serial approach is violated, the segment's delay will be different than specified.

- This condition only guarantees that there will be no congestion (and is not associated with the actual time delay of the segment)!

- In the limit when we take exactly $N_{cells,p} = T_{SEG}/T_{AVG}$, there will be a case when all delay cells in the segment will be occupied with events, i.e. we are making full use of the structure here!

A.3.2 Energy

In the previous section, we showed that (demanding equal jitter in the two approaches, as well as equal delay) the charging capacitance in each cell of the parallel approach must equal to the sum of the charging capacitances of all the delay cells in the serial approach. Referring back to eq. A.3, a single parallel cell dissipates (per event) the same datapath energy $E_{datapath}$ as all the serial cells combined.

How about the total energy, $E_{segment}$, required for one event to undergo a delay of T_{SEG}? The remaining factor to be considered is $E_{control}$ in eqn. A.3.

Serial approach

In this approach we have:

$$C_S = C_0 \tag{A.29}$$

$$N_{cells,s} = \frac{T_{SEG}}{T_{MIN}} \tag{A.30}$$

, so using eqs. A.3, A.29, A.30:

$$\begin{aligned} E_{segment,s} &= N_{cells,s} \cdot E_{event,s} \\ &= \frac{T_{SEG}}{T_{MIN}} \cdot (E_{datapath,C_0} \cdot \frac{C_0}{C_0} + E_{control}) \\ &= \frac{T_{SEG}}{T_{MIN}} \cdot (E_{datapath,C_0} + E_{control}) \end{aligned} \tag{A.31}$$

Parallel approach

In this case, each cell must hold a charging capacitor large enough to generate a T_{SEG} delay with the same current as the serial approach, so using eqs. A.4, A.18, A.25, A.30:

$$C_P = N_{cells,s} \cdot C_0 = \frac{T_{SEG}}{T_{MIN}} \cdot C_0 \tag{A.32}$$

$$\begin{aligned} E_{segment,p} &= 1 \cdot E_{event,p} \\ &= (E_{datapath,C_0} \cdot \frac{\frac{T_{SEG}}{T_{MIN}} \cdot C_0}{C_0} + E_{control}) \\ &= \frac{T_{SEG}}{T_{MIN}} \cdot E_{datapath,C_0} + E_{control} \\ &\approx \frac{T_{SEG}}{T_{MIN}} \cdot E_{datapath,C_0} \end{aligned} \tag{A.33}$$

In the last step, $E_{control}$ was ignored since it is typically much smaller than the product of the number of cells and the datapath energy of one cell.

So energy-wise we expect the parallel approach to be more energy-efficient. The reason behind this outcome is that each sample in the parallel approach only goes through one delay cell (of much larger delay), hence the control path (hand-shaking gates etc.) of only one delay cell are exercised. This is a big advantage compared to the serial structure, in which each sample exercises the control part of all the delay cells. The relative improvement in energy of the parallel approach depends on the ratio of $E_{datapath}$ and $E_{control}$, i.e. the contribution of the charging capacitance to the delay cell's energy relative to the contribution of the hand-shaking gates.

A.3.3 Area

To compare the two approaches for the total area, we follow the same line of reasoning as for the energy comparison.

Serial approach

We use eqs. A.4, A.29, A.30 to get:

$$C_S = C_0 \tag{A.34}$$

$$N_{cells,s} = \frac{T_{SEG}}{T_{MIN}} \tag{A.35}$$

$$\begin{aligned} A_{segment,s} &= N_{cells,s} \cdot A_{cell,s} \\ &= \frac{T_{SEG}}{T_{MIN}} \cdot (A_{datapath,C_0} \cdot \frac{C_0}{C_0} + A_{control}) \\ &= \frac{T_{SEG}}{T_{MIN}} \cdot (A_{datapath,C_0} + A_{control}) \end{aligned} \tag{A.36}$$

It's easy to see why the area increases *linearly* with T_{SEG} (and constant T_{MIN}. Ignoring the

jitter to first order to simplify the discussion, when one needs to generate a larger segment delay in the serial approach, one needs to keep the delay cells unaltered and simply add more of them.

Parallel approach

Similarly, using eqs A.4, A.25, A.28:

$$C_P = \frac{T_{SEG}}{T_{MIN}} \cdot C_0 \tag{A.37}$$

$$N_{cells,p} = \frac{T_{SEG}}{T_{AVG}} \tag{A.38}$$

$$\begin{aligned} A_{segment,p} &= N_{cells,p} \cdot A_{cell,p} \\ &= \frac{T_{SEG}}{T_{AVG}} \cdot (A_{datapath,C_0} \cdot \frac{\frac{T_{SEG}}{T_{MIN}} \cdot C_0}{C_0} + A_{control}) \\ &= \frac{T_{SEG}^2}{T_{AVG} \cdot T_{MIN}} \cdot A_{datapath,C0} + \frac{T_{SEG}}{T_{MIN}} \cdot A_{control} \end{aligned} \tag{A.39}$$

The area in the parallel approach increases *quadratically* with T_{SEG}. This is because (again ignoring jitter) the segment delay T_{SEG} now affects area in two ways:

- More area is needed in each individual cell (larger capacitance to generate larger delay)
- More cells are needed, in order to meet the "event capacitance" requirement!

The requirement of keeping jitter constant when increasing the segment delay will affect both approaches in the same way. The effect of designing for the same jitter but a larger segment delay is to increase both the charging capacitors and the currents by the same amount, in order to compensate for the increment in delay (see eqs. A.17 and A.20). In the parallel approach, this increase in the charging capacitors will be on top of the increase required to make the delay of

all cells larger. Both approaches suffer equally from the jitter requirement; however, the area of the parallel approach depends more heavily on the total segment delay requirement, as explained previously.

This means that to design fine-grain delay segments, i.e. when one is trying to generate large delays T_{SEG} with a small T_{MIN}, the area price may be prohibitive in parallel applications, even though the power benefit at first seems enticing.

In order to compare the two approaches, we present plots for the total segment area, the energy to handle a single sample, as well as the product of these two, all versus the segment delay. The energy-per-sample (eqs. A.31, A.33) is enough to compare the two approaches from an energy perspective. The total number of samples is the same for the two approaches (and is only set by the ADC used in a particular CT DSP system); hence the energy-per-sample suffices for energy and power comparisons of the two approaches. We remind that in this case power equals energy-per-sample times the sample rate.

The parameters for the datapath and control energy/area of the delay cells are taken from an 8-bit CT DSP system [36], configured as an audio-band low-pass filter. The traffic parameters T_{MIN}, T_{SEG}, T_{AVG} are taken by an 8-bit level-crossing ADC in [33], with maximum input frequency of 20 kHz. The parameters have the following values:

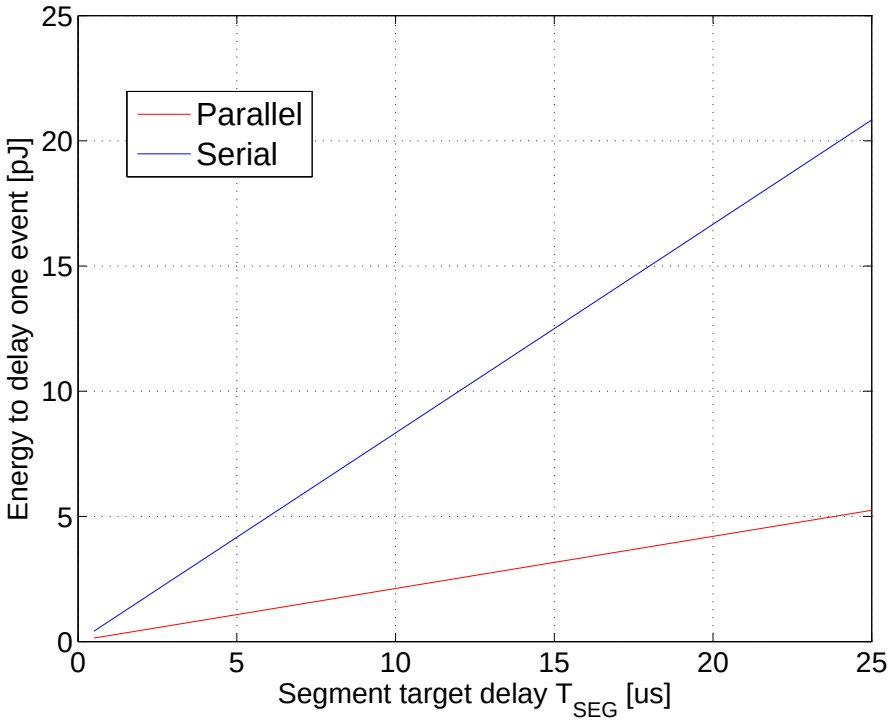

Figure A.5: Comparison for energy to handle / delay a single sample between serial and parallel delay line approaches.

$$E_{datapath} = 12 \text{ fJ}$$

$$E_{control} = 36 \text{ fJ}$$

$$A_{datapath} = 4 \ \mu\text{m}^2$$

$$A_{control} = 16 \ \mu\text{m}^2$$

$$T_{MIN} = 60 \text{ ns}$$

$$T_{SEG} = 20 \ \mu\text{s}$$

$$T_{AVG} = 200 \text{ ns}$$

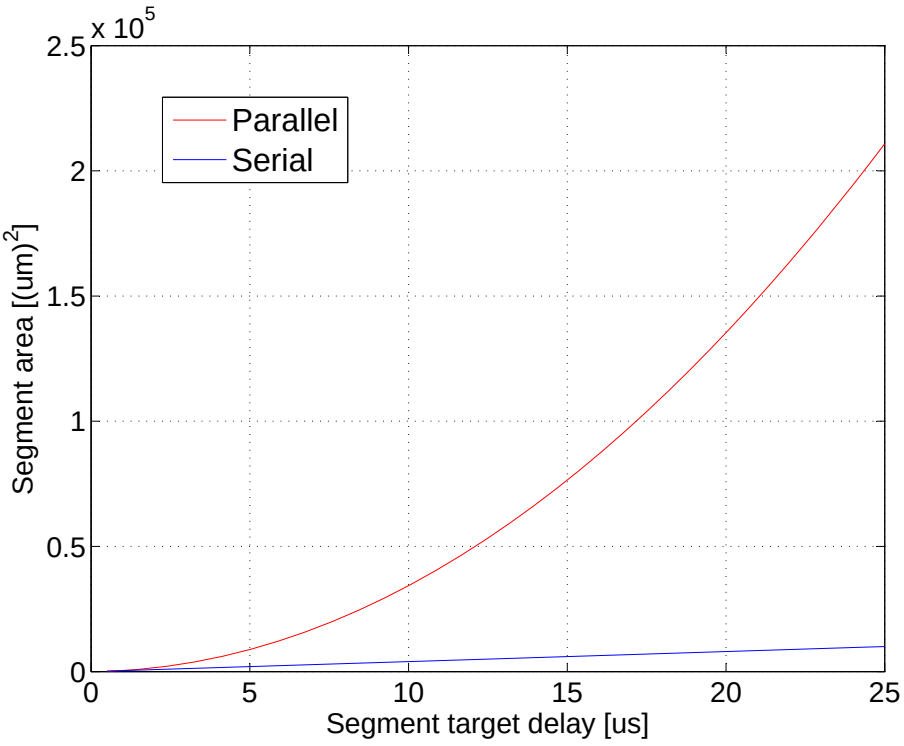

Figure A.6: Area comparison for serial and parallel delay line approaches.

Figure A.5 shows how the energy-per-sample of the two approaches varies with the segment's delay, while the minimum and average sample spacing are held constant, and the jitter of the two lines is equal. It is clear that the serial approach is less efficient; as explained previously, this is due to the extra overhead of the control hardware, which is exercised multiple times by each sample. In contrary, each sample in the parallel approach only exercises one delay cell, hence one control part.

Figure A.6 shows the total area comparison for the two approaches versus the delay of the segment, again with the minimum and average sample spacing held constant, and for the same jitter in the two approaches. Given the quadratic dependence of area on the segment's delay in the parallel approach, the latter has a very severe overhead as the segment's delay increases beyond the region of a few μ s, e.g. in voice-band applications.

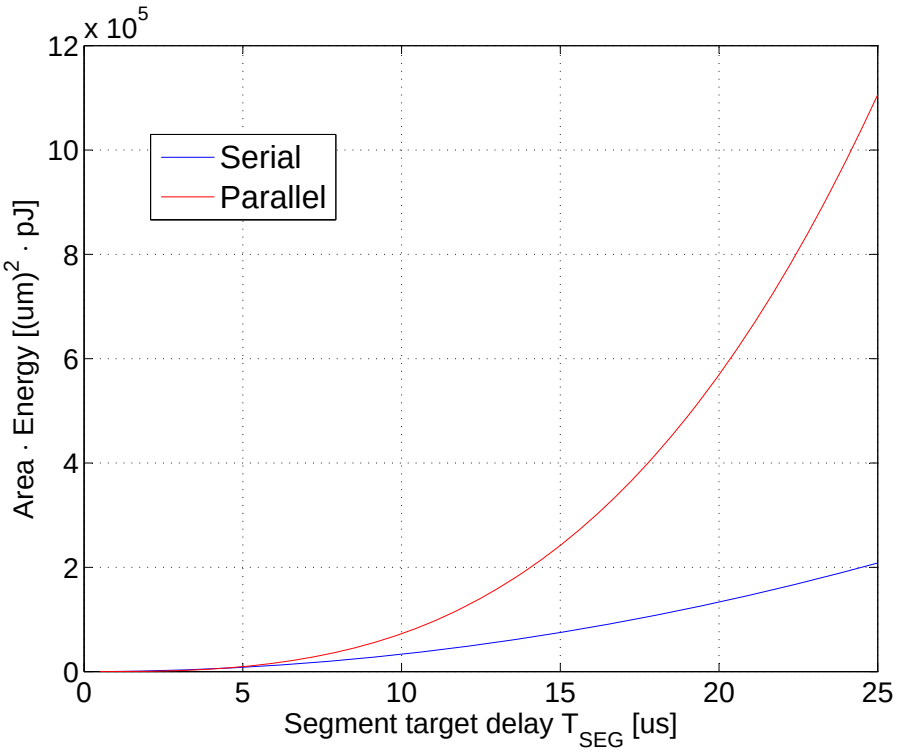

Figure A.7: Area·Energy product comparison foe serial and parallel delay line approaches.

Finally, Fig. A.7 helps compare the two approaches for both area and energy, by showing the product of total area and energy-per-sample, for the same conditions as in the two previous Figs. The smaller the area-energy product, the more efficient the structure is, requiring less power (for a give sample rate) and overall area. Despite the increased energy efficiency of the parallel approach, the area-energy product of the serial approach is much smaller, especially for segment delays above the 15-μs region. As we discussed previously, this is largely due to the quadratic dependence of the parallel version's total area on the overall segment delay. This is the reason for choosing the serial approach for the chip design presented in Chap. 5.

Appendix B

Burst-Mode Specifications for Asynchronous Controllers Used in the Adaptive Delay Line

This appendix presents the specifications and implementations of all burst-mode asynchronous controllers that were used for the control path of the adaptive delay line, i.e. the mode controller, in chapter 7. It includes all BM specifications used for the asynchronous controllers, which were then synthesized using the Minimalist tool [93], [90], [92]. For each BM specification, the initial state of each controller is the one marked “0”.

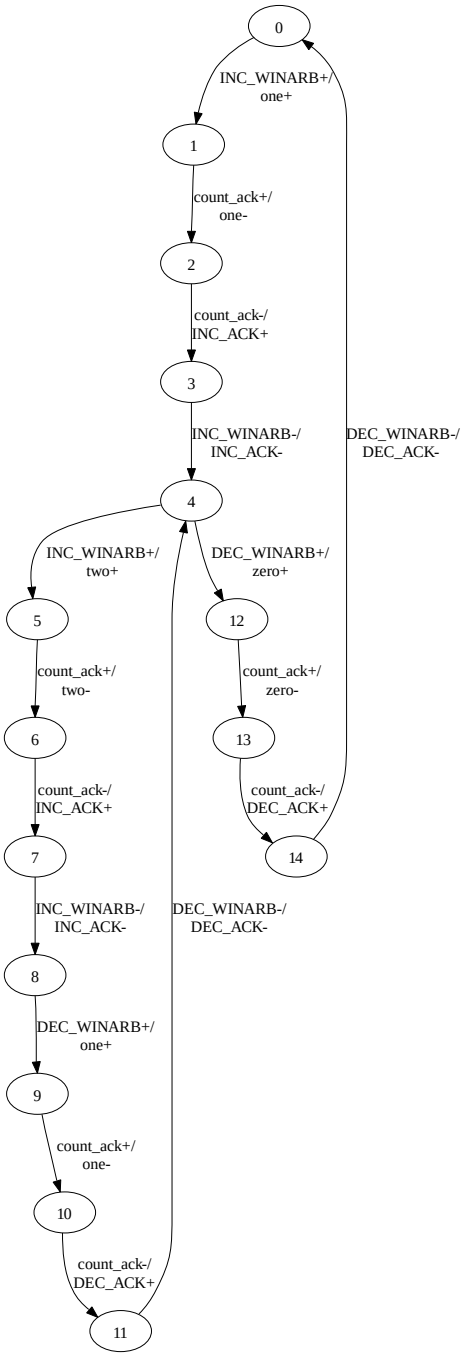

Figure B.1: BM specification for the low-controller BM machine: bi-modal adaptive delay line.

B.1 Bi-modal design without the MTBF bug fix

B.1.1 Low controller

The low-controller in the bi-modal design receives samples either entering or exiting the monitored 2-cell sub-segment of the delay line's first segment and provides a count of the total number of samples inside the sub-segment to the mid controller. The specification of the controller is shown in Fig. B.1.

At each operation, the controller first receives the entering/exiting sample, then performs a full 4-phase hand-shaking with the mid controller reporting the updated count, and when that is complete it also completes the remaining 3 phases of the 4-phase hand-shaking at the active input channel. A sample of this operation is states $0-4$: initially the count is 0 and first the entering sample arrives ($INC_W INARB+$); the controller updates the sample 0 to 1 and performs a 4-phase handshaking with the mid controller ($one+ \Rightarrow count_{ack}+ \Rightarrow one- \Rightarrow count_{ack}-$). Such a protocol, which first completes one 4-phase transaction after having received first a request on another channel, then completes handshaking at the original channel, is called "encapsulated 4-phase", [6].

The operation is identical for the other cases, with the only difference being the different reported counts.

B.1.2 Mid controller

There are two mid controllers, left and right. The left one is simple, and the right is an enhanced version of the left with increased capabilities.

The left controller's specification is shown in Fig. B.2. The left controller receives the low left counts and talks only to the top level at its output channel. For most counts from the low level, the mid left simply hand-shakes with the low level. However, for the count *sequence* of $0, 1, 2$ it reports to the top using a simple hand-shake.

As shown in Fig. B.2, in most cases the mid left simply acknowledges the low level. An example is the transition from state 0 to 2 and vice versa ($one+ \Rightarrow count_{ack}+ \Rightarrow one- \Rightarrow count_{ack}-$). In the special case (from state 2 to 7) where a *new* count of 2 appears, the mid left controller first does a full 4-phase hand-shake with the top ($two+ \Rightarrow Enter, left, req+ \Rightarrow Enter, left, ack+ \Rightarrow Enter, left, req-$) and then completes at the input channel the remaining 3 phases of the protocol ($count_{ack}+ \Rightarrow two- \Rightarrow count_{ack}-$).

The mid right controller, whose specification is the one in Fig. B.3, is more complicated. It has both the same functionality as the left mid, as we showed previously, as well as additional ones. When the mid right receives a new 2 count (coming from a 0, 1 sequence), it still reports to the top ($two+ \Rightarrow Enter, right, req+$), but now it can receive two types of acknowledgements. In the "normal" case (states (3-8) it simply completes the protocol; when the next 1 appears from the low controller the right mid simply acknowledges it. In the "special" case, the right mid de-asserts its request (but the protocol completes without the ack from the top being de-asserted). When the next 1 is reported from the low, the right mid sends to the top the special signal $slot, fine, grain$ (used for reconfiguration); the top de-asserts the special acknowledgment and the right mid then completes

the 4-phase at its input channel. The right mid also reports to the top (through $Exit, right$) when it sees a count sequence of $2, 1, 0$ from the low level. Again, first the mid hand-shakes with the top, and once that is completed it finishes the 4-phase transaction with the low level.

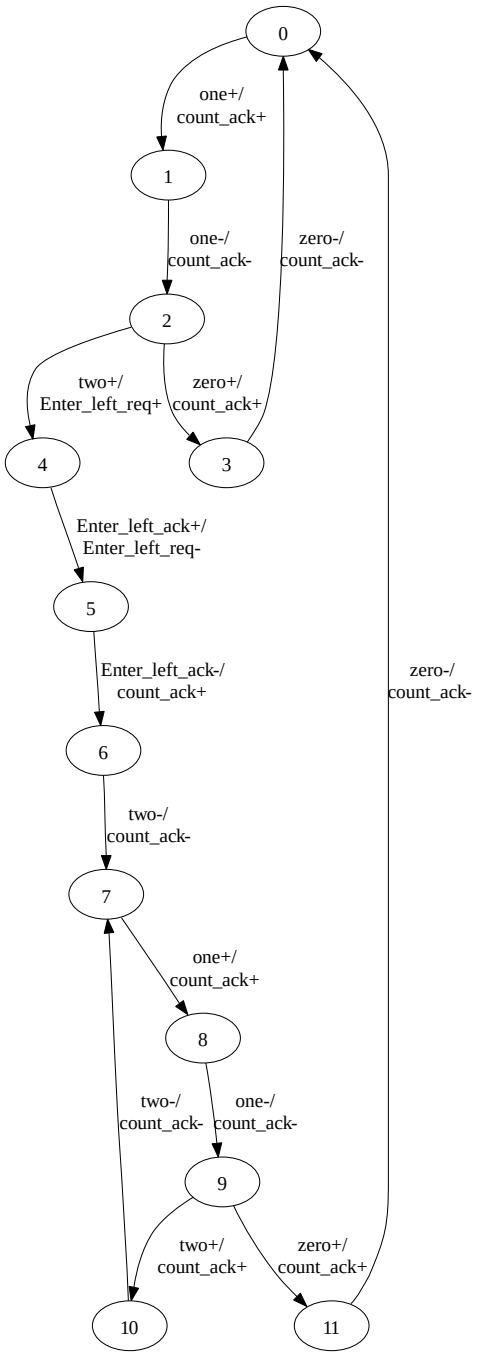

Figure B.2: BM specification for the left mid-controller BM machine: bi-modal adaptive delay line without the MTBF-bug fix.

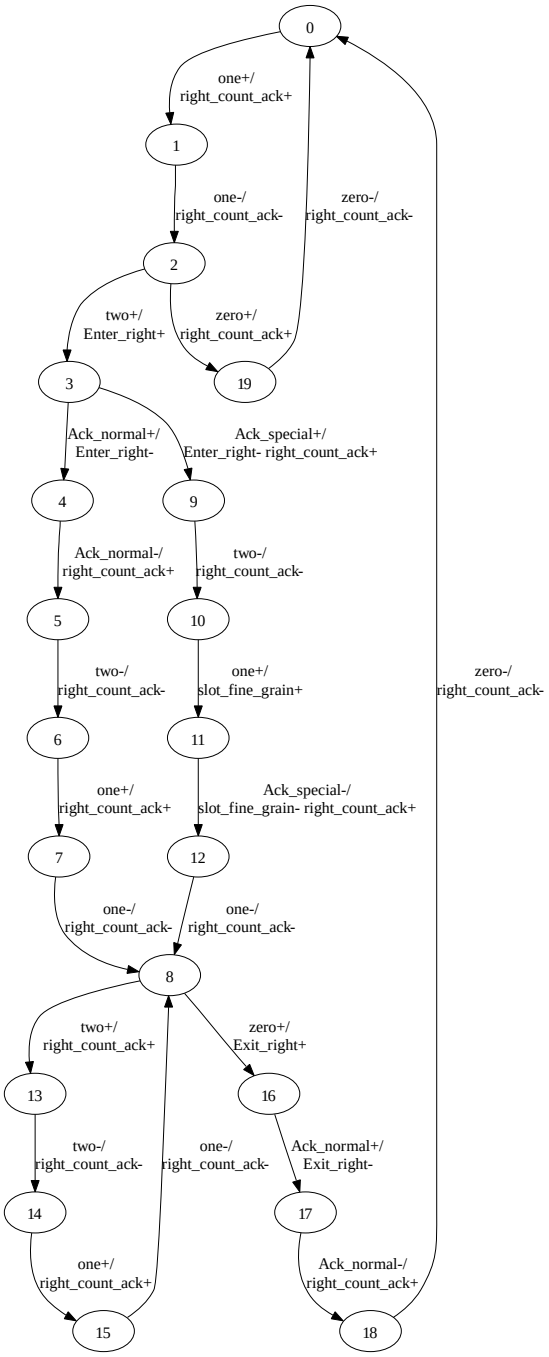

Figure B.3: BM specification for the right mid-controller BM machine: bi-modal adaptive delay line without the MTBF-bug fix.

B.1.3 Top level

The top controller takes the left and right mid information as input channels, and controls an UP/DOWN counter and sends granularity to the asynchronous control line through its output channels. Its specification is shown in Fig. B.4.

Starting from the initial state (where 0 high regions are inside segment 1), the first thing that happens is a new region entering from the left ($Enter, left, req+$). The controller first updates the count ($count, UP+ \Rightarrow count, isnot, zero+ \Rightarrow count, UP-$), and then completes the input transaction ($Enter, left, ack+ \Rightarrow Enter, left, req- \Rightarrow Enter, left, ack-$). Then more entering regions from the left can be reported, where the same thing happens. This is specified in states $0-7$.

The moment the right environment reports on an entering region ($Enter, right, req+$), the top controller is almost ready to reconfigure the delay line. It uses the special ack ($Right, special, ack+$), which causes $Enter, right, req-$ and this completes this transaction, without de-asserting the acknowledgement to the right. Immediately after that, the next thing to arrive is the reconfiguration signal $Slot, fine, grain+$: the top controller will immediately do the reconfiguration event at the 2-phase output channel ($Mode, req- \Rightarrow Mode, ack-$), and then it will acknowledge the input by just de-asserting the special ack (states $4, 8, 9, 10, 11, 12$).

After that, more entering or regions can arrive, treated in the same way as described above. So will the exiting regions which do not cause a count decrement to exactly zero (states $12, 16, 17, 18, 19$). If, however, the count is decremented to zero, then the top controller will first do a transaction with the control line ($Mode, req+ \Rightarrow Mode, ack+$), then it will acknowledge the right environment ($Right, normal, ack+ \Rightarrow Exit, right, req- \Rightarrow Right, normal, ack-$) before returning to the initial state, as specified in states $12, 17, 20, 21, 22$.

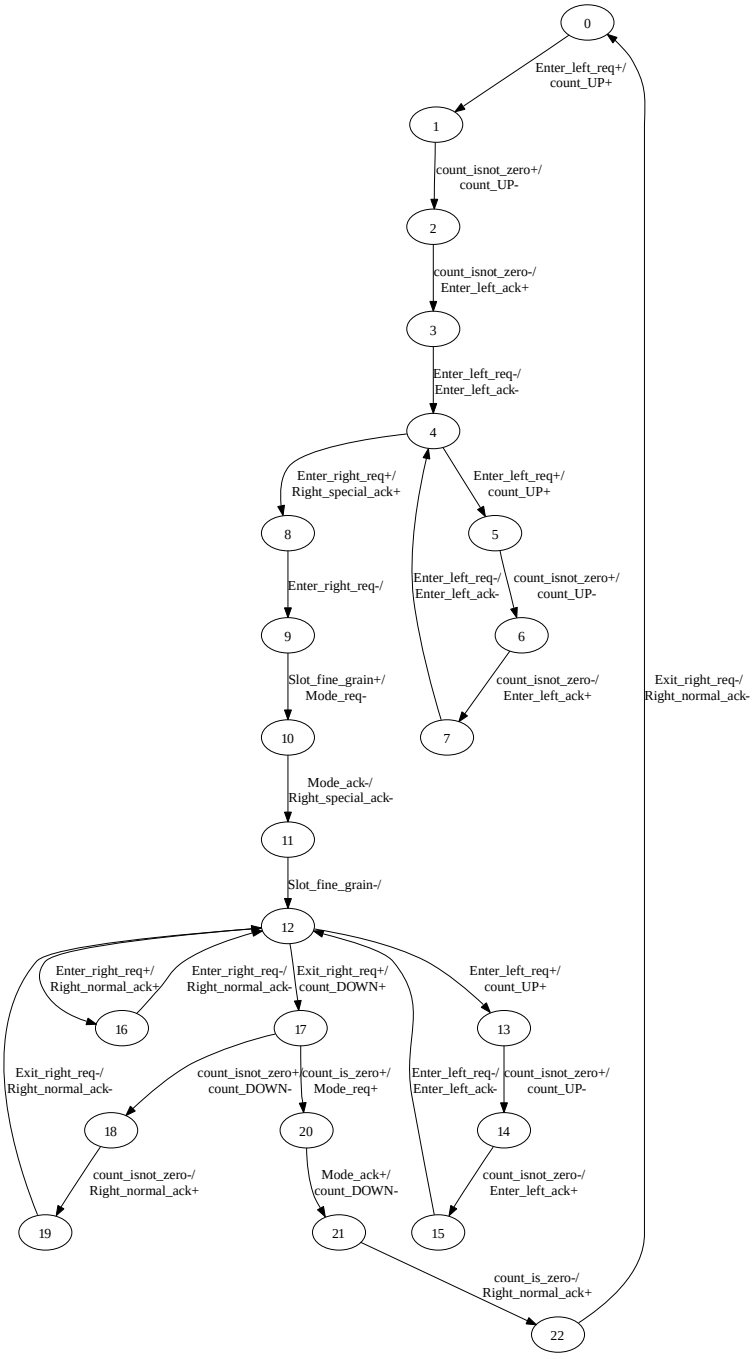

Figure B.4: BM specification for the top BM machine: bi-modal adaptive delay line without the MTBF-bug fix.

B.2 Bi-modal adaptive delay line with MTBF bug fix

In this case, the low level remains identical to the previous version. The mid reduces to just the left mid controller, and the top is slightly modified.

B.2.1 Mid controller

The left mid controller is a modified version of the old left mid. It performs the same functional operations as the *combination* of the left and right old mid controllers, but some events are reported to the top and some to the right environment (which is the asynchronous delay line).

The new left mid still reports on entering regions in the case of a low-level count sequence of $0,1,2$. However, it can be acknowledged in two different ways, similar to the old mid right controller. In the normal case (states $3-4-5-...-8$), no further operation is performed. In the special case (staes $3-9-10-...-8$) the special acknowledgement leads first to full completion of the 4-phase protocol (as opposed to the old mid right controller, where the protocol was left half-complete), followed by the full 4-phase at the input channel. Then, at the next 1 reported from the low level, a full 4-phase transaction is done at the right output channel (interfacing with the asynchronous delay line) and then the input hand-shaking completes. Finally, the sequence of $2,1,0$ from the low level is reported to the right output channel in the same way (encapsulated 4-phase).

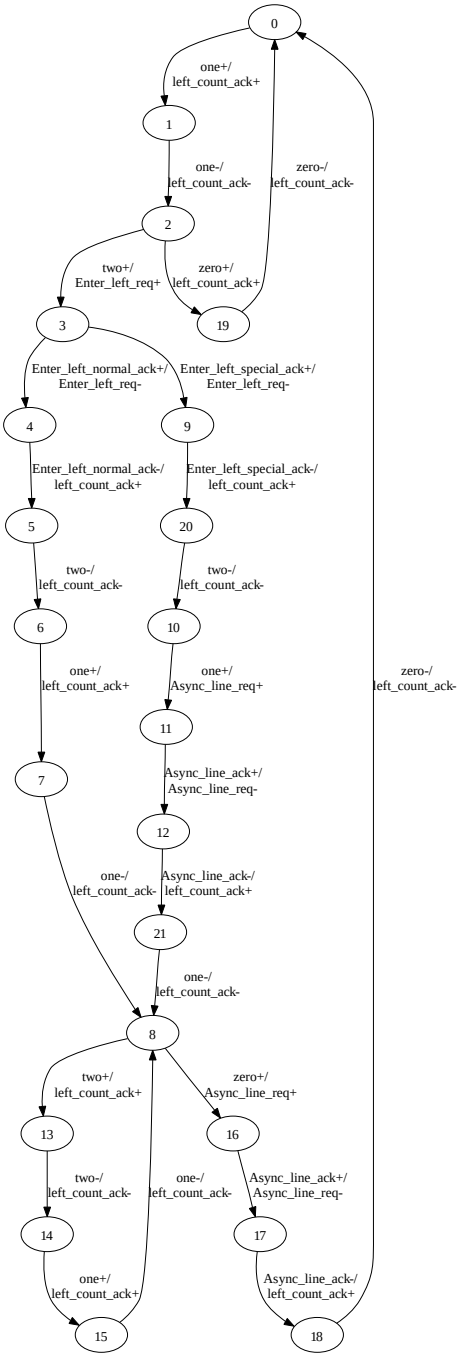

Figure B.5: BM specification for the left mid-controller BM machine: bi-modal adaptive delay line with MTBF-bug fix.

B.2.2 Top level

The top controller BM machine, shown in Fig. B.6, controls an UP/DOWN counter to count the total number of high regions, and sends reconfiguration control when required.

When receiving the first entering high region (states $0-4$), it first orders an "increment" operation to the UP/DOWN counter and then acknowledges by using the special ack. When receiving the second region and on, it instead uses the normal ack (states $4-5-6-7$ and $10-11-12-13$). Once it receives the "reconfiguration" event (states $4,8,9,10$) is does a transaction on its 2-phase output channel ($Mode, req- \Rightarrow Mode, ack-$).

When receiving exiting high regions, the controller orders the counter for a "decrement" event ("count,down"). If the response from the adder is that the updated count of high regions is still not zero (states $10,14,15,16$), nothing else is done but acknowledging the input channel. If the new count is zero, then fist a transaction is done at the 2-phase output channel ($Mode, req+ \Rightarrow Mode, ack+$) and then the input is acknowledged returning to the initial state.

B.3 Tri-modal adaptive delay line

This section shows how to enhance the previous controllers, intended for bi-modal operation, to incorporate the tri-modal example shown in chapter 7. The operation of all controllers is similar to the bi-modal ones. The key differences will be highlighted as each controller is presented.

B.3.1 Low level

The tri-modal low-level has two controllers. Both look at a 4-cell group (its input, output and mid-point): one controller counts the total number of samples inside the group, while the other the

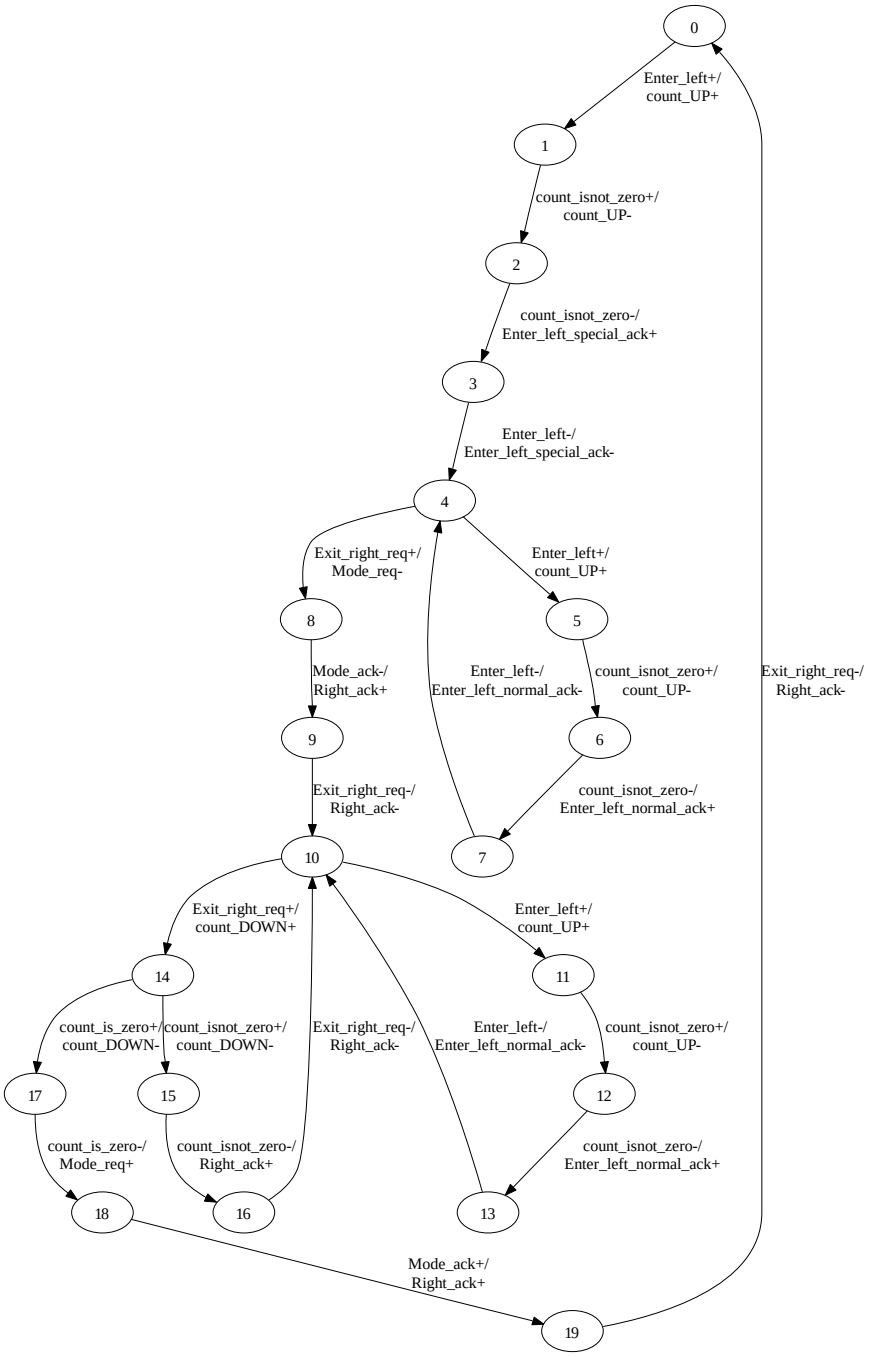

Figure B.6: BM specification for the top-controller BM machine: bi-modal adaptive delay line with MTBF-bug fix.

number in the second half.

The specification for the higher-level controller, looking at the entire group, is shown in Fig. B.7. It can now count up to four samples, so it's an extended (and larger) version of the one in B.1. It increments the count for "entering samples" (labeled *IN*) and decrements the count for exiting ones (OUT). The mid-point samples (MID) are simply acknowledged directly at the input side without changing the count or reporting to the mid level.

The small controller is also an enhanced version of the one in Fig. B.1. The only difference is that the sample count increases/decreases for mid-point/exiting samples (MID/OUT), while the entering samples (*IN*) are only acknowledged directly at the input side.

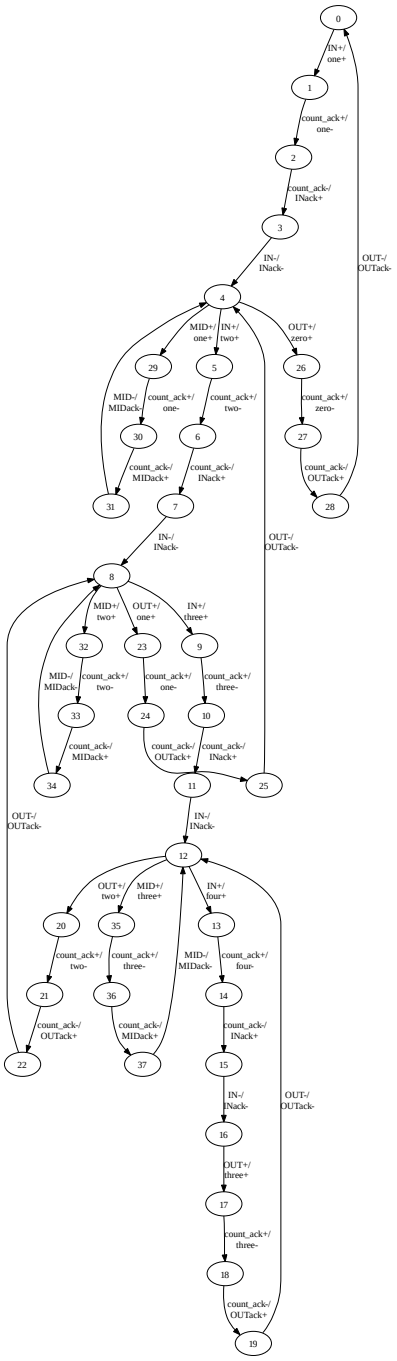

Figure B.7: BM specification for the low-controller BM machine: tri-modal adaptive delay line, "total" low controller.

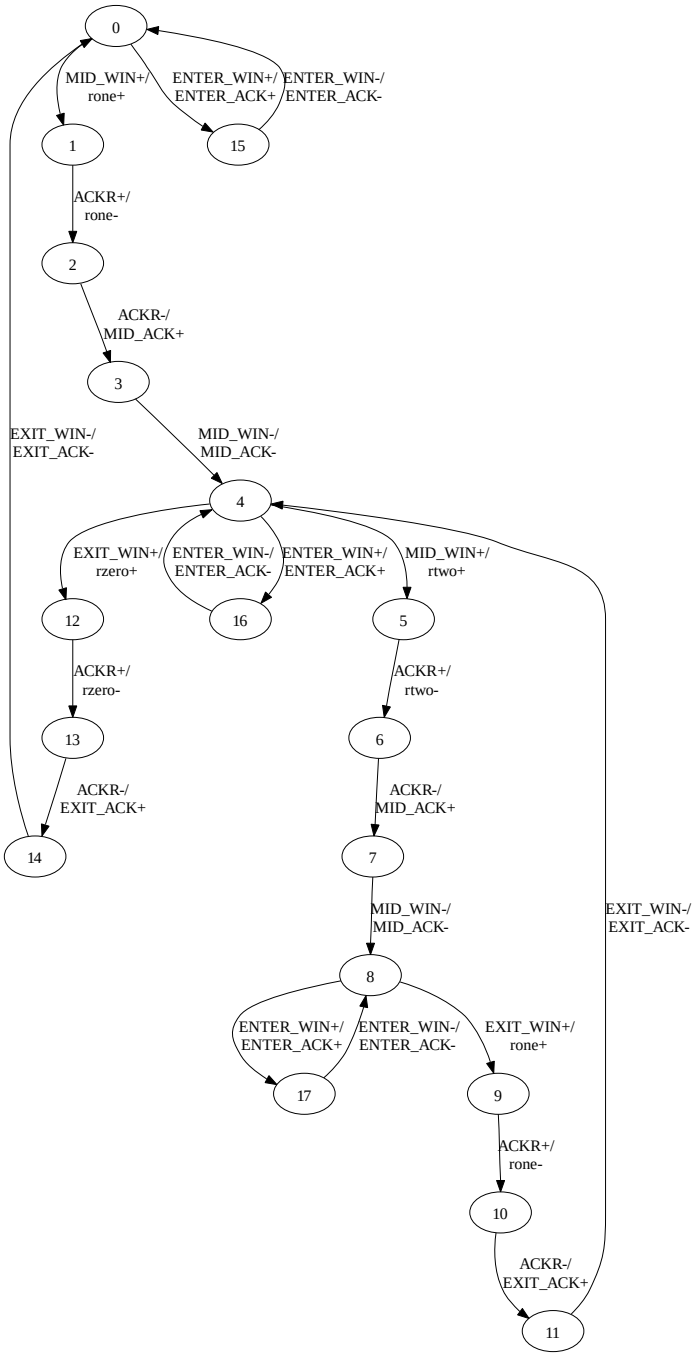

Figure B.8: BM specification for the low-controller BM machine: tri-modal adaptive delay line, "total" low controller.

B.3.2 Mid level

The new mid controller for tri-modal operation has two controller, one for high traffic looking at the output of the small low controller, and one for medium traffic looking at the big low controller. The former is identical to the one in Fig. B.5, and the latter is an extension of it as shown in Fig. B.9.

The new controller still reports entering and exiting regions for count sequences of $0,1,2$ and $2,1,0$ respectively. The only difference is that in the special case of the first region reported, for which the 4-phase transaction with the top level is completed using the special ack, the mid controller attempts to find the *first* sample exiting from the quad group. This is indicated as the first *decrement* in the count reported from the low controller, which can lead to more than one count sequences. This operation corresponds to states $17-21$: when the first exiting sample is found (by keeping track of the previous count for comparison), first the 4-phase at the input is completed and *then* the transaction is made at the output channel. This use of *serial* 4-phase protocol is a differentiation from the remaining controllers, which used the encapsulated version; the reason behind this choice is that the serial version minimizes the number of states for the controller.

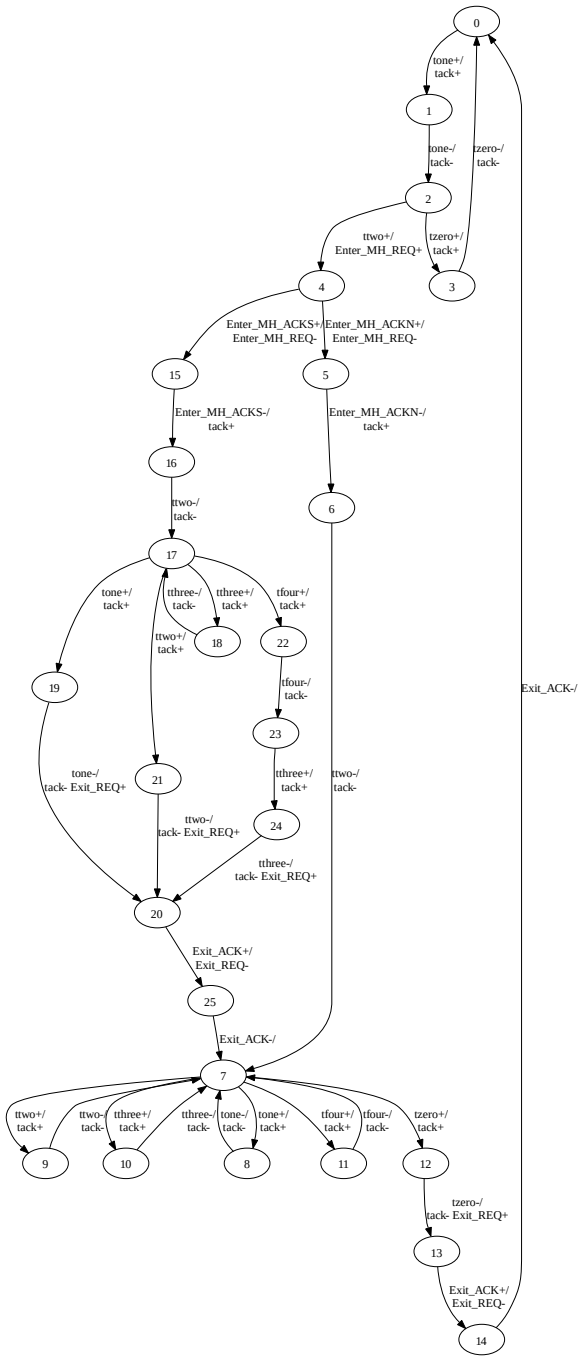

Figure B.9: BM specification for the mid-controller BM machine for medium-high traffic: tri-modal adaptive delay line.

B.3.3 Top controller

The top controller involves three BM machines: two of them for controlling the two UP/DOWN counters (keeping track of the number of high and medium regions respectively), and the "combine" module. The former are two identical copies of the controller in B.6.

The "combine" module is shown in Fig. B.10. It talks to both of the other two top controllers, each of which provides a binary decision on traffic (high vs. medium-low and low vs. medium-high), and combines the two binary choices for a tri-modal one, which is reported to the asynchronous control line. All interfaces here are 2-phase.

The operation of this controller is very simple. Once it receives a "high" setting from the high-top controller (states $0-7-11-5$), it immediately sends to the output the 2-bit control for high-grain setting in the delay line using a 2-phase transaction with two requests and two acknowledgments ($HIGH- \Rightarrow MODE, B0, REQ- \ MODE, B1, REQ- \ MODE, B0, ACK- MODE, B1, ACK-$). Once this is completed, it acknowledges the input channel ($HIGH, ACK-$).

A similar operation is performed for the medium setting: when receiving a medium-high event without first having received a high one, the combine module sends to the asynchronous line the medium control setting ($HIGH- \Rightarrow MODE, B0, REQ- \ MODE, B0, ACK-$) and then acknowledges the input. If the high event has been received before the medium-high, the combine module simply acknowledges the latter without doing an output transaction.

If the controller is already in high setting, then the "exit-high" even ($HIGH+$) will cause first the transition of the state to medium-high ($HIGH+ \Rightarrow -MODE, B1, REQ \Rightarrow MODE, B1ACK-$), then the acknowledgement at the input side (states (5-8-9). When finally the medium-exit event is also received, the module will follow the same procedure (first a transaction at the output, then

acknowledgement at the input) to return to the initial state.

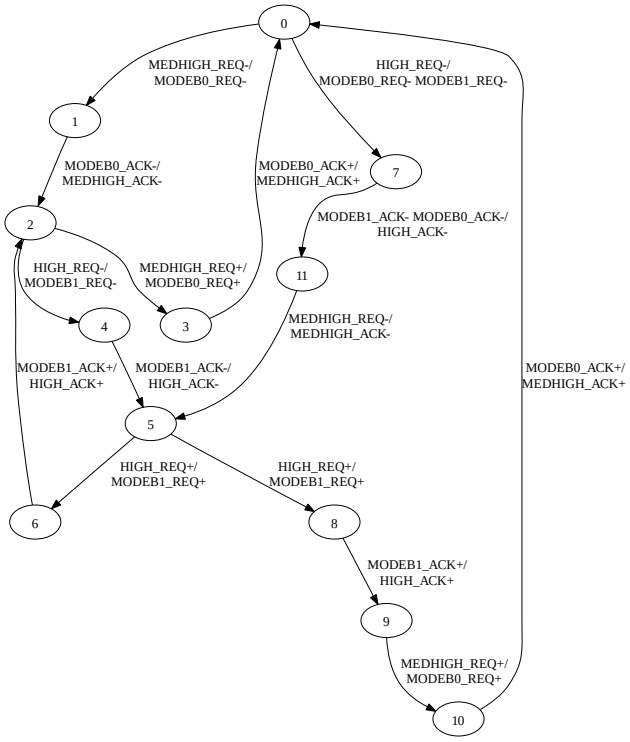

Figure B.10: BM specification for the top-controller's "combine" module: tri-modal adaptive delay line.

B.4 Summary

This appendix presented the specification and implementation of all the BM asynchronous controllers used for the design of the mode controller in the adaptive delay line. The section shows all controllers (low, mid and top) for both design paradigms of Chapter 7. The specifications of the controllers is straight-forward and is simply derived from the combination of the controllers'

functionality and the communication protocol used, which was mostly 4-phase, with the exception of some 2-phase designs. The controller specifications are easy to extend to implement different traffic thresholds for the adaptive delay line. The general approach is indicated by the direct way the bi-modal controllers were extended to the tri-modal ones, especially the low-level controllers.

Bibliography

[1] S. Mitra and Y. Kuo, *Digital Signal Processing: A Computer-Based Approach*, vol. 2. McGraw-Hill New York, 2006.

[2] V. Madisetti and V. Madisetti, *VLSI Digital Signal Processors*. Butterworth-Heinemann, 1995.

[3] S.-M. Kuo and W.-S. Gan, *Digital Signal Processors: Architectures, Implementations, and Applications*. Prentice Hall, 2005.

[4] Y. Tsividis, "Digital signal processing in continuous time: a possibility for avoiding aliasing and reducing quantization error," in *Proc. IEEE Int. Conf. on Acoustins, Speech and Signal Processing (ICASSP), 2004*, pp. 589–592, 2004.

[5] Y. Tsividis, "Event-driven, continuous-time ADCs and DSPs for adapting power dissipation to signal activity," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pp. 3581–3584, 2010.

[6] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design*. Kluwer Academic Publishers, 2001.

[7] S. Hauck, "Asynchronous design methodologies: An overview," *Proceedings of the IEEE*, vol. 83, no. 1, pp. 69–93, 1995.

[8] H. Nyquist, "Certain topics in telegraph transmission theory," *Trans. of the American Institute of Electrical Engineers*, vol. 47, no. 2, pp. 617–644, 1928.

[9] H. S. Black and J. Edson, "Pulse code modulation," *Trans. of the American Institute of Electrical Engineers*, vol. 66, no. 1, pp. 895–899, 1947.

[10] D. G. Holmes and T. A. Lipo, *Pulse Width Modulation For Power Converters: Principles And Practice*, vol. 18. Wiley-IEEE Press, 2003.

[11] P. M. Aziz, H. V. Sorensen, *et al.*, "An overview of sigma-delta converters," *IEEE Signal Processing Magazine*, vol. 13, no. 1, pp. 61–84, 1996.

[12] J. Candy and O. Benjamin, "The structure of quantization noise from sigma-delta modulation," *IEEE Trans. on Communications*, vol. 29, no. 9, pp. 1316–1323, 1981.

[13] H. S. Black, *Modulation Theory*. Van Nostrand, 1953.

[14] J. Das and P. Sharma, "Some asynchronous pulse-modulation systems," *Electronics Letters*, vol. 3, no. 6, pp. 284–286, 1967.

[15] D. F. Hoeschele, *Analog-to-Digital, Digital-to-Analog Conversion Techniques*. Wiley New York, 1968.

[16] C. Inacio and D. Ombres, "The DSP decision: fixed point or floating?," *IEEE Spectrum*, vol. 33, no. 9, pp. 72–74, 1996.

[17] Q. Wu, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," *IEEE Trans. Circuits and Systems I*, vol. 47, no. 3, pp. 415–420, 2000.

[18] D. Brooks and M. Martonosi, "Value-based clock gating and operation packing: dynamic strategies for improving processor power and performance," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 2, pp. 89–126, 2000.

[19] W. Lee, P. Landman, B. Barton, S. Abiko, H. Takahashi, H. Mizuno, S. Muramatsu, K. Tashiro, M. Fusumada, L. Pham, F. Boutaud, E. Ego, G. Gallo, H. Tran, C. Lemonds, A. Shih, M. Nandakumar, B. Eklund, and I.-C. Chen, "A 1 V DSP for wireless communications," in *IEEE Solid-State Circuits Conf., Digest of Technical Papers*, pp. 92–93, IEEE, 1997.

[20] H. Jacobson, P. Bose, Z. Hu, A. Buyuktosunoglu, V. Zyuban, R. Eickemeyer, L. Eisen, J. Griswell, D. Logan, B. Sinharoy, and J. Tendler, "Stretching the limits of clock-gating efficiency in server-class processors," in *11th Int. Symp. on High-Performance Computer Architecture, (HPCA-11).*, pp. 238–242, 2005.

[21] A. Strollo, E. Napoli, and D. De Caro, "New clock-gating techniques for low-power flip-flops," in *Proc. ACM Int. Symp. on Low Power Electronics and Design*, pp. 114–119, 2000.

[22] L. Karam, I. AlKamal, A. Gatherer, G. Frantz, D. Anderson, and B. Evans, "Trends in multicore DSP platforms," *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 38–49, 2009.

[23] T. Ogunfunmi and M. Narasimha, *Principles of Speech Coding*. CRC Press, Inc., 2010.

[24] R. H. Katz and G. Borriello, *Contemporary Logic Design*. Pearson Prentice Hall, 2005.

[25] Y. Tsividis, "Continuous-time digital signal processing," *Electronics Letters*, vol. 39, pp. 1151–1152, Oct. 2003.

[26] Y. Tsivididis, "Event-driven data acquisition and digital signal processing: a tutorial," *IEEE Trans. on Circuits and Systems - II*, vol. 57, no. 8, pp. 577–581, 2010.

[27] B. Schell and Y. Tsividis, "Analysis of continuous-time digital signal processors," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pp. 2232–2235, 2007.

[28] D. Bruckmann, T. Faldengut, B. Hosticka, R. Kokzinski, K. Konrad, and N. Tavangaran, "Optimization and implementation of continuous time DSP systems by using granularity reduction," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pp. 410–413, 2011.

[29] F. Aeschlimann, E. Allier, L. Fesquet, and M. Renaudin, "Asynchronous FIR filters: towards a new digital processing chain," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pp. 198–206, 2004.

[30] F. Akopyan, R. Manohar, and A. Apsel, "A level-crossing flash asynchronous analog-to-digital converter," in *Proc. IEEE Int. Symp. on Asynchronous Circuits and Systems (ASYNC)*, pp. 11–22, 2006.

[31] C. Weltin-Wu and Y. Tsividis, "An event-driven, alias-free adc with signal-dependent resolution," in *Proc. IEEE Symp. on VLSI Circuits (VLSIC)*, pp. 28–29, 2012.

[32] E. Allier, J. Goulier, G. Sicard, A. Dezzani, E. Andre', and M. Renaudin, "A 120nm low power asynchronous ADC," in *Proc. ACM Int.Symp on Low Power Electronics and Design (ISLPED)*, pp. 60–65, 2005.

[33] B. Schell and Y. Tsividis, "A continuous-time ADC/DSP/DAC system with no clock and with activity-dependent power dissipation," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 43, no. 11, pp. 2472–2481, 2008.

[34] M. Kurchuk, C. Weltin-Wu, D. Morche, and Y. Tsividis, "Event-driven GHz-range continuous-time digital signal processor with activity-dependent power dissipation," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 47, no. 9, pp. 2164–2173, 2012.

[35] T.-T. Liu and J. Rabaey, "A 0.25 v 460nw asynchronous neural signal processor with inherent leakage suppression," in *Proc. IEEE Symp. on VLSI Circuits (VLSIC)*, pp. 158–159, IEEE, 2012.

[36] C. Vezyrtzis, W. Jiang, S. Nowick, and Y. Tsividis, "A flexible, clockless digial filter," in *In Proc. IEEE Eur. Solid-State Circ. Conf. (ESSCIRC)*, 2013.

[37] C. Vezyrtzis, Y. Tsividis, and S. M. Nowick, "Designing pipelined delay lines with dynamically-adaptive granularity for low-energy applications," in *Proc. IEEE Int. Conf. on Computer Design (ICCD)*, pp. 329–336, 2012.

[38] C. Vezyrtzis, Y. Tsividis, and S. M. Nowick, "Designing multimodal delay lines with adaptive granularity for low-energy applications," *under review, IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 2013.

[39] J. Mark and T. Todd, "A nonuniform sampling approach to data compression," *IEEE Trans. on Communications*, vol. 29, no. 1, pp. 24–32, 1981.

[40] C. Vezyrtzis and Y. Tsividis, "Processing of signals using level-crossing sampling," in *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. 2293–2296, 2009.

[41] R. M. Dolby, "Compressor and expander circuits," 1973. US Patent 3,775,705.

[42] Y. Tsividis, "Externally linear time-invariant systems and their application to companding signal processors," *IEEE Trans. Circuits and Systems II*, vol. 44, pp. 65–85, Feb. 1997.

[43] A. Klein and Y. Tsividis, "Externally linear time invariant digital signal processors," *IEEE Trans. on Signal Processing*, vol. 58, pp. 4897–4909, Sept 2010.

[44] C. Vezyrtzis, A. Klein, D. Ellis, and Y. Tsividis, "Direct processing of MPEG audio using companding and BFP techniques," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 361–364, 2011.

[45] Z. Zhao and A. Prodic, "Continuous-time digital controller for high-frequency DC-DC converters," *IEEE Transactions on Power Electronics*, vol. 23, pp. 564–573, Mar. 2008.

[46] Y. W. Li, K. L. Shepard, and Y. P. Tsividis, "A continuous-time programmable digital fir filter," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 41, no. 11, pp. 2512–2520, 2006.

[47] T. Beyrouthy and L. Fesquet, "An event-driven FIR filter: Design and implementation," in *Proc. IEEE Int. Symp. on Rapid System Prototyping (RSP)*, pp. 59–65, 2011.

[48] M. Trakimas and S. Sokunsale, "An adaptive resolution asynchronous ADC architecture for data compression in energy constrained sensing applications," *IEEE TCAS-I*, vol. 58, pp. 921–934, May 2011.

[49] S.-W. Chen and R. W. Brodersen, "A 6-bit 600-ms/s 5.3-mw asynchronous ADC in 0.13μm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 12, pp. 2669–2680, 2006.

[50] T. Wang, D. Wang, P. Hursts, B. Levy, and S. Lewis, "A level-crossing analog-to-digital converter with triangular dither," *IEEE Ttrans. on Circuits and Systems I*, vol. 56, pp. 2089–2099, Sep. 2009.

[51] W. Tang, A. Osman, D. Kim, B. Goldstein, C. Huang, B. Martini, V. Pieribone, and E. Culurciello, "Continuous time level crossing sampling adc for bio-potential recording systems," *IEEE Trans. on Circuits and Systems-I*, vol. 60, no. 99, pp. 1–12, 2012.

[52] D. Grimaldi, "A 10-bit 5khz level-crossing ADC," in *Proc. IEEE European Conf. on Circuit Theory and Design (ECCTD)*, pp. 564–567, 2011.

[53] S. Qaisar, L. Fesquet, and M. Renaudin, "Spectral analysis of a signal driven sampling scheme," in *Proc. European Signal Processing Conference (EUSIPCO)*, 2006.

[54] M. Kurchuk and Y. Tsividis, "Signal-dependent variable-resolution clockless A/D conversion with application to continuous-time digital signal processing," *IEEE Trans. on Circuits and Systems*, vol. 57, no. 5, pp. 982–991, 2010.

[55] K. Guan and A. Singer, "Sequential placement of reference levels in a level-crossing analog-to-digital converter," in *Proc. IEEE Annual Conf. on Information Sciences and Systems*, pp. 464–469, 2008.

[56] B. Fesquet and L. Fesquet, "A fully non-uniform approach to FIR filtering," in *Proc. Int. Conf. on Sampling Theory and Applications (SAMPTA)*, 2009.

[57] D. Bruckmann, "Optimization of continuous time filters by delay line adjustment," in *Proc. IEEE Int. Midwest Symp. on Circuits and Systems*, pp. 1238–1241, 2010.

[58] N. Tavangaran, D. Bruckmann, R. Kokozinski, and K. Konrad, "Continuous time digital systems with asynchronous sigma delta modulation," in *Proc. IEEE European Signal Processing Conference (EUSIPCO)*, pp. 225–229, 2012.

[59] C. Van Berkel, M. B. Josephs, and S. M. Nowick, "Applications of asynchronous circuits," *Proceedings of the IEEE*, vol. 87, no. 2, pp. 223–233, 1999.

[60] P. Beerel, G. Dimou, and A. Lines, "Proteus: An ASIC flow for ghz asynchronous designs," *IEEE Design and Test of Computers*, vol. 28, pp. 36–51, Sept.–Oct. 2011.

[61] M. Singh, J. Tierno, A. Rylyakov, S. Rylov, and S. Nowick, "An adaptively-pipelined mixed synchronous-asynchronous digital FIR filter chip operating at 1.3 gigahertz," in *Proc. IEEE Int. Symp. on Asynchronous Circuits and Systems*, pp. 84–95, 2002.

[62] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, 1989.

[63] M. N. Horak, S. M. Nowick, M. Carlberg, and U. Vishkin, "A low-overhead asynchronous interconnection network for GALS chip multiprocessors," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 494–507, 2011.

[64] S. M. Nowick and D. L. Dill, "Automatic synthesis of locally-clocked asynchronous state machines," in *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 318–321, 1991.

[65] G. Gill, S. Attarde, G. Lacourba, and S. Nowick, "A low-latency adaptive asynchronous interconnection network using bi-modal router nodes," in *Proc. ACM Int. Symp. on Networks on Chip (NoCS)*, pp. 193–200, 2011.

[66] J. Teife and R. Manohar, "Programmable asynchronous pipeline arrays," in *Field Programmable Logic and Application*, pp. 345–354, Springer, 2003.

[67] K. Van Berkel, R. Burgess, J. Kessels, M. Roncken, F. Schalij, and A. Peeters, "Asynchronous circuits for low power: A DCC error corrector," *IEEE Design & Test of Computers*, vol. 11, no. 2, pp. 22–32, 1994.

[68] C. Van Berkel, M. B. Josephs, and S. M. Nowick, "Applications of asynchronous circuits," *Proceedings of the IEEE*, vol. 87, no. 2, pp. 223–233, 1999.

[69] S. M. Nowick and M. Singh, "High-performance asynchronous pipelines: an overview," *IEEE Design & Test of Computers*, vol. 28, pp. 8–22, 2011.

[70] C. Molnar *et al.*, "Counterflow pipeline processor architecture," *Sun Microsystems technical report*, 1994.

[71] J. Teifel and R. Manohar, "An asynchronous dataflow FPGA architecture," *IEEE Trans. on Computers*, vol. 53, no. 11, pp. 1376–1392, 2004.

[72] L. S. Nielsen and J. Sparsø, "Designing asynchronous circuits for low power: An IFIR filter bank for a digital hearing aid," *Proceedings of the IEEE*, vol. 87, no. 2, pp. 268–281, 1999.

[73] R. Benes, S. M. Nowick, and A. Wolfe, "A fast asynchronous Huffman decoder for compressed-code embedded processors," in *Proc. IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, pp. 43–56, 1998.

[74] S. M. Nowick, "Design of a low-latency asynchronous adder using speculative completion," *Proc. IEE Computers and Digital Techniques*, vol. 143, no. 5, pp. 301–307, 1996.

[75] M. Singh and S. M. Nowick, "Mousetrap: ultra-high-speed transition-signaling asynchronous pipelines," in *Proc. IEEE Int. Conf. on Computer Design (ICCD)*, pp. 9–17, 2001.

[76] M. Singh and S. M. Nowick, "Asynchronous pipeline with latch controllers," Oct. 25 2005. US Patent 6,958,627.

[77] R. O. Ozdag and P. A. Beerel, "High-speed QDI asynchronous pipelines," in *Proc. IEEE Int. Symp. on Asynchronous Circuits and Systems*, pp. 13–22, 2002.

[78] R. Kol and R. Ginosar, "A doubly-latched asynchronous pipeline," in *Proc. IEEE Int. Conf. on Computer Design*, pp. 706–711, 1997.

[79] A. E. Sjogren and C. J. Myers, "Interfacing synchronous and asynchronous modules within a high-speed pipeline," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 5, pp. 573–583, 2000.

[80] E. Beigné and P. Vivet, "Design of on-chip and off-chip interfaces for a GALS NoC architecture," in *Proc. IEEE Int. Symp. on Asynchronous Circuits and Systems*, pp. 10–pp, 2006.

[81] L. Plana, S. Furber, S. Temple, M. Khan, Y. Shi, J. Wu, and S. Yang, "A GALS infrastructure for a massively parallel multiprocessor," *IEEE Design & Test of Computers*, vol. 24, no. 5, pp. 454–463, 2007.

[82] M. Singh and M. Theobald, "Generalized latency-insensitive systems for gals architectures," *Proc. of FMGALS*, vol. 3, 2003.

[83] M. N. Horak, S. M. Nowick, M. Carlberg, and U. Vishkin, "A low-overhead asynchronous interconnection network for gals chip multiprocessors," in *Fourth ACM/IEEE International Symposium onNetworks-on-Chip (NOCS)*, pp. 43–50, IEEE, 2010.

[84] T. Chelcea and S. M. Nowick, "Robust interfaces for mixed-timing systems with application to latency-insensitive protocols," in *Proc. ACM Design Automation Conference*, pp. 21–26, 2001.

[85] T. Chelcea and S. Nowick, "Robust interfaces for mixed-timing systems," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 8, pp. 857–873, 2004.

[86] A. Mitra, W. F. McLaughlin, and S. M. Nowick, "Efficient asynchronous protocol converters for two-phase delay-insensitive global communication," in *Proc. IEEE Int. Symp. on Asynchronous Circuits and Systems, (ASYNC)*, pp. 186–195, 2007.

[87] P. B. McGee, M. Y. Agyekum, M. A. Mohamed, and S. M. Nowick, "A level-encoded transition signaling protocol for high-throughput asynchronous global communication," in *Proc. IEEE Int. Symp. on Asynchronous Circuits and Systems, (ASYNC)*, pp. 116–127, 2008.

[88] C. LaFrieda, B. Hill, and R. Manohar, "An asynchronous FPGA with two-phase enable-scaled routing," in *Proc. IEEE Int. Symp. on Asynchronous Circuits and Systems (ASYNC)*, pp. 141–150, 2010.

[89] T. Verhoeff, "Delay-insensitive codes: an overview," *Distributed Computing*, vol. 3, no. 1, pp. 1–8, 1988.

[90] R. Fuhrer and S. Nowick, *Sequential Optimization of Asynchronous and Synchronous Finite-State Machines: Algorithms and Tools*. Kluwer Academic Publishers, 2001.

[91] L.Lavagno and S. Nowick, *Asynchronous control circuits*. Springer, 2002.

[92] "http://www.cs.columbia.edu/ nowick/asynctools/.".

[93] R. M. Fuhrer, S. M. Nowick, M. Theobald, N. K. Jha, B. Lin, and L. Plana, "Minimalist: an environment for the synthesis, verification and testability of burst-mode asynchronous machines," *Department of Computer Science, Columbia University*, 1999.

[94] S. Nowick and D. Dill, "Automatic synthesis of locally-clocked asynchronous state machines," in *Proc. ICCAD*, pp. 318–321, 1991.

[95] S. M. Nowick, K. Y. Yun, and D. L. Dill, "Practical asynchronous controller design," in *Proc. IEEE Int. Conf. on Computer Design (ICCD)*, pp. 341–345, 1992.

[96] M. Theobald, S. M. Nowick, and T. Wu, "Espresso-HF: a heuristic hazard-free minimizer for two-level logic," in *Proc. ACM Design Automation Conference (DAC)*, pp. 71–76, 1996.

[97] A. Marshall, B. Coates, and F. Siegel, "Designing an asynchronous communications chip," *IEEE Design & Test of Computers*, vol. 11, no. 2, pp. 8–21, 1994.

[98] B. Coates, A. Davis, and K. Stevens, "The post office experience: designing a large asynchronous chip," *Integration, the VLSI journal*, vol. 15, no. 3, pp. 341–366, 1993.

[99] A. Davis, B. Coates, and K. Stevens, "The post office experience: designing a large asynchronous chip," in *Proc. of the Twenty-Sixth Hawaii International Conference*, vol. 1, pp. 409–418, 1993.

[100] T. Chelcea and S. M. Nowick, "Low-latency asynchronous FIFOs using token rings," in *Proc. IEEE Int. Symp. on Asynchronous Circuits and Systems (ASYNC)*, pp. 210–220, 2000.

[101] F. Marvasti, *Nonuniform Sampling: Theory And Practice*. Springer, 2001.

[102] K. "Guan and A. C. Singer, "A level-crossing sampling scheme for non-bandlimited signals," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 3, pp. III–III, 2006.

[103] N. Sayiner, H. V. Sorensen, and T. R. Viswanathan, "A level-crossing sampling scheme for A/D conversion," *IEEE Trans. on Circuits and Systems II*, vol. 43, no. 4, pp. 335–339, 1996.

[104] F. Akopyan, R. Manohar, and A. B. Apsel, "A level-crossing flash asynchronous analog-to-digital converter," in *Proc. IEEE Int. Symp. on Asynchronous Circuits and Systems (ASYNC)*, pp. 11–pp, 2006.

[105] R. Staszewski, K. Muhammad, D. Leipold, C.-M. Hung, Y.-C. Ho, J. Wallberg, C. Fernando, K. Maggio, R. Staszewski, T. Jung, J. Koh, S. John, I. Y. Deng, V. Sarda, O. Moreira-Tamayo, V. Mayega, R. Katz, O. Friedman, O. Eliezer, E. de Obaldia, and P. Balsara, "All-digital TX frequency synthesizer and discrete-time receiver for Bluetooth Radio in 130-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 12, pp. 2278–2291, 2004.

[106] J. Yen, "On nonuniform sampling of bandwidth-limited signals," *IRE Transactions on Circuit Theory*, vol. 3, no. 4, pp. 251–257, 1956.

[107] F. J. Beutler, "Error-free recovery of signals from irregularly spaced samples," *Siam Review*, vol. 8, no. 3, pp. 328–335, 1966.

[108] K. Yao and J. Thomas, "On some stability and interpolatory properties of nonuniform sampling expansions," *IEEE Trans. on Circuit Theory*, vol. 14, no. 4, pp. 404–408, 1967.

[109] T. Strohmer, "Numerical analysis of the non-uniform sampling problem," *Journal of computational and applied mathematics*, vol. 122, no. 1, pp. 297–316, 2000.

[110] P. Butzer, H. G. Feichtinger, and K. Grōchenig, "Error analysis in regular and irregular sampling theory," *Applicable Analysis*, vol. 50, no. 3-4, pp. 167–189, 1993.

[111] B. Schell, "Continuous-time digital signal processors: analysis and implementation," *Ph.D. thesis, Department of Electrical Engineering, Columbia University*, 2008.

[112] D. E. Muller and W. Bartky, "A theory of asynchronous circuits," in *Proc. Int. Symp. on the Theory of Switching*, vol. 29, pp. 204–243, 1959.

[113] C. L. Seitz, "System timing," *Introduction to VLSI systems*, pp. 218–262, 1980.

[114] A. Hajimiri, S. Limotyrakis, and T. Lee, "Jitter and phase noise in ring oscillators," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 6, pp. 790–804, 1999.

[115] A. Agarwal, S. Mathew, S. Hsu, M. Anders, H. Kaul, F. Sheikh, R. Ramanarayanan, S. Srinivasan, R. Krishnamurthy, and S. Borkar, "A 320mv-to-1.2 V on-die fine-grained reconfigurable fabric for DSP/media accelerators in 32nm CMOS," in *IEEE Solid-State Circuits Conf., Digest of Technical Papers (ISSCC), 2010 IEEE International*, pp. 328–329, 2010.

[116] A. Waizman, "A delay line loop for frequency synthesis of de-skewed clock," in *IEEE Solid-States Circuits Conf., Digest of Technical Papers*, pp. 298–299, 1994.

[117] P. Raha, S. Randall, R. Jennings, B. Hemick, A. Amerasekera, and B. Haroun, "A robust digital delay line architecture in a 0.13 um CMOS technology node for reduced design and process sensitivities," in *Proc. Int. Symp. on Quality Electronic Design*, pp. 148–153, 2002.

[118] Y. Jeon, J. Lee, H. Lee, K. Jin, K. Min, J. Chung, and H. Park, "A 66-333 MHz 12-mW register-controlled DLL with a single delay line and adaptive-duty-cycle clock dividers for production DDR SDRAMs," *IEEE Journal Solid-State Circuits*, vol. 39, pp. 2087–2092, Nov. 2004.

[119] M. Rawat, K. Rawat, and F. Ghannouchi, "Adaptive digital predistortion of wireless power amplifiers/transmitters using dynamic real-valued focused time-delay line neural networks," *IEEE Trans. on Microwave Theory and Techniques*, vol. 58, pp. 95–104, Jan. 2010.

[120] B. Patella, A. Prodic, A. Zieger, and D. Maksimovic, "High frequency digital PWM controller IC for DC-DC converters," *IEEE Trans. on Power Electronics I*, vol. 18, pp. 438–446, Jan. 2003.

[121] M. Johnson and E. Hudson, "A variable delay line PLL for CPU-coprocessor synchronization," *IEEE Journal of Solid-State Circuits*, vol. 23, pp. 1218–1223, Oct. 1998.

[122] G. Li, Y. Tousi, A. Hassibi, and E. Afshari, "Delay-line-based analog-to-digital converters," *IEEE Trans. on Circuits and Systems II*, vol. 56, pp. 464–468, June 2009.

[123] G. Taylor, S. Moore, S. Wilcox, and P. Robinson, "An on-chip dynamically recalibrated delay line for embedded self-timed systems," in *Proc. IEEE Int. Symp. on Asynchronous Circuits and Systems (ASYNC)*, pp. 45–51, 2000.

[124] A. E. Klein and Y. Tsividis, "Externally linear time invariant digital signal processors," *IEEE Trans. on Signal Processing*, vol. 58, no. 9, pp. 4897–4909, 2010.

[125] D. Pan, "A tutorial on MPEG/Audio compression," in *IEEE Mult. Med.*, pp. 60–74, Summer 1995.

[126] G. Haus and G. Vercellesi, "State of the art and new results in direct manipulation of MPEG audio codes," in *Proc. of SMC 2005*, November 2005.

[127] C. A. Lanciani, *Compressed-Domain Processing of MPEG Audio Signals*. PhD thesis, Georgia Institute of Technology, Electrical Engineering Department, Atlanta, GA, 1999.

[128] C. Lanciani and R. Schafer, "Subband-domain filtering of MPEG audio signals," in *Proc. 1999 IEEE Int. Conf. on Acoustics Speech and Signal Processing (ICASSP)*, pp. 917–920, 1999.

[129] A. Touimi, "A generic framework for filtering in subband-domain," in *Proc. of the Ninth DSP Workshop (DSP2000)*, 2000.

[130] "http://www.ee.columbia.edu/companding/spconf2011.htm."

[131] A. Klein and Y. Tsividis, "Companding digital signal processors," in *Proc. 2006 IEEE ICASSP*, vol. 3, pp. III–700 – III–703, May 2006.

[132] A. Klein and Y. Tsividis, "Instantaneously companding digital signal processors," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 3, pp. III–1433 – III–1436, April 2007.

[133] A. Klein and Y. Tsividis, "Companding digital signal processors," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 3, pp. 700 –703, May 2006.

[134] A. Oppenheim, "Realization of digital filters using block-floating-point arithmetic," *IEEE Trans. on Audio and Electroacoustics*, vol. 18, pp. 130–136, Jun 1970.

[135] S. Sridharan, "Implementation of state-space digital filters using block-floating point arithmetic," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 908–911, 1987.

[136] I. Karatzas and S. E. Shreve, *Brownian Motion and Stochastic Calculus*, vol. 113. Springer Verlag, 1991.

[137] B. Leung, "Noise/jump phenomenon of relaxation oscillators based on phase change using integral/lagranian formulation in quantum mechanics," in *Proc. IEEE Int. Midwest Symp. on Circuits and Systems*, pp. 246–249, 2012.