Timing correction of time-interleaved ADCs

van Otten, R.

Award date:
2009

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

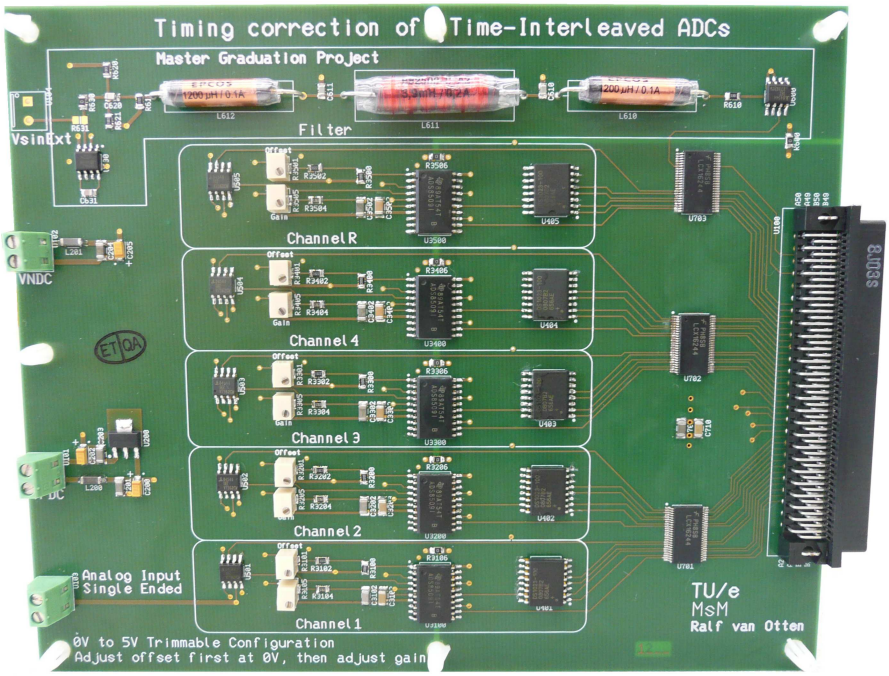
Timing Correction of Time-Interleaved ADCs

Ralf van Otten

June 2009

Master of Science Thesis

Project period: September 2008 – June 2009



Eindhoven University of Technology
Department of Electrical Engineering
Mixed-signal Microelectronics group

Supervisors:
Dr. ir. J.A. Hegt
Prof. Dr. ir. A.H.M. van Roermund

Abstract

Mismatches between the channels of time-interleaved analog to digital converters (TI-ADCs) cause offset, gain and timing errors of which the timing errors are dominant at high frequencies. The thesis explains the mismatches and evaluates existing correction methods. This thesis also proposes a background mode, mixed domain, calibration method for timing correction of TI-ADCs. The method improves the spurious free dynamic range (SFDR) and signal to noise and distortion ratio (SNDR) of the TI-ADC for time-invariant and time-variant timing mismatches. A hardware demonstrator of the proposed method is realized with off-the-shelve components. The results of the demonstrator are compared with those of existing methods.

Contents

List of Abbreviations

ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
DCDL	Digital Controlled Delay Line
DCM	Digital Clock Manager
DLL	Delay-Locked Loop
EMC	Electro-Magnetic Compatibility
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
IC	Integrated Circuit
LPF	Low Pass Filter
MLS	Maximum-Length Sequence
PCB	Printed Circuit Board
PLL	Phase Locked Loop
SFDR	Spurious Free Dynamic Range
SNDR	Signal to Noise and Distortion Ratio
SNR	Signal to Noise Ratio
SPDT	Single Pole Dual Throw
S&H	Sample and Hold
TI-ADC	Time-Interleaved ADC
VCDL	Voltage Controlled Delay Line
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WSS	Wide Sense Stationary

List of Symbols

$\alpha_i(t)$	output signal S&H
$\gamma_i[m]$	discrete-time output signal sub-ADC
$\gamma_i(t)$	continuous-time output signal sub-ADC
$\delta(.)$	dirac function
Δt_i	timing-mismatch
φ_i	sample clock
A	Amplitude
d	default delay
$D_i[k]$	estimated delay
$\hat{e}_i[k]$	estimated timing error
f_0	input frequency
f_s	sample frequency TI-ADC
g_i	gain
M	number of Channels
S	slope
$s_i(t)$	sampling signal channel i
$S_i(j\Omega)$	Fourier transform of $s_i(t)$
T_M	sample period sub-ADC
T_s	sample period TI-ADC
o_i	offset
q	quantization step size
$x(t)$	analog input signal
$X(j\Omega)$	Fourier transform of $x(t)$
$\hat{x}_i(t)$	analog input signal with mismatches in channel i
$\hat{X}_i(j\Omega)$	Fourier transform of $\hat{x}_i(t)$
$y[n]$	discrete-time digital output signal
$y(t)$	continuous-time digital output signal
$Y(j\Omega)$	Fourier transform of $y(t)$
$\mathbb{Z}$	set of integer numbers

Chapter 1

Introduction

This chapter introduces the master graduation project. The chapter gives the motivation, defines the assignment and sketches the overview of the report.

1.1 Motivation

Nowadays all communication is getting faster and faster. Therefore, the demand for high data rate wireless digital communication is high and research into 60GHz communication and above with several GSps data rate is done. The data rate of the wireless digital communication is limited by the sampling rate and the accuracy of the data converters. Other applications are fast and accurate oscilloscopes that are designed to measure signals in the GHz range.

These applications require high bandwidth analog to digital converters (ADCs) with high accuracy and high sampling rate. These requirements are difficult to realize in a single ADC. Therefore, Black and Hodges proposed the Time-Interleaved ADC (TI-ADC) architecture [1] in 1980. Since then lots of research has been done into the possibilities of TI-ADCs. The time-interleaving architecture brings some problems with respect to mismatch between its channels. There are three main mismatch problems: gain-mismatch, offset-mismatch and timing-mismatch. Since TI-ADCs are mainly used in high frequency applications, where the timing-mismatch is dominant, timing-mismatch is the most challenging problem. The architecture and its problems are explained in chapter 2.

1.2 Assignment

The master graduation project is concentrated on the timing correction of TI-ADCs. The first part of the assignment is to summarize and evaluate existing correction methods. Based on the existing methods a new method was developed. The developed method is evaluated with simulations in terms of a Simulink model. Because of promising results a demonstration model is realized in hardware with off-the-shelve components.

Fitsum B. Mesadi has done preliminary research to timing error calibration in TI-ADCs for his internship [2]. His research is used as a starting point for this master graduation project.

1.3 Organization of the Report

Chapter 2 will explain the concept of TI-ADCs and their problems. Chapter 3 describes models of normal TI-ADCs that are used as a reference for the developed method. In chapter 4 the main directions of timing correction methods, currently described in literature, are introduced and evaluated. The developed method is described and simulated in chapter 5. To verify the results of the simulations a demonstrator of the method was realized in hardware. The realization with

off-the-shelf components is described in chapter 6. Chapter 7 shows the measurements results of the demonstrator and compares them with the results of the simulated model. Finally in chapter 8 conclusions are drawn and recommendations are given.

Chapter 2

Time-Interleaved ADCs

This chapter will explain the concept of TI-ADCs and their non-idealities. The first part of the chapter describes the concept of TI-ADCs and in the second part problems that TI-ADCs can have are summarized.

2.1 Concept of Time-Interleaved ADCs

The basic idea of a TI-ADC is to make a fast ADC out of several relatively slow sub-ADCs working together in time-multiplexed mode. The basic concept of a M channel TI-ADC is shown in figure 2.1.

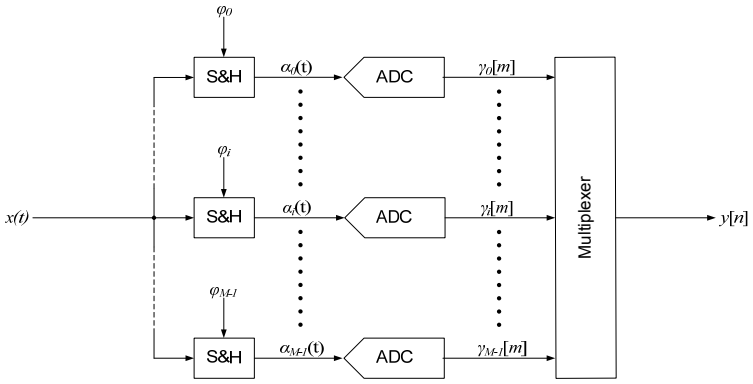


Figure 2.1: Basic Time-Interleaved ADC.

In this figure $x(t)$ is a time-continuous and amplitude-continuous signal which is fed to M sample and holds (S&H). Each S&H takes a sample at time φ_i (i from 0 to $M-1$). φ_i are the sample moments shifted in time as shown in figure 2.2. Ideally the sample moments are equidistant with time T_s.

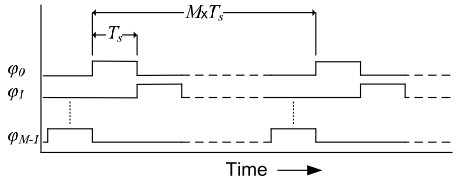


Figure 2.2: Signals φ_i.

The S&H takes a sample of $x(t)$ and holds this value until the next sample. This is represented by

$$\alpha_i(t) = x(t - (t + iT_S) \mathrm{mod} T_M) \tag{2.1}$$

where

$$T_M = MT_S \tag{2.2}$$

This signal is quantized to the time-discrete and amplitude-discrete signal $\gamma_i[m]$ by sub-ADC-i:

$$\gamma_i[m] = \mathrm{round}(\alpha_i(mT_M)) = \mathrm{round}(x(mT_M + iT_S)), \qquad m \in \mathbb{Z} \tag{2.3}$$

where the quantization steps are normalized to 1. The outputs of the sub-ADCs are then multiplexed to

$$y[n] = \gamma_{n \mathrm{mod} M}[n \mathrm{div} M] \tag{2.4}$$

such that

$$y[n] = \mathrm{round}(x(nT_s)) \tag{2.5}$$

With this concept a TI-ADC with an overall sample period T_S is created from M sub-ADCs with per sub-ADC a sample period T_M which is M times slower than the overall sample time. A graphical representation of the signals in a TI-ADC for M=2 is shown in figure 2.3.

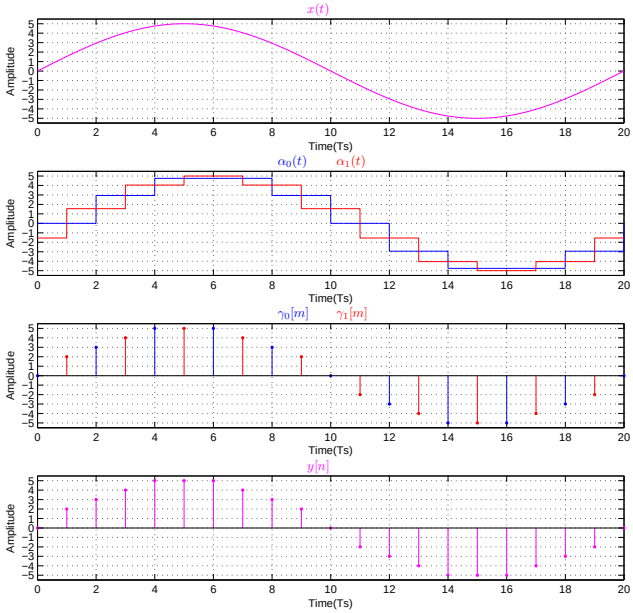


Figure 2.3: Signals in a basic 2-channel TI-ADC.

2.2 Non-Idealities of Time-Interleaved ADCs

The non-idealities of TI-ADCs cause errors in the digital signal and limits the performance of the converter. These limitations can be divided in two groups. The first group of limitations are limitations that are common to all ADCs in general. The second group are the limitations that arise with time-interleaving and are therefore specific to time-interleaving. These groups will be explained in the next sections and in the final section the spectral signatures of the non-idealities are summarized.

2.2.1 Common ADC Limitations

The three most common ADC problems are described in this section, they are sampling jitter, quantization noise and nonlinearity. There has already been a lot of research into these problems and several solutions are available. Therefore, these problems are taken notice of, but will not be considered in the error correction algorithm.

Sampling Jitter

Sampling jitter are random variations of the sample moments of the S&Hs. The main cause of sampling jitter is device noise and random noise from the power supply and substrate. Sampling jitter is also known as phase noise. Because of the random nature of this error it spreads out through the whole spectrum and increases the noise floor, which results in a decrease of the signal to noise ratio (SNR) [3]. Figure 2.4 shows an example of a simulated spectrum of an ideal ADC, without sampling jitter (a) and with sampling jitter (b).

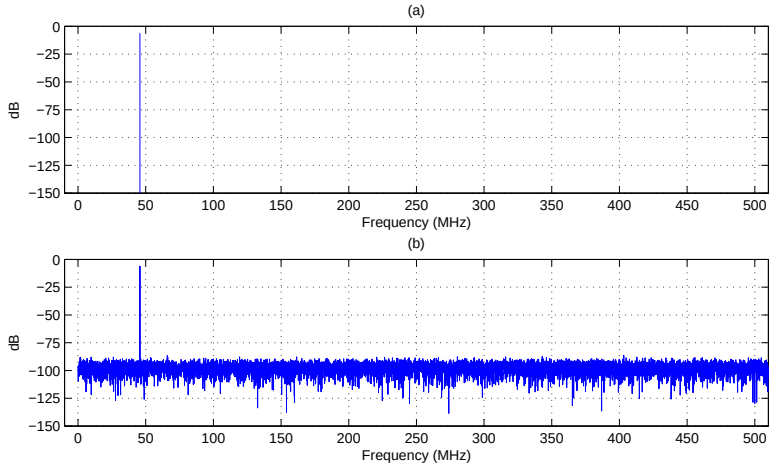


Figure 2.4: Simulated Spectrum of a sine wave with f_0=749(f_s/16384)≈45.7153MHz sampled with an ideal 1GSps ADC, 16384 point FFT,
(a) without sampling jitter,
(b) with sampling jitter, Gaussian distributed with σ=0.01ns.

Quantization Noise

Quantization noise is the error introduced by the amplitude discretization of the signal and is therefore signal dependent 'noise', but can be considered to be stochastic noise under the following conditions: transitions equally distributed in time, many transitions and equal quantization steps. The noise power under these conditions is $q^2/12$, where q is the quantization step size. Since q is inversely proportional to the number of bits this error decreases when more bits are used. This is shown in figure 2.5 for (a) an ideal ADC (no quantization noise), (b) an 8-bit ADC (with quantization noise) and (c) a 16-bit ADC (with quantization noise). For a sine wave with maximal swing SNR=$6.02n + 1.76$, where n is the number of bits.

Nonlinearity

ADCs and S&Hs have certain nonlinearities. Their static nonlinearities are expressed in integral nonlinearity (INL) and differential nonlinearity (DNL). For periodic input signals nonlinearities show up in the frequency spectrum as harmonic distortion tones and determine, a.o., the spurious free dynamic range (SFDR) and the signal to noise and distortion ratio (SNDR).

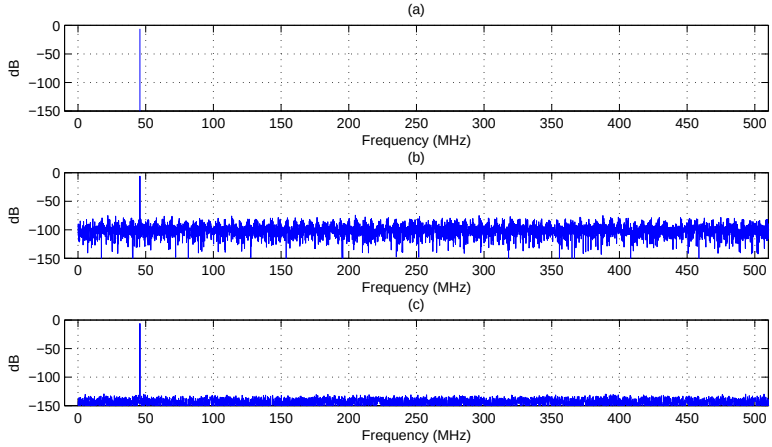


Figure 2.5: Simulated Spectrum of sine wave with f_0=749(f_s/16384)≈45.7153MHz sampled with an 1GSps ADC, 16384 point FFT,
(a) ideal ADC (no quantization noise),
(b) 8-bit ADC (with quantization noise),
(c) 16-bit ADC (with quantization noise).

2.2.2 Specific Time-Interleaving Limitations

The specific time-interleaving limitations discussed in this section appear because of mismatches between channels. They can have a mismatch in offset, gain and timing, of which the timing-mismatch is the largest problem in fast TI-ADCs. All these limitations occur in the frequency domain as spurious tones and therefore degrade the SFDR of the TI-ADC. To understand the occurrence of the spurious tones, the discrete-time domain equations (2.3) and (2.4) are represented in the continuous-time domain by equations (2.6) and (2.7), where the quantization is neglected:

$$\gamma_i(t) = x(t) \cdot \sum_{m=-\infty}^{\infty} \delta(t - mT_M - iT_S) \tag{2.6}$$

$$y(t) = \sum_{i=0}^{M-1} \gamma_i(t) = \sum_{i=0}^{M-1} x(t) \cdot \sum_{m=-\infty}^{\infty} \delta(t - mT_M - iT_S) \tag{2.7}$$

First the combination of the mismatches is shown, subsequently the mismatches are analyzed separately. The mismatch errors are inserted in equation (2.6):

$$\gamma_i(t) = \underbrace{(g_i x(t - \Delta t_i) + o_i)}_{\hat{x}i(t)} \cdot \underbrace{\sum_{m=-\infty}^{\infty} \delta(t - mT_M - iT_S)}_{s_i(t)} \tag{2.8}$$

where for channel i: o_i is the offset, g_i is the gain and Δt_i is the timing error [4]. Then the output of the TI-ADC is:

$$y(t) = \sum_{i=0}^{M-1} \gamma_i(t) = \sum_{i=0}^{M-1} \hat{x}_i(t) s_i(t) \tag{2.9}$$

To obtain the output spectrum, the Fourier transforms of $\hat{x}_i(t)$ and $s_i(t)$ are needed:

$$\hat{X}_i(j\Omega) = g_i X(j\Omega) e^{-j\Omega\Delta t_i} + o_i 2\pi\delta(\Omega) \tag{2.10}$$

where $X(j\Omega)$ is the Fourier transform of $x(t)$

$$\begin{aligned} S_i(j\Omega) =& \frac{2\pi}{MT_s} \sum_{m=-\infty}^{\infty} \delta\left(\Omega - m\frac{\Omega_s}{M}\right) e^{-j\Omega i T_s} \\ =& \frac{2\pi}{MT_s} \sum_{m=-\infty}^{\infty} \delta\left(\Omega - m\frac{\Omega_s}{M}\right) e^{-jmi\frac{2\pi}{M}} \end{aligned} \tag{2.11}$$

The Fourier transform of $y(t)$ then becomes:

$$\begin{aligned} Y(j\Omega) &= \sum_{i=0}^{M-1} \frac{1}{2\pi} \hat{X}_i(j\Omega) * S_i(j\Omega) \\ &= \sum_{i=0}^{M-1} \frac{1}{MT_s} \sum_{m=-\infty}^{\infty} \hat{X}_i\left(\Omega - m\frac{\Omega_s}{M}\right) e^{-jmi\frac{2\pi}{M}} \\ &= \frac{1}{MT_s} \sum_{i=0}^{M-1} \sum_{m=-\infty}^{\infty} \left[g_i X\left(\Omega - m\frac{\Omega_s}{M}\right) e^{-j\left(\Omega - m\frac{\Omega_s}{M}\right)\Delta t_i} + \right. \\ &\quad \left. + o_i 2\pi\delta\left(\Omega - m\frac{\Omega_s}{M}\right)\right] e^{-jmi\frac{2\pi}{M}} \end{aligned} \tag{2.12}$$

For a sine wave at the input, $x(t)$ is

$$x(t) = A\sin(\Omega_0 t) \tag{2.13}$$

and $X(j\Omega)$

$$X(j\Omega) = \frac{A\pi}{j}\left(\delta(\Omega - \Omega_0) - \delta(\Omega + \Omega_0)\right) \tag{2.14}$$

With this, the general formula for the combination of the offset-mismatch, gain-mismatch and timing-mismatch becomes:

$$\begin{aligned} Y(j\Omega) =& \frac{1}{MT_s} \sum_{i=0}^{M-1} \sum_{m=-\infty}^{\infty} \\ & \left[g_i \frac{A\pi}{j}\left(\delta\left(\Omega - m\frac{\Omega_s}{M} - \Omega_0\right) - \delta\left(\Omega - m\frac{\Omega_s}{M} + \Omega_0\right)\right) e^{-j\Omega_0\Delta t_i} + \right. \\ & \left. + o_i 2\pi\delta\left(\Omega - m\frac{\Omega_s}{M}\right)\right] e^{-jmi\frac{2\pi}{M}} \end{aligned} \tag{2.15}$$

The delta functions represent tones at $\Omega{=}m\frac{\Omega_s}{M}\pm\Omega_0$ and $\Omega{=}m\frac{\Omega_s}{M}$.
Ideally $o_i{=}0, g_i{=}1$ and $\Delta t_i{=}0$ and the ideal formula for $Y(j\Omega)$ becomes:

$$Y(j\Omega) = \frac{1}{MT_s} \sum_{i=0}^{M-1} \sum_{m=-\infty}^{\infty} \left[\frac{A\pi}{j}\left(\delta\left(\Omega - m\frac{\Omega_s}{M} - \Omega_0\right) - \delta\left(\Omega - m\frac{\Omega_s}{M} + \Omega_0\right)\right)\right] e^{-jmi\frac{2\pi}{M}} \tag{2.16}$$

Simplifying equation (2.16), only the tones at $k\Omega_s \pm \Omega_0$ are left because all other tones cancel out:

$$\begin{aligned} & Y(j\Omega) = \tfrac{A\pi}{MT_s j} \sum_{m=-\infty}^{\infty} \left[\left(\delta\left(\Omega - m\tfrac{\Omega_s}{M} - \Omega_0\right) - \delta\left(\Omega - m\tfrac{\Omega_s}{M} + \Omega_0\right)\right)\right] \sum_{i=0}^{M-1} e^{-jmi\frac{2\pi}{M}} \\ & \sum_{i=0}^{M-1} e^{-jmi\frac{2\pi}{M}} = \begin{cases} M, & m\mathrm{mod}M = 0 \\ 0, & m\mathrm{mod}M \neq 0 \end{cases} \\ & Y(j\Omega) = \tfrac{A\pi}{T_s j} \sum_{k=-\infty}^{\infty} \left[\left(\delta\left(\Omega - k\Omega_s - \Omega_0\right) - \delta\left(\Omega - k\Omega_s + \Omega_0\right)\right)\right] \end{aligned} \tag{2.17}$$

where $k = m/M$ for $m \text{mod} M = 0$. For the fundamental interval equation (2.17) becomes:

$$Y(j\Omega) = \frac{A\pi}{T_s j}\left[\left(\delta\left(\Omega - \Omega_0\right) - \delta\left(\Omega + \Omega_0\right)\right)\right] \tag{2.18}$$

The canceling of the non-ideal tones also happens if the offset, gain and timing in all channels are equal ($o_i = o$, $g_i = g$ and $\Delta t_i = \Delta t$):

$$\begin{aligned} Y(j\Omega) =& \frac{gA\pi}{jMT_s} e^{-j\Omega_0 \Delta t} \cdot \\ & \cdot \sum_{m=-\infty}^{\infty} \left[\left(\delta\left(\Omega - m\frac{\Omega_s}{M} - \Omega_0\right) - \delta\left(\Omega - m\frac{\Omega_s}{M} + \Omega_0\right)\right) + o2\pi\delta\left(\Omega - m\frac{\Omega_s}{M}\right)\right] \cdot \\ & \cdot \sum_{i=0}^{M-1} e^{-jmi\frac{2\pi}{M}} \\ Y(j\Omega) =& \frac{gA\pi}{jT_s} e^{-j\Omega_0 \Delta t} \sum_{k=-\infty}^{\infty} \left[\left(\delta\left(\Omega - k\Omega_s - \Omega_0\right) - \delta\left(\Omega - k\Omega_s + \Omega_0\right)\right) + o2\pi\delta\left(\Omega - k\Omega_s\right)\right] \end{aligned} \tag{2.19}$$

For the fundamental interval equation (2.19) becomes:

$$Y(j\Omega) = \frac{gA\pi}{jT_s} e^{-j\Omega_0 \Delta t}\left[\left(\delta\left(\Omega - \Omega_0\right) - \delta\left(\Omega + \Omega_0\right)\right) + o2\pi\delta\left(\Omega\right)\right] \tag{2.20}$$

With an offset unequal to zero a DC tone for the offset of the channels is introduced.
Figure 2.6 shows the single sided spectrum of an ideal TI-ADC for two input frequencies.

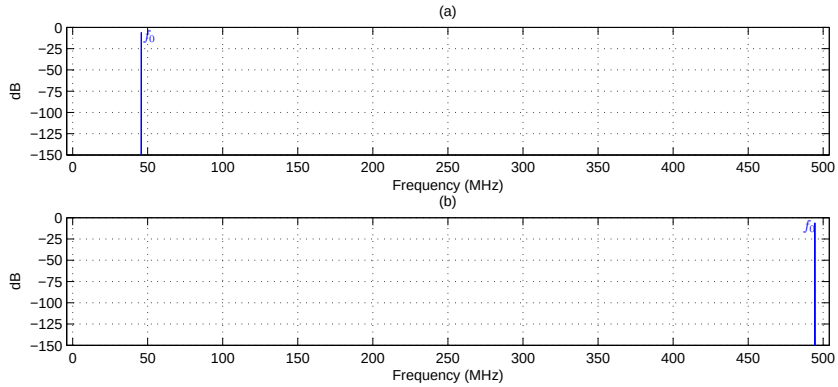


Figure 2.6: Simulated spectrum of an ideal 4-channel TI-ADC with f_s=1GHz,
(a) single sided spectrum of $y[n]$ with $f_0 \approx$45.7153MHz 16384 point FFT,
(b) single sided spectrum of $y[n]$ with $f_0 \approx$494.4458MHz 16384 point FFT.

In the next sections the different mismatch errors are discussed separately.

Offset Mismatch

For the discussion about offset-mismatch the offset errors are assumed to be different for each channel, and all other characteristics are the same. Offset is a DC error per sub-ADC which becomes periodic with time-interleaving. Therefore, the offset-mismatch is periodic with period MT_s and independent of the input signal. In the frequency-domain the offset-mismatch appears as tones at frequencies independent of the input frequency and independent of the input amplitude.

This can be shown by eliminating the gain and timing mismatches in equation (2.9) and (2.15):

$$y(t) = \sum_{i=0}^{M-1} \sum_{m=-\infty}^{\infty} (x(t)\underline{+o_i})\delta(t - mT_M - iT_S) \tag{2.21}$$

$$Y(j\Omega) = \frac{1}{MT_s}\sum_{i=0}^{M-1}\sum_{m=-\infty}^{\infty} \left[\frac{A\pi}{j}\left(\delta\left(\Omega - m\frac{\Omega_s}{M} - \Omega_0\right) - \delta\left(\Omega - m\frac{\Omega_s}{M} + \Omega_0\right)\right) + \underline{+o_i 2\pi\delta\left(\Omega - m\frac{\Omega_s}{M}\right)}\right] e^{-jmi\frac{2\pi}{M}} \tag{2.22}$$

Therefore, the tones are at:

$$\Omega_{error} = \tfrac{m}{M}\Omega_s, \qquad m \in \mathbb{Z} \tag{2.23}$$

The power of the offset error is constant and independent of the input signal as well. Figure 2.7 shows the simulation of the TI-ADC as in figure 2.6 but with the offset-mismatch (o=[−0.002 0.0033 −0.0021 −0.004]). [5]

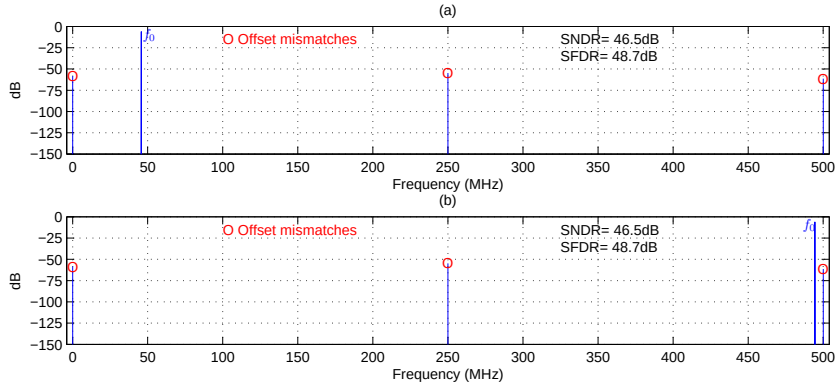


Figure 2.7: Simulated spectrum of a 4-channel TI-ADC with f_s=1GHz and offset-mismatch (o=[−0.002 0.0033 −0.0021 −0.004]),
(a) single sided spectrum of $y[n]$ with f_0≈45.7153MHz 16384 point FFT,
(b) single sided spectrum of $y[n]$ with f_0≈494.4458MHz 16384 point FFT.

The simulation shows that the spurious tones are at a fixed frequency and the SFDR and SNDR are also independent of the input frequency.

Gain Mismatch

For the discussion about gain-mismatch the gain errors are assumed to be different for each channel, and all other characteristics are the same. The errors also occur with a period of MT_s, just as offset mismatch, but the errors are amplitude modulated with the input frequency Ω_0. The largest absolute errors occur at the peaks off the input signal. Therefore, in frequency domain, the location of the error is dependent on the input frequency while the power of the error is independent of Ω_0 but dependent on the amplitude of the input signal. This can be shown by eliminating the offset and timing mismatches in equation (2.9) and (2.15):

$$y(t) = \sum_{i=0}^{M-1}\sum_{m=-\infty}^{\infty} \underline{g_i} x(t)\delta(t - mT_M - iT_S) \tag{2.24}$$

$$Y(j\Omega) = \frac{1}{MT_s}\sum_{i=0}^{M-1}\sum_{m=-\infty}^{\infty} \left[\underline{g_i}\frac{A\pi}{j}\left(\delta\left(\Omega - m\frac{\Omega_s}{M} - \Omega_0\right) - \delta\left(\Omega - m\frac{\Omega_s}{M} + \Omega_0\right)\right)\right] e^{-jmi\frac{2\pi}{M}} \tag{2.25}$$

Therefore, the tones are at:

$$\Omega_{error} = \frac{m}{M}\Omega_s \pm \Omega_0, \qquad m \in \mathbb{Z}, m\text{mod}M \neq 0 \tag{2.26}$$

Figure 2.8 shows the simulation of the TI-ADC as in figure 2.6 but with the gain-mismatch (g=[0.994 0.9891 1.009 0.996]). The simulation shows that the location of the spurious tones are

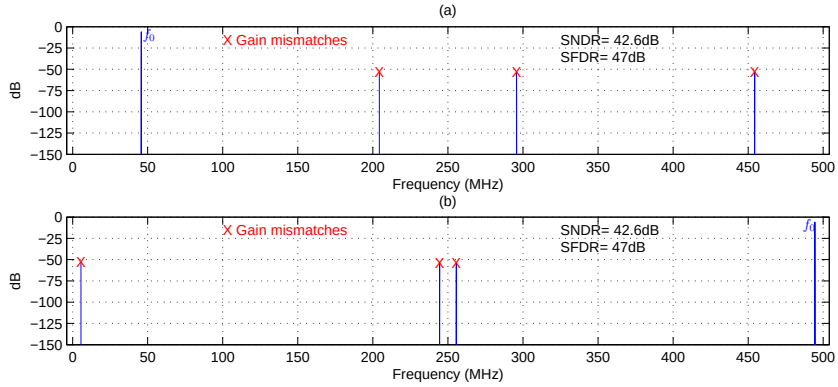


Figure 2.8: Simulated spectrum of a 4-channel TI-ADC with f_s=1GHz and gain-mismatch (g = [0.994 0.9891 1.009 0.996]),
(a) single sided spectrum of $y[n]$ with f_0≈45.7153MHz 16384 point FFT,
(b) single sided spectrum of $y[n]$ with f_0≈494.4458MHz 16384 point FFT.

at a frequency dependent on the input frequency but the SFDR and SNDR are independent of the input frequency.

Timing Mismatch

For the discussion about timing-mismatch, the timing error, due to clock-skew, is assumed to be different for each channel, and all other characteristics are the same. The errors again occur with a period of MT_s and are amplitude modulated with the input frequency Ω_0 just as the gain-mismatch. The largest errors occur at the largest slew-rate of the sine wave. Therefore, the location of the error in the frequency domain is again dependent on the input frequency, and the power of the error is proportional to Ω_0 and dependent to the amplitude of the input signal. This can be shown by eliminating the offset and gain mismatches in equation (2.9) and (2.15):

$$y(t) = \sum_{i=0}^{M-1} \sum_{m=-\infty}^{\infty} x(t\underline{-\Delta t_i})\delta(t - mT_M - iT_S) \tag{2.27}$$

$$\begin{aligned} Y(j\Omega) = &\frac{1}{MT_s} \sum_{i=0}^{M-1} \sum_{m=-\infty}^{\infty} \\ &\left[\frac{A\pi}{j}\left(\delta\left(\Omega - m\frac{\Omega_s}{M} - \Omega_0\right) - \delta\left(\Omega - m\frac{\Omega_s}{M} + \Omega_0\right)\right)\underline{e^{-j\Omega_0\Delta t_i}}\right] e^{-jmi\frac{2\pi}{M}} \end{aligned} \tag{2.28}$$

Therefore, the tones are at:

$$\Omega_{error} = \frac{m}{M}\Omega_s \pm \Omega_0, \qquad m \in \mathbb{Z}, m\text{mod}M \neq 0 \tag{2.29}$$

Figure 2.9 shows the simulation of the TI-ADC as in figure 2.6 but with the timing-mismatch (Δt=[−0.009 −0.002 −0.008 0.004]ns).

The simulation shows that the locations of the spurious tones are dependent on the input frequency. It also shows that the SFDR and SNDR are also dependent on the input frequency.

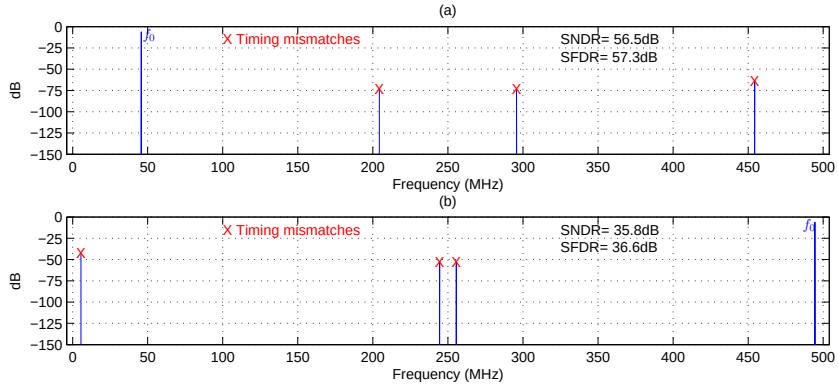


Figure 2.9: Simulated spectrum of a 4-channel TI-ADC with f_s=1GHz and timing-mismatch (Δt=[−0.009 −0.002 −0.008 0.004]ns),
(a) single sided spectrum of $y[n]$ with f_0≈45.7153MHz 16384 point FFT,
(b) single sided spectrum of $y[n]$ with f_0≈494.4458MHz 16384 point FFT.

Because the error is proportional to the input frequency, this error is dominating at higher speeds and therefore important and challenging to correct with high accuracy. The timing-mismatch spurs are located at the same frequencies as the gain-mismatch spurs, but can be distinguished because the power of the gain-mismatch is independent of the input frequency (dominant at low frequencies) and timing-mismatch is dependent on the input frequency (dominant at high frequencies).

Total Mismatch

Figure 2.10 shows the same graphs as in figure 2.6 but with all mismatch errors included. Offset (o=[−0.002 0.0033 −0.0021 −0.004]), gain (g=[0.994 0.9891 1.009 0.996]), timing (Δt=[−0.009 −0.002 −0.008 0.004]ns).

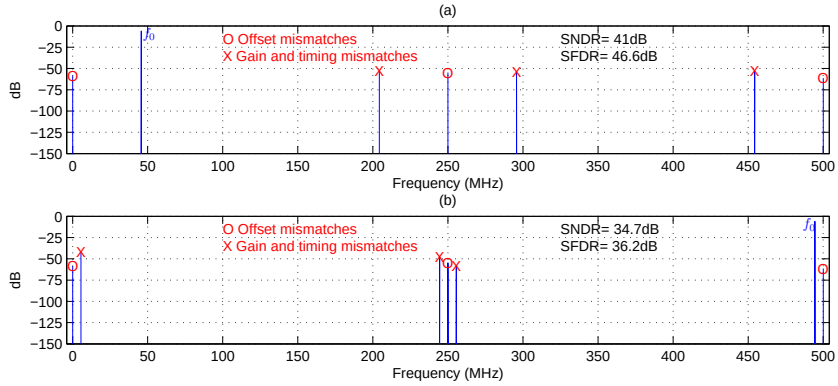


Figure 2.10: Simulated spectrum of a 4-channel TI-ADC with f_s=1GHz and all three mismatches, offset (o=[−0.002 0.0033 −0.0021 −0.004]), gain (g=[0.994 0.9891 1.009 0.996]), timing (Δt=[−0.009 −0.002 −0.008 0.004]ns),
(a) single sided spectrum of $y[n]$ with f_0≈45.7153MHz 16384 point FFT,
(b) single sided spectrum of $y[n]$ with f_0≈494.4458MHz 16384 point FFT.

The simulation shows that the timing-mismatch becomes dominant over the gain-mismatch for higher frequencies, as expected.

2.2.3 Spectral Signatures

Each non-ideality affects in a specific way the spectrum of the output signal. An overview of the spectral signatures of the non-idealities of TI-ADCs is given below.

Common ADC Limitations:

- Sampling Jitter
  - Increases the noise floor
  - Signal amplitude and frequency dependent
- Quantization Noise
  - Increases the noise floor
  - Dependent on the number of bits
- Nonlinearity
  - Generates spurs for periodic signals at multiples of input frequency and at all combinations of input frequency and clock frequency (due to sampling)
  - Power of spurs dependent on signal amplitude

Specific TI-ADC Limitations:

- Offset-Mismatch
  - Generates spurs
  - Location dependent on number of channels
  - Independent of signal amplitude
  - Independent of input frequency
- Gain-Mismatch
  - Generates spurs
  - Location dependent on number of channels and input frequency
  - Power of spurs dependent on signal amplitude
  - Power of spurs independent of input frequency
- Timing-Mismatch
  - Generates spurs
  - Location dependent on number of channels and input frequency
  - Power of spurs dependent on signal amplitude
  - Power of spurs dependent on input frequency

Chapter 3

MATLAB/Simulink Model of TI-ADC

This chapter describes models that are used as a reference for the developed method in chapter 5. In the first section an ideal TI-ADC with only quantization is modeled in MATLAB m-code and in Simulink. The second section describes the MATLAB m-code and Simulink model of a TI-ADC with quantization and mismatches included.

3.1 Ideal TI-ADC

An ideal 4-channel TI-ADC with quantization is modeled in Simulink(figure 3.1) and MATLAB m-code(appendix A.1).

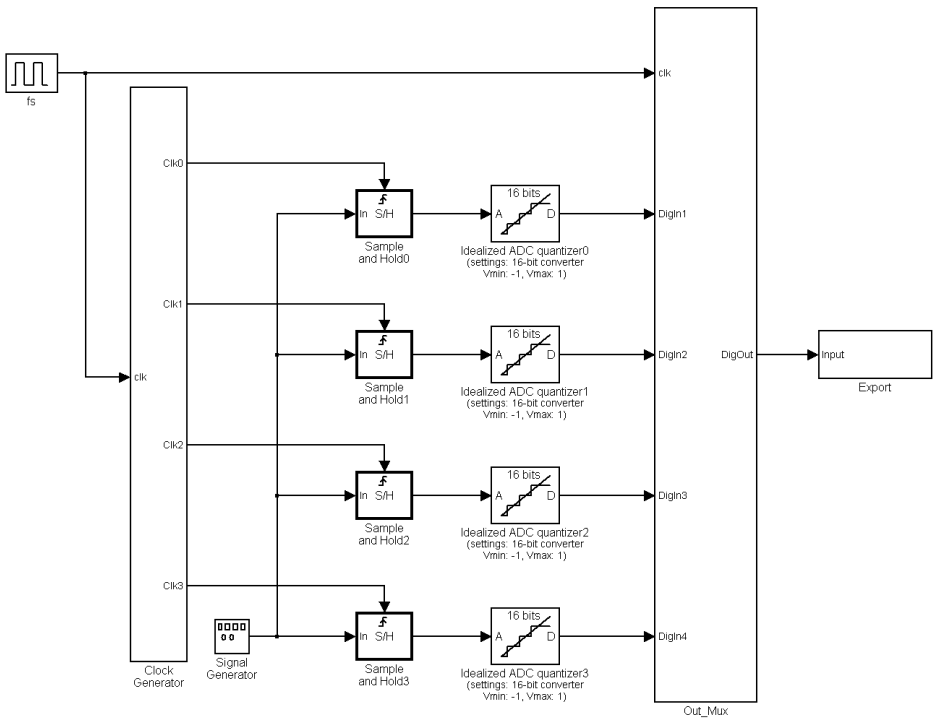


Figure 3.1: Simulink model of ideal TI-ADC.

The Simulink model is comparable with the basic TI-ADC structure in figure 2.1. The model

uses a signal generator for the generation of the analog input signal. The clock generator generates four phase shifted clocks at $f_s/4$. The S&Hs are modeled by ideal S&H blocks and the sub-ADCs are represented by ideal ADC quantizer blocks. The output multiplexer is modeled by a 4-to-1 multiplexer controlled by a counter, running at f_s, counting from 0 to 3. The output is exported to a mat-file, the data is used for plotting the spectrum of the output signal.

The MATLAB m-code model calculates the output for a given number of samples, the output of the sub-ADCs is calculated directly from the sine function at the input. The signal is then multiplexed and the output signal and its single sided spectrum are plotted.

3.1.1 Simulation Results

Simulation is done for a 1GSPS TI-ADC, with f_0=1013f_s/2048≈494.6MHz. The single sided spectrum of a 2048 point FFT is shown in figure 3.2. The SNDR is 98.2dB and the SFDR is

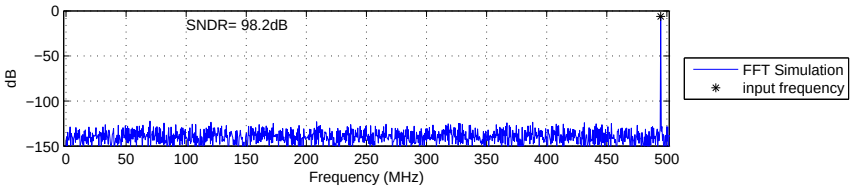


Figure 3.2: Spectrum of 4-channel TI-ADC, no correction algorithm, no mismatch errors, f_0≈494.6MHz, 2048 point FFT.

covered by the quantization noise. The ideal SNDR for a 16 bit converter with max swing input is 98.08dB. Therefore, the SNDR in simulation is quite accurate.

3.2 TI-ADC with Mismatch Errors

In the Simulink model of figure 3.3 and the MATLAB m-code model of appendix A.2 the mismatch errors are added.

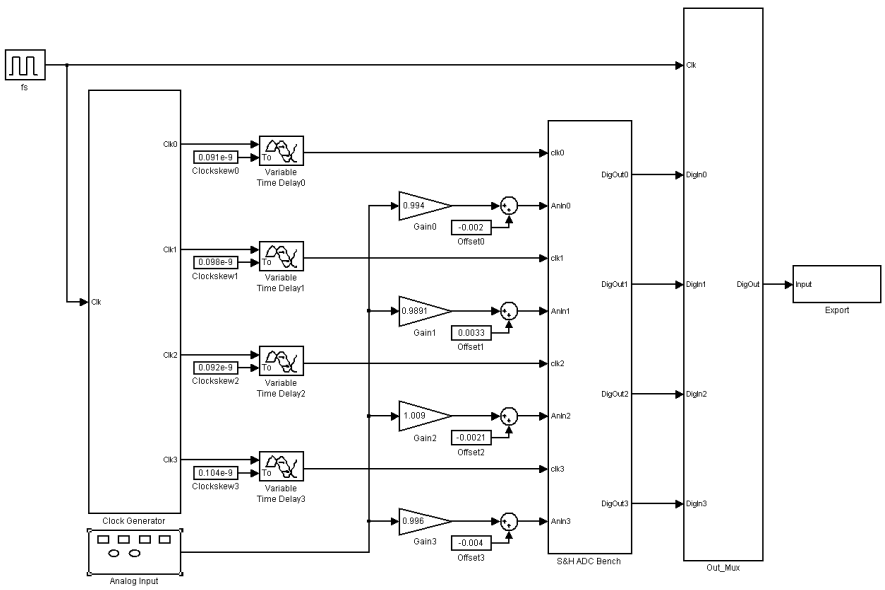


Figure 3.3: Simulink model of ideal TI-ADC with mismatch.

In the Simulink model the S&Hs and the sub-ADCs are placed in the subsystem S&H ADC Bench. The gain errors are modeled by a gain block in each channel. The offsets are modeled by a constant added to the analog input for each channel. The clock skew errors are represented by a constant controlling a variable time delay in each clock line.

In the MATLAB m-code model the mismatches are integrated in the functions for each channel, all values for each channel are calculated at once. This uses less simulation time then the Simulink model, but this model is not useful for modeling a correction algorithm which corrects the sample clock. The correction algorithm uses a feedback loop adapting the timing of the samples clocks dependent on the signal.

3.2.1 Simulation Results

Simulation is done for a 1GSPS TI-ADC, with f_0=1013f_s/2048≈494.6MHz. Here with all three mismatches, offset, gain and timing. The single sided spectrum of a 2048 point FFT is shown in figure 3.4. The location and power of spurs due to the mismatches correspond with the theory

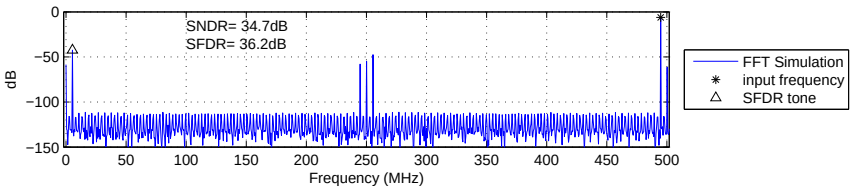


Figure 3.4: Spectrum of 4-channel TI-ADC, no correction algorithm, with all mismatches, offset (o=[−0.002 0.0033 −0.0021 −0.004]), gain (g=[0.994 0.9891 1.009 0.996]), timing (Δt=[−0.009 −0.002 −0.008 0.004]ns), f_0≈494.6MHz, 2048 point FFT.

described is chapter 2 which result in a SNDR of 34.7dB and a SFDR of 36.2dB.

Since the developed algorithm from chapter 5 assumes only a timing-mismatch and an offset of zero and a gain of one for each channel, the same simulation is repeated with only timing-mismatch, the results are shown in figure 3.5. The figure shows spurs at the expected frequencies

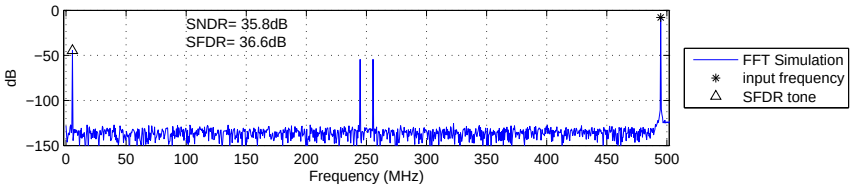


Figure 3.5: Spectrum of 4-channel TI-ADC, no correction algorithm, with timing-mismatch(Δt=[−0.009 −0.002 −0.008 0.004]ns), f_0≈494.6MHz, 2048 point FFT.

with expected power, this results in a SNDR of 35.8dB and a SFDR of 36.6dB.

Chapter 4

Existing Timing Correction Methods

In this chapter timing correction methods currently described in literature are reviewed. There are three main directions of correction: Two-ranks Sample & Hold, Channel Randomization and Calibration. These are introduced in the first sections of this chapter. After that some comparison criteria are discussed. Finally the present methods are evaluated in the last section of the chapter.

4.1 Two-Ranks Sample & Hold

The two-ranks S&H is one of the first methods proposed to avoid clock-skew in TI-ADCs [6]. Figure 4.1 shows the basic idea of this method.

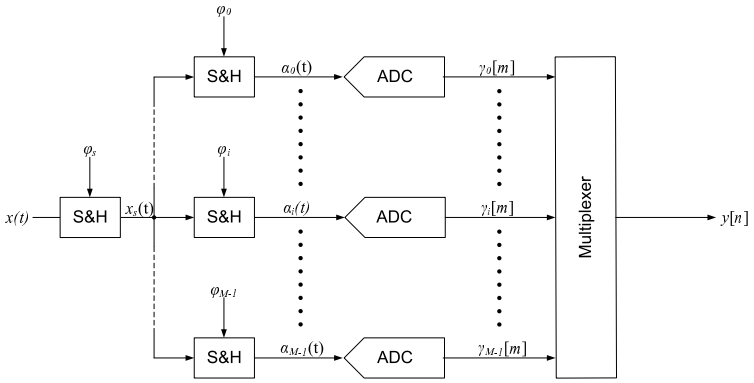


Figure 4.1: Basic Two-ranks Sample & Hold TI-ADC.

The S&H in front of the other S&Hs samples input $x(t)$ with a sample frequency Ω_s. The output $x_s(t)$ is then sampled again by the next S&H at the lower frequency Ω_s/M. Because $x_s(t)$ is nearly constant around the re-sampling time, the clock skew in ϕ_i can be neglected.

The advantage of this architecture is that only one sampling clock determines all sampling instants, fully avoiding any clock skew during the input signal sampling.

The disadvantage of this architecture is that the front S&H is still sampling at high speed. This means that the front S&H limits the overall bandwidth and linearity.

4.2 Channel Randomization

Randomizing the order of the channels is another way of decreasing the impact of the mismatch errors at the cost of extra ADCs. This method spreads the distortion over a wider frequency range, improving the SFDR. Figure 4.2 shows a diagram of a Random TI-ADC with additional sub-ADCs to further increase the performance.

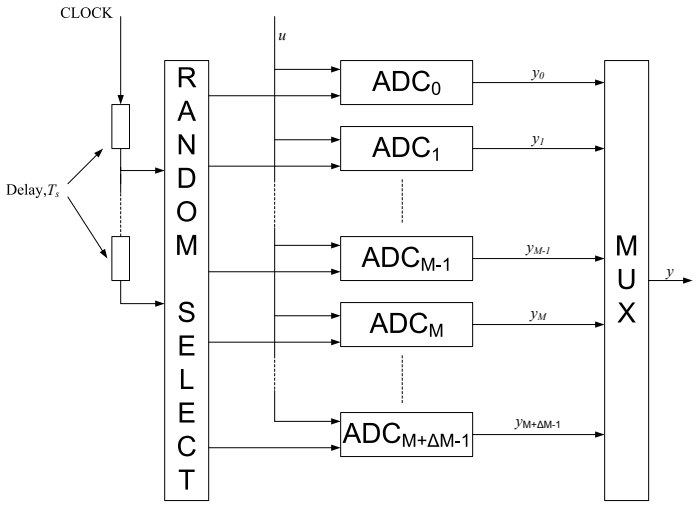


Figure 4.2: Channel Randomization of a M-channel TI-ADC with ΔM additional sub-ADCs, each sampling instance $\Delta M + 1$ sub-ADCs are available.

This technique is not removing the errors, it decorrelates the errors of the output samples. Therefore, the power of the noise and distortion are still in the output samples and the SNDR is not reduced [7]. The amount of improvement of the SFDR is dependent on the number of extra sub-ADCs.

4.3 Calibration

Calibration is the most promising method, and widely discussed in current literature. For this technique, detection and correction of the error is required. The continuity of the conversion process during calibration defines the mode of calibration, this is discussed in the next part of this section. After that, calibration domains to detect and correct the error are discussed.

4.3.1 Calibration Mode

Calibration can either be done by interrupting the conversion process, which is called foreground mode, and otherwise during conversions, known as background mode.

Foreground Mode

Calibrating the TI-ADC at start-up is relatively easy by means of using a known input signal to detect and correct the error. In this way static errors are corrected and after calibration the analog input signal has no constraints. On the other hand, time varying errors can't be corrected without interrupting the conversion to recalibrate. Interrupting the conversion is usually not an option in case of continuous data streams, because data is lost. An advantage of foreground mode is that the calibration circuit can be turned off during conversion; this saves power and therefore might be an option for battery powered systems.

Background Mode

Calibrating the TI-ADC in background has the advantage that no data is lost and both static and time varying errors can be corrected. Nevertheless the first data are uncorrected. Several methods are proposed using background mode, each with its own advantages and disadvantages. Some disadvantages are: constraints on the analog input signal, only applicable to a fixed number of channels or oversampling of the input signal.

4.3.2 Calibration Domain

Since ADCs work in the mixed analog and digital domain, calibration domains are classified according to the domain where the detection and correction are done. Therefore, there are three categories: all analog, all digital and mixed calibration.

All Analog Calibration

In all analog calibration architectures, detection and correction are fully done in the analog domain. This usually means that the sample clock is measured and corrected. This is potentially simpler than in the digital domain. The clock-skew is however detected before the S&Hs, so delay mismatches in the S&Hs are not detected and therefore the correction is less accurate. Another disadvantage of analog calibration systems is that they are sensible to fabrication inaccuracies ("process spread").

All Digital Calibration

The all digital calibration architectures detect and correct fully in the digital domain. The digital blocks have the advantage that they are insensible to process spread. The calibration is mostly done by estimating the error from the digital output, and correcting the digital output by interpolation. This is a complex method compared to the analog calibration and the correction has to be done for every sample. Therefore, the correction has to be done at full clock speed, which is bad for the power dissipation.

Mixed Calibration

Mixed calibration architectures try to combine the best of both domains, normally with detection in the digital domain, by estimating or measuring the error from the digital output and correcting in the analog domain, by adjusting the sample clock. In this way it is relative simple, accurate and robust to process spread.

4.4 Comparison Criteria

Comparison criteria are defined below to appropriately compare the methods in literature.

Complexity

Complexity in terms of extra hardware required for timing correction.

Oversampling Ratio

Ratio of Nyquist frequency (half the sampling frequency) to input signal bandwidth. Without oversampling (oversampling ratio of one), the input signal is sampled at Nyquist rate.

Test Signal

The requirement of an internally or externally generated test signal for the timing correction.

Input Restrictions

Restrictions to the input signal in order to correct the timing error properly.

4.5 Evaluation of Existing Methods

This section describes and evaluates some methods presented in literature, the methods are arranged by calibration domain.

4.5.1 All Analog Calibration Example

All analog calibration is the domain of which the least publications are available. Therefore, only one is description is given.

Wu and Black's Calibration [8]

Wu and Black proposed "A low-jitter skew-calibrated multiphase clock generator for time-interleaved applications." Their method can be used for a conventional ring oscillator based phase locked loop (PLL) or delay line based delay-locked loop (DLL) multi phase clock generator, with one output phase that is aligned to an external reference clock through feedback. The conventional feedback controls all delay elements at once, which generates clock skew because of long delay paths from generator to samplers and due to mismatch of the delay stages. In this way only one clock phase is controlled. The proposal from Wu and Black is to add independently controllable delay cells to each phase without interfering with the main loop. Figure 4.3 shows the system architecture of their proposal for an 8-channel multiphase clock generator.

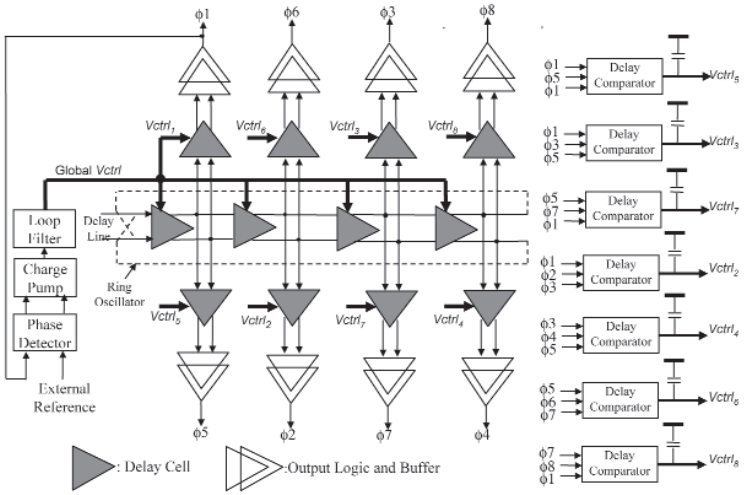


Figure 4.3: System architecture of Wu and Black's calibration method for an 8-channel multiphase clock generator.

The control voltages (V_{ctlj}) for the delay cells are calculated by a delay comparator. The delay comparator determines V_{ctlj} from 3 clocks (ϕ_i, ϕ_j and ϕ_k) by comparing $\Delta td_{i,j}$ and $\Delta td_{j,k}$. $\Delta td_{i,j}$ is the time difference between the rising edge of ϕ_i and ϕ_j. The calibration for the 8-channel multiphase clock generator is illustrated in figure 4.4.

For the 8-channel multiphase clock generator, ϕ_1 is controlled by the external reference. ϕ_5 is the inverted version of ϕ_1 and V_{ctl5} is therefore calculated by the comparison of $\Delta td_{1,5}$ and $\Delta td_{5,1}$. With ϕ_1 and ϕ_5 controlled, ϕ_3 is calibrated by comparing $\Delta td_{1,3}$ and $\Delta td_{3,5}$, and ϕ_7 is controlled

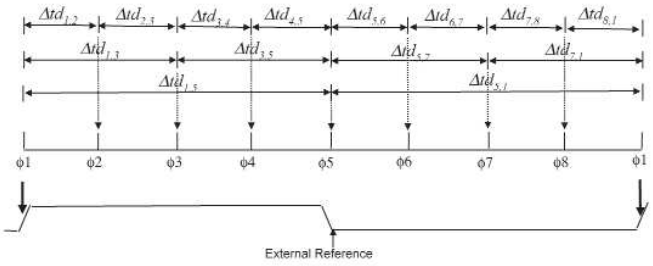


Figure 4.4: Calibration illustration of Wu and Black's calibration method for an 8-channel multiphase clock generator.

by comparison of $\Delta td_{5,7}$ and $\Delta td_{7,1}$. This process is continued for the other phases and is carried out simultaneously and continuously. Therefore, the calibration is in background mode.

The drawback of this method is that it assumes that the timing-mismatch is only caused by clock skew in the clock generator and differences in timing in S&Hs are not corrected.

4.5.2 All Digital Calibration Example

There are quite some publications about all digital calibrations. Most all digital calibration methods are based on the same principle but have a different algorithm of estimating the error. One example is given here.

Iroaga *et al.* Calibration [9]

Iroaga *et al.* present a background mode calibration architecture for correcting the timing-mismatch. They require an extra calibration sub-ADC and a digital interpolation filter. The use of the extra sub-ADC removes any statistical restriction on the converted input signal and spectrum, except the Nyquist criterium ($f_0<=f_s/2$), but requires a periodic calibration signal (periodic ramp) to extract timing information used for correction of the output. A block diagram of the proposal is shown in figure 4.5.

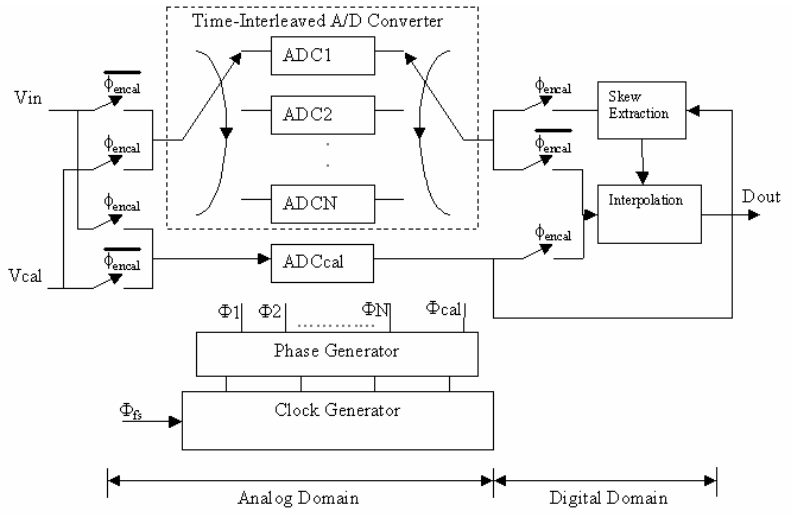


Figure 4.5: N-channel self-calibrating TI-ADC from Iroaga *et al.*

The sub-converters ADC1..ADCN are running at f_s/N and ADCcal is running at $f_s/(N+1)$,

ADCcal is then sampling at the same time as one of the sub-converters but at $N/(N+1)$ the rate. Therefore, ADCcal cycles through the phases of the other sub-converters (figure4.6) and is able to replace the sub-ADC under test. ADCcal samples the the normal input signal and the sub-ADC

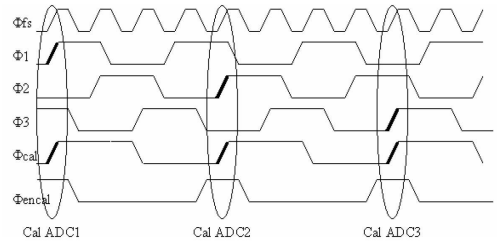


Figure 4.6: Timing relationship between sub-converter clocks for N=3

under test samples the periodic calibration signal. The sample from the sub-ADC under test is then used to extract the timing error in this channel. The timing error is estimated by subtracting the sample from a reference sample and dividing the difference by the slope of the periodic calibration signal. The reference sample is a sample from the periodic calibration signal sampled by ADCcal at the start of the calibration. The timing error is used in a digital interpolation filter to correct the digital output. The interpolation filter keeps track of the last K output codes and their timing information, with K odd so for every interpolation the middle sample is corrected using $K-1$ surrounding samples. The interpolation is based on Neville's algorithm, the algorithm requires $K(K-1)$ multiplications and $K(K-1)/2$ subtractions per output sample.

Simulation results of the presented method show improvement of SFDR and SNDR for frequencies up to Nyquist but a significant improvement of SFDR and SNDR is achieved for an oversampling ratio of at least 2 times, for a 10-bit converter.

This method extracts a sub-ADC one-by-one for calibration, making it a foreground method but by using an extra sub-ADC to replace the extracted sub-ADC, it is able to run in background without interrupting the conversion. The timing error is referenced to ADCcal but this reference value is only calculated once. Therefore, drift in timing of ADCcal is not corrected.

4.5.3 Mixed Calibration Examples

In mixed calibration methods there is a larger variety, four rather different methods are described here.

Seo *et al.* Calibration [10]

Seo *et al.* proposed "a low computation adaptive blind mismatch correction for TI-ADCs". Their method uses the autocorrelation properties of the input signal to estimate the gain-mismatch and timing-mismatch. Figure 4.7 shows a block diagram of their proposal for a 4-channel TI-ADC.

For this method the input signal is assumed to be zero mean and wide sense stationary(WSS). Under the restrictions assumed for the input, the autocorrelation is shift independent and the unit-lag autocorrelation depends only on timing-mismatches. Therefore, if timing-mismatches are present, the output of the TI-ADC is no longer WSS and the autocorrelation has become shift-dependent. The zero-lag output autocorrelation should be equal for all channels because of the zero mean property, this is used for estimation of the gain-mismatch. Estimation for gain-mismatch and timing-mismatch is done by comparing the autocorrelation of one channel with the average autocorrelation of all channels. The gain-mismatch is corrected by adjusting the output of the sub-ADCs and the timing-mismatch is corrected by adapting the sampling clock.

The methods does not require an external signal to be able to calibrate correctly, but this means that there are requirements on the converted input signal in order to function correctly.

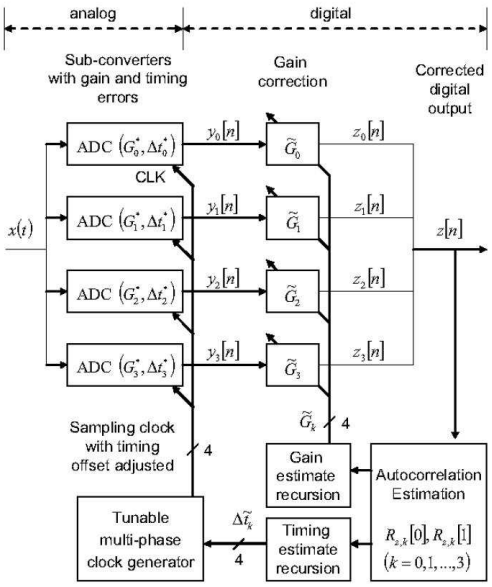


Figure 4.7: 4-channel TI-ADC system proposed by Seo *et al.*

Liu *et al.* Calibration [11]

Liu *et al.* presented a "Simultaneous Compensation of RC Mismatch and Clock Skew in Time-Interleaved S/H Circuits". They use a negative feedback control loop constituted with a voltage controlled delay line (VCDL), a digital comparator, a digital integrator, and a digital to analog converter (DAC) for each channel. The diagram of this compensation method is shown in figure 4.8.

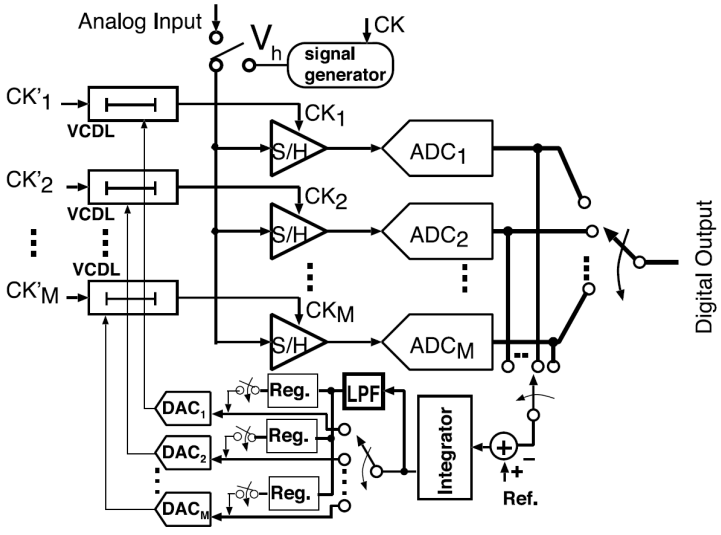


Figure 4.8: Diagram of Liu *et al.* compensation method for time-interleaved S&H circuits.

The sampling clocks are delayed by the VCDL, the delay is controlled by the DACs. The error is calculated by a pulse input signal of the sampling frequency with a linear slope (S_0) at the falling edge. The signal and sampling is shown in figure 4.9.

Without Mismatch, the outputs of the sub-ADCs are the same, when mismatches occur the

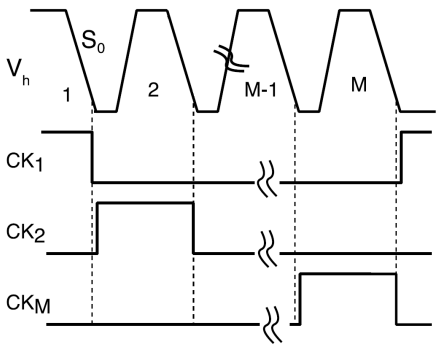


Figure 4.9: Sampling during calibration.

feedback loop controls the VCDL to be such that the outputs of the sub-ADCs equal a reference voltage. A digital integrator is used to reduce the influence of noise. Another low-pass filter (LPF) can be used to further reduce the influence of noise. The compensation codes calculated during calibration are stored in registers after calibration is finished.

This method is an example of calibration in foreground. A disadvantage is that the calibration is done with a signal that has frequencies (twice the Nyquist frequency and higher harmonics) far above the bandwidth of the S&Hs. An advantage of this method is that all delays in the system are compensated.

Harpe *et al.* Calibration [12]

Harpe *et al.* present an on-chip measurement and correction method of gain errors, offsets and time-skew errors in TI-ADCs. The measurement and processing is done in the digital domain and that information is used to control analog parameters for gain, offset and timing in the circuit. The use of foreground mode makes it possible to switch off the power of the calibration logic to save power. The method uses a DAC to apply a deterministic signal to the TI-ADC for measurements as shown in figure 4.10.

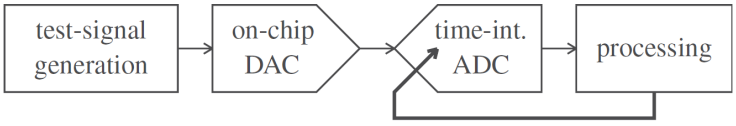


Figure 4.10: System overview of Harpe *et al.* calibration method.

The deterministic input signal described is a pseudo-random maximum-length sequence (MLS) with length m. An MLS sequence is chosen for three reasons: it has a frequency spectrum up to the Nyquist frequency, it is always periodic and it is easy to implement in hardware. The length m is chosen such that it is relatively prime to the number of channels p. In this way after $m{\cdot}p$ sample moments, each symbol of the sequence is applied once to each channel. After reordering the output data the response of each channel to the input sequence can be determined separately. Because of this, the mismatch information of each channel is also available separately.

One channel is taken as a reference to calibrate the other channels. The offset is estimated by averaging the difference between the reference channel and the channel to be corrected over the length of the sequence. Based on the estimated average offset error the gain-mismatch is calculated by subtracting the estimated offset from a sample of the to be corrected channel before dividing by the corresponding reference sample. The estimated average gain-mismatch is then

calculated by averaging the gain-mismatch over m samples. With the offset-mismatch and gain-mismatch known, together with properties of the output signal of the DAC, crosscorrelation is used to estimate the time-skew.

This is a method with little overhead, is very well expandable to any number of channels, it has no accurate calibration signal and no restriction to the to be converted input signal. Nevertheless, because foreground mode is chosen, the method is not capable of correcting time varying clock-skew.

Mesadi's Calibration [2]

Mesadi proposes the mixed calibration architecture for a N-channel TI-ADC shown in figure 4.11.

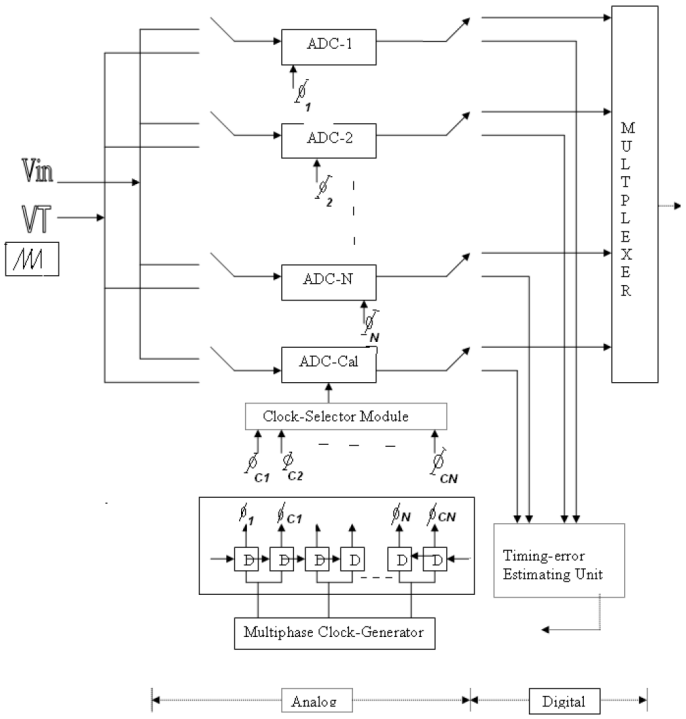


Figure 4.11: Mesadi's mixed calibration architecture.

Mesadi has chosen to use an extra sub-ADC (ADC-Cal) to replace the sub-ADC calibrated at that moment as in [9] but instead of correcting by digital interpolation, the sample clock is corrected directly by a controllable delay line. Also the way in which the sub-ADCs alternate to properly sample the input signal and the calibration signal in parallel differs. For calculating the mismatch, a periodic ramp signal with frequency f_s is used, at this frequency each sample should ideally have the value 0. A sample from the periodic ramp that is not 0 indicates a timing-mismatch which can be calculated because the slope of the periodic ramp is known. The calculated mismatch is then fed back to the controllable delay line and the timing is corrected. In a normal cycle of N samples, all N sub-ADCs have processed the input signal once. Mesadi's method repeats after $2N+1$ cycles. During cycle one of a repetition ADC-Cal and ADC-1 sample simultaneously but ADC-Cal samples the input signal and ADC-1 samples the calibration signal and is calibrated. During the second cycle ADC-Cal and ADC-1 sample again simultaneously but ADC-Cal samples the calibration signal and is calibrated and ADC-1 samples the input signal. At the third cycle ADC-Cal and ADC-2 sample simultaneously, ADC-Cal replaces ADC-2 and ADC-2 is calibrated. During the fourth cycle ADC-Cal and ADC-2 sample again simultaneously, but ADC-Cal is calibrated and ADC-2 operates normally. This continues until all sub-ADCs are

calibrated in cycle $2N$. Cycle $2N+1$ is a cycle without calibrating any sub-ADC to be able to continue with cycle one. During the cycles ADC-Cal samples simultaneous with ADC-X twice in a row and then twice in a row with ADC-X+1 etc. This results in a very complex sample clock for ADC-Cal. Mesadi solves this problem by letting ADC-Cal sample at the positive and negative edge of the sample clock.

The improvement compared to [9] is that drift in ADC-Cal is also calibrated, but with overkill because in $2N+1$ cycles the sub-ADCs ADC-1 to ADC-N are calibrated once where ADC-Cal is calibrated N times. Furthermore the calibration channel has to react on the negative and positive edge of the sample clock or the sample clock gets complex. The periodic ramp signal is running at f_s which is twice the Nyquist frequency and therefore the S&Hs needs at least twice the bandwidth needed in normal operation. Another problem can be mismatch of the corner frequencies of the S&Hs. The periodic ramp signal has a large bandwidth and the high frequency components can be filtered differently for each channel. Therefore, the time signal is shaped differently for each channel and the samples are inaccurate, which leads to an inaccurate correction estimation.

4.6 Comparison

Table 4.1 shows the comparison of the discussed methods in section 4.5 according to the comparison criteria of section 4.4. For all shown methods, the number of channels is scalable.

Method	Calibration		Criteria			
	Domain	Mode	Complexity	Oversampling Ratio	Test Signal	Input Restrictions
Wu and Black	All Ana.	Background	Low	N/A	No	No
Iroaga *et al.*	All Dig.	Background	High	2	Yes	No
Seo *et al.*	Mixed	Background	Medium	1	No	Yes
Liu *et al.*	Mixed	Foreground	Low	1	Yes	No
Harpe *et al.*	Mixed	Foreground	Medium	1	Yes	No
Mesadi	Mixed	Background	Medium	1	Yes	No

Table 4.1: Comparison of the discussed calibration methods.

From all different techniques discussed in this chapter, the calibration technique seems the most promising to correct timing errors. Most of them can be combined with other techniques to correct gain-mismatch and offset-mismatch, some of them already take care of these other errors, like the method of Harpe *et al.*. Depending on the architecture of the method, it can be used for a fixed number of channels or is easy to expand to any number of channels.

Foreground mode interrupts the conversion to detect the mismatch and correct it. This is done mostly because a known input signal is used to measure the mismatch at the output and then the sample clock is corrected. Therefore, foreground mode is almost always used for mixed signal calibration methods. The drawback of the the foreground mode is that it either has to interrupt the conversion or can't correct mismatches that vary in time. But it has the advantage of removing any restrictions from the analog input signal.

Background mode has the advantage that it is able to correct time varying mismatches without interrupting the conversion. Because background mode is widely used in all calibration domains, the comparison is given for each calibration domain.

All analog calibration adds little hardware to the TI-ADC, but it usually is not able to correct mismatches in path delays because the measurement is done right after the generation of the clock. On the other hand there is no extra calibration signal needed and there are no constraints for the analog input signal.

All digital calibration requires a digital interpolation filter for the correction, most often these are quite complex and power hungry.

Mixed calibration has a complexity between all analog calibration and all digital calibration, it requires less hardware than all digital calibration because no digital interpolation filter is needed for correction. On the other hand detection is often more complex in the digital domain than in the analog domain, but since the mismatch detection is done after conversion, the full delay in the channels is detected and can be corrected.

The digital mismatch detection is done either by estimation or by measurement. Estimation usually means that the analog input signal needs to have some properties to be able to estimate the mismatch by for example correlation. Measurement requires a known signal like the techniques in foreground mode and in order to make it background mode, an extra sub-ADC is required to replace the sub-ADC under test.

Chapter 5

Newly Developed Timing Correction Method

This chapter presents a newly developed background mode, mixed domain calibration method for timing correction of TI-ADCs. The method improves the SFDR and SNDR of the TI-ADC for time-constant and time-varying timing mismatches. The chapter starts with the chosen calibration type and specifications. After that the developed measurement and correction algorithm is explained. In the third section the designed Simulink model of the algorithm is described. The last section describes the results of the simulation from the Simulink model.

5.1 Choices

Chapter 4 evaluated some correction methods presented in literature. Based on this evaluation the following choices for a new correction method are made:

- Method: Calibration
- Domain: Mixed
- Mode: Background
- Complexity: Extra sub-ADC
- Oversampling ratio: 1
- Test signal: yes
- Input restrictions: no

A calibration method is chosen because this is the most promising one. The mixed domain has the advantages of being accurate and relatively easy to implement. Background mode is chosen because of the ability of correcting drift without interrupting the conversion process. A test signal is used in combination with an extra sub-ADC to be able to work in background mode and to have no restrictions on the analog input signal except the Nyquist criterium. Furthermore the algorithm should easily be expandable to any number of channels.

5.2 Algorithm

The calibration method uses an extra sub-ADC and a reference sine wave for measuring the timing error. The sub-ADC under test is replaced by the extra sub-ADC and the reference sine wave is applied to the sub-ADC under test. The sample of the reference signal is then used to estimate

the timing error in the digital domain. This timing error is then corrected in the analog domain by adjusting the delay of the sample clock of the sub-ADC under test. The normal conversion process in not interrupted because the extra sub-ADC is sampling the normal analog input signal.

The designed algorithm is a combination of the calibration methods presented by Liu *et al.* [11] and Iroaga *et al.* [9]. The block diagram is comparable with the method from Mesadi [2], but with a different timing for the calibration and another type of calibration signal. The block diagram of the designed algorithm is shown in figure 5.1.

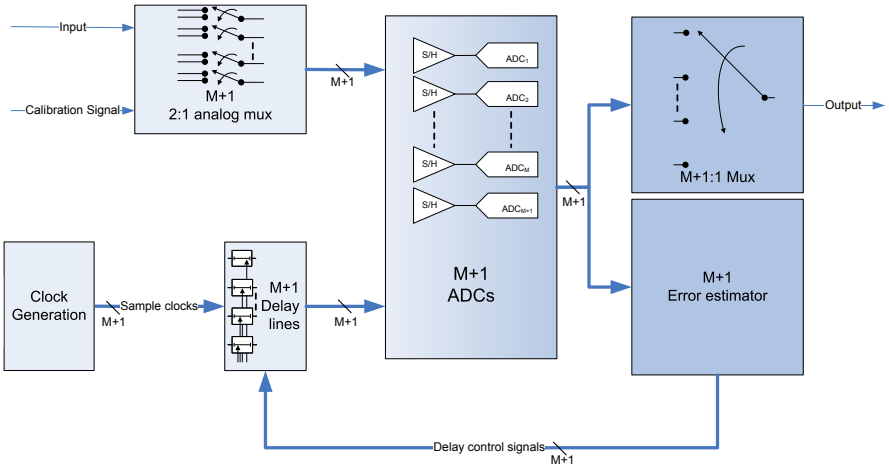


Figure 5.1: Block diagram of proposed algorithm.

5.2.1 Overview

The proposed algorithm uses M+1 2-to-1 analog multiplexers to select between the analog input signal and the calibration signal for each channel. The clock generation block generates M phase shifted sample clocks with a frequency of f_s/M for the M sub-ADCs (ADC-0...ADC-(M-1)) and a sample clock for the replacement sub-ADC (ADC-R) with a frequency of $f_s/(M+1)$. The sample clocks are delayed by controllable delay lines. The multiplexed analog signals and sample clocks are fed to the M+1 identical sub-ADCs. The outputs of the sub-ADCs are multiplexed to the output signal and the samples from the calibration signal are used in the error estimator to determine the timing error. The error estimator controls the delay lines to correct the timing error.

5.2.2 Calibration Signal

The calibration signal is sampled synchronously with ADC-R. Therefore, the calibration signal requires a frequency of $f_s/(M+1)$. This frequency is always below the Nyquist frequency of the TI-ADC. To reduce the influence of the S&H's bandwidth, a sine wave, which by definition has no higher harmonics, is chosen as calibration signal. To get the highest accuracy possible the peak-peak voltage of the sine wave should be equal to the max swing of the TI-ADC.

5.2.3 Timing

In a normal cycle of M samples, all standard M sub-ADCs have processed the input signal once. This takes MT_s seconds. During the first cycle ADC-R replaces ADC-1, and ADC 1 samples the calibration signal. During the second cycle ADC-R replaces ADC-2 and ADC-2 samples the calibration signal. etc. After M cycles all standard M sub-ADCs are replaced and calibrated once. In cycle M+1 no ADC is calibrated.

In cycle M+2 ADC-R does not replace ADC-1 as expected but is calibrated.

In cycle M+3 ADC-2 is replaced by ADC-R and ADC-2 is calibrated. After $2M$+2 cycles ADC-R and ADC-1 are calibrated once and the other sub-ADCs are calibrated twice and during two of the cycles no sub-ADCs are calibrate. In cycle $2M$+3 ADC-1 is replaced by ADC-R. In cycle $2M$+4 ADC-2 is not replaced but ADC-R is calibrated. This is shown in figure 5.2 for M=4. The conversion-time of the sub-ADCs is assumed to be MT_s. Therefore, the converted sample is seen at the output with one cycle delay.

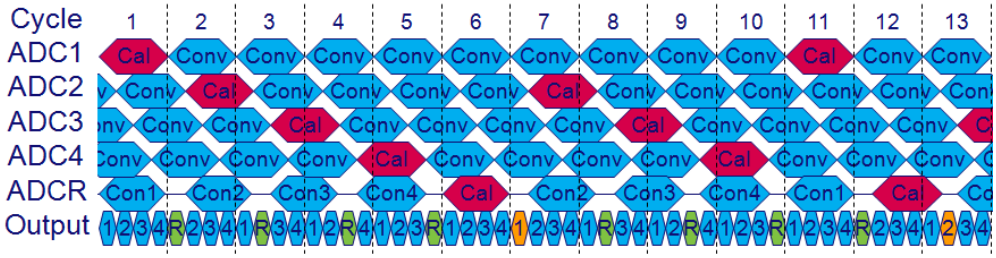


Figure 5.2: Timing for 4-channel TI-ADC, 12 cycles.

This is continued for $(M+1)^2$ cycles. After $(M+1)^2$ cycles all M+1 sub-ADCs are calibrated M times including ADC-R and during M cycles no calibration was done. These $(M+1)^2$ cycles are repeated continuously. As per cycle M samples are taken, one repetition takes $(M+1)^2M$ samples or $(M+1)^2MT_s$ seconds.

5.2.4 Error Estimator

In the ideal case without timing mismatch, ADC-i samples the calibration signal at a zero crossing (figure 5.3). However when there is a delay in the sampling clock the output of ADC-i (γ_i) will

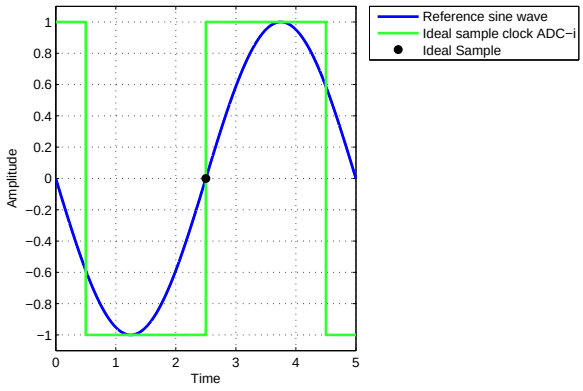


Figure 5.3: Ideal sampling of the reference sine wave.

be greater than zero as shown in figure 5.4.

The calibration signal is almost linear around the zero crossings, and has a known slope S:

$$S = A\frac{2\pi f_s}{M+1} \tag{5.1}$$

where A is the amplitude of the calibration signal. Assuming the timing error to be small, the slope can be used to estimate the timing error $\hat{e}_i[k]$ by dividing the output of ADC-i by the slope:

$$\hat{e}_i[k] = \frac{\gamma_i[k]}{S} \tag{5.2}$$

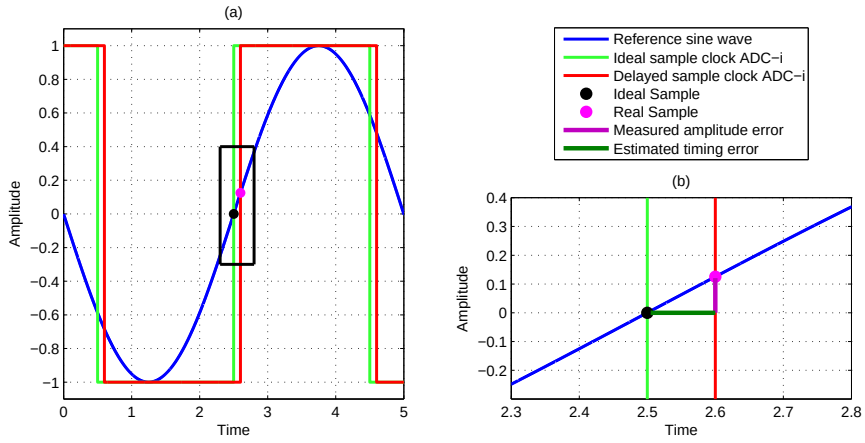


Figure 5.4: Error estimation, (a) One period of the reference sine wave with an ideal sample clock and a delayed sample clock, (b) enlargement of the rectangle box in (a).

The estimated timing error is then subtracted from the current delay:

$$D_i[k] = D_i[k-1] - \hat{e}_i[k] \tag{5.3}$$

Since the delay lines that are used for the correction have positive delays, the sample clocks can not be accelerated. In order to be able to delay and 'accelerate' the sample clock, a default delay (d) is used as initial value for the delay lines ($D_i[0] = d$). The delay line is then adjusted with the estimated delay to correct the timing error. The correction is done in one step. Therefore, it could be sensitive to timing jitter. The timing jitter in the realized demonstrator (chapter 6) is smaller than the resolution of the delay lines used. Therefore, the correction is almost insensitive to timing jitter.

5.3 Model in Simulink

The algorithm is modeled for a 4-channel TI-ADC (M=4) at a sample rate of 1GSps (f_s=1GHz) in a Simulink model. The overview of the model is shown in figure 5.5.

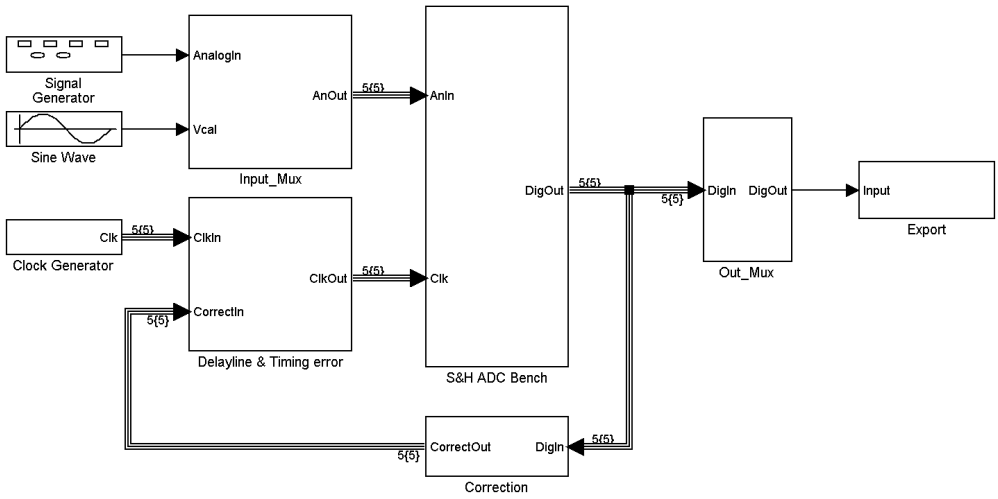


Figure 5.5: Simulink model of TI-ADC with developed algorithm: Top level.

The analog input signal is generated by a signal generator. The calibration signal is a sine wave of $f_s/(M+1)$=200MHz. The clock generator generates 4 phase shifted sample clock signals at f_s/M=250MHz and a sample clock at $f_s/(M+1)$=200MHz for the replacement sub-ADC. The output is exported to the MATLAB workspace for analysis. The other blocks are discussed separately hereafter.

5.3.1 Input Multiplexers

The model of the input multiplexers is shown in figure 5.6.

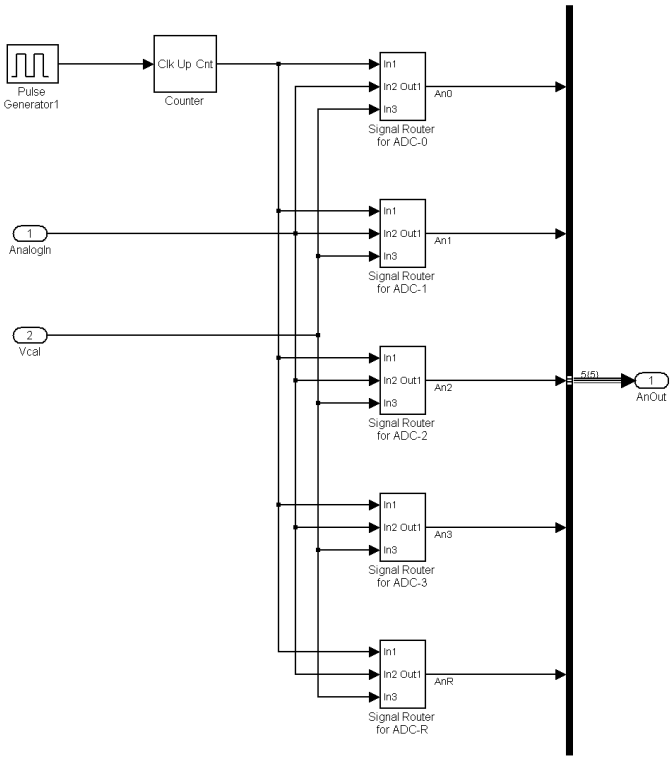


Figure 5.6: Simulink model of the input multiplexers.

The signal routers multiplex the analog input signal to the sub-ADCs except when a sub-ADC is under calibration, then one period of the sine wave is routed to that sub-ADC. One repetition of the algorithm takes $(M+1)^2M/f_s$=100ns which is 20 periods of the sine wave. Therefore, the signal routers are controlled by a counter counting from 0 to 19 at 200MHz. The 200MHz clock for the counter is generated by a pulse generator. Each signal router consists of a embedded m-file and a 2-to-1 multiplexer. The embedded m-file decides which signal is routed to the sub-ADC based on the value of the counter. The embedded m-files are added in appendix A.3.

5.3.2 Delay Line & Timing Error

Figure 5.7 shows the model of the delay lines with the addition of the timing error.

The sample clocks are delayed by the delay lines. The delay lines are controlled by the correction signals and a timing error. In the timing-error generator an arbitrary error can be added. For the generation of a fixed clock skew a constant value is applied.

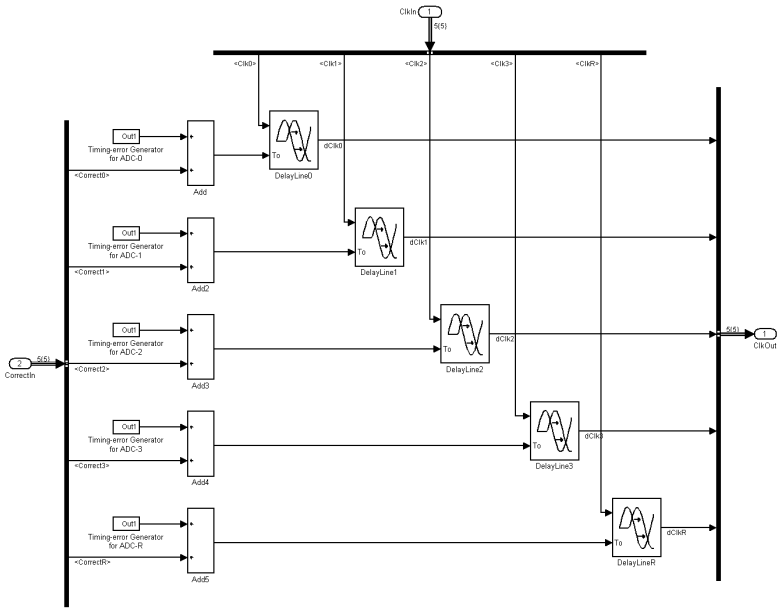


Figure 5.7: Simulink model of the delay lines & timing error.

5.3.3 S&Hs and sub-ADCs

The S&Hs and sub-ADCs are modeled as shown in figure 5.8. The S&Hs are ideal and the sub-ADCs are modeled by ideal ADC quantizers like the models of chapter 3.

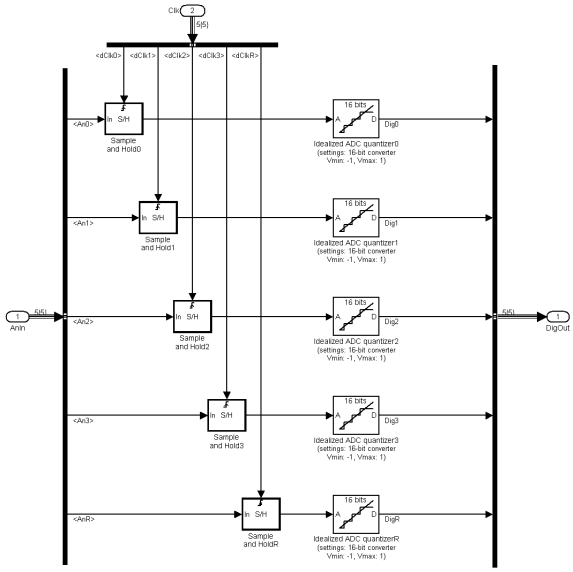


Figure 5.8: Simulink model of the S&Hs & sub-ADCs.

5.3.4 Output Multiplexer

The routing of the output multiplexer is controlled by an embedded m-file (appendix A.3). The embedded m-file determines the routing according to the value of the counter. The counter runs

at f_s and counts from 0 to 99, the algorithm repeats after 100 samples. The clock of the counter is generated by a pulse generator. The model is shown in figure 5.9.

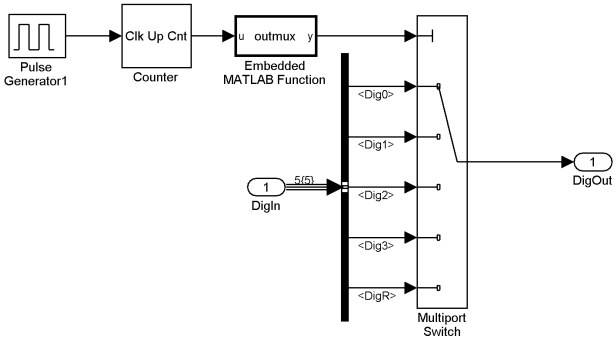


Figure 5.9: Simulink model of the output multiplexer.

5.3.5 Correction Estimation

In the correction block the error is estimated from the output of the sub-ADCs according to the method described in subsection 5.2.4. The model of the correction is shown in figure 5.10.

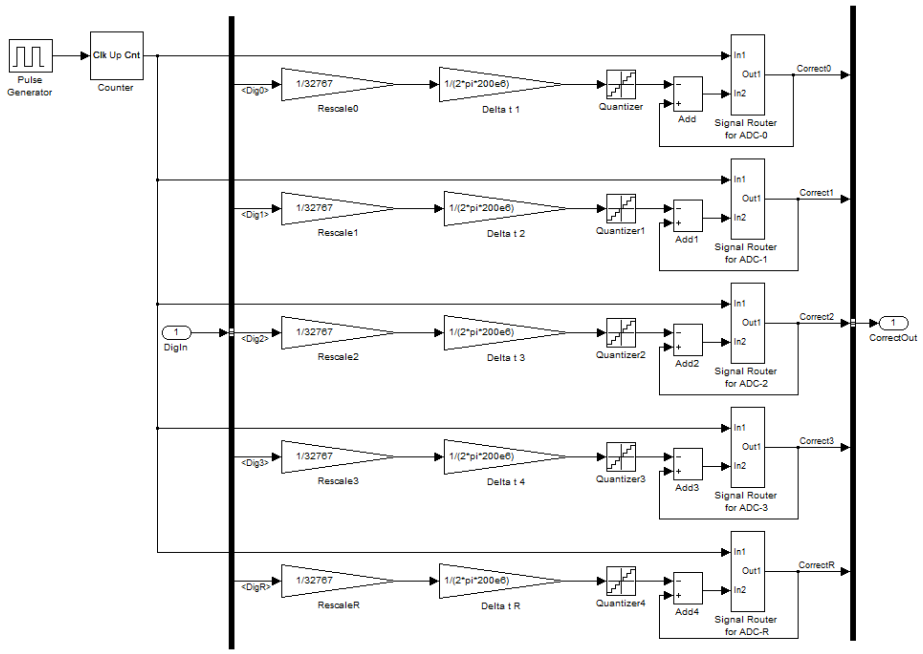


Figure 5.10: Simulink model of the correction estimation.

The output of the sub-ADCs is first scaled to a signal with amplitude 1. Then $\hat{e}_i[k]$ is calculated according to equation (5.2). The estimated error is quantized with a timing resolution that does not exceed the time steps of the simulator. For this simulation the time steps are set to 1ps, this corresponds with an accuracy of 0.1% of the sample time T_s, and the quantization steps become 10^{-12}. After quantization, the estimated error is subtracted from the previous delay.

Because the calibration samples do not have a fixed sample rate a signal router block resamples the data from the sub-ADC and selects the data corresponding to the sample from the calibration signal. The signal router blocks are controlled similarly to the input signal routers.

5.4 Simulation Results

Simulations were done for a 1GSPS TI-ADC, with f_0=1013f_s/2048≈494.6MHz and timing-mismatch (Δt=[−0.009 −0.002 −0.008 0.004 0]ns). The replacement sub-ADC and the timing of the algorithm influences the system by spreading the spurious tones. The timing of the algorithm repeats after $(M+1)^2M$ samples, whereas a normal TI-ADC repeats after M samples. Therefore, the developed algorithm acts as a $(M+1)^2M$ channel TI-ADC. The results of a simulation without correction, clearly showing the spreading of the tones, is shown in figure 5.11.

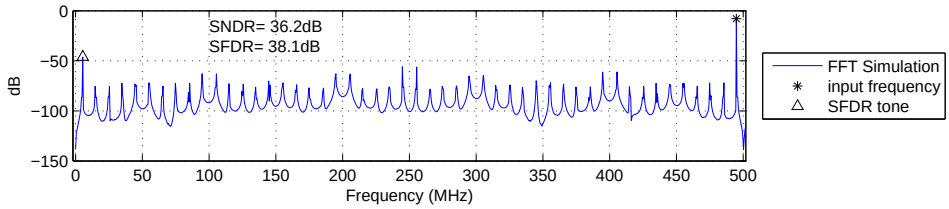


Figure 5.11: Spectrum of 4-channel TI-ADC, with spreading, with timing-mismatch (Δt=[−0.009 −0.002 −0.008 0.004 0]ns), without correction, f_0≈494.6MHz, 2048 point FFT.

Compared to the simulation with timing-mismatch in chapter 3 (see figure 3.5), the SNDR is improved by 0.4dB to 36.2dB and the SFDR is improved by 1.5dB to 38.1dB. The improvement of SFDR is due to the fact that the timing algorithm reduces the correlation between the channels like the channel randomization method described in chapter 4. This improvement in SFDR due to spreading can be considered as an additional advantage of the proposed timing (see figure 5.2). The three largest spurs are still at $m/(M)f_s \pm f_0$ because of the sample period of the M normal sub-ADCs. The next set of large spurs are at $m/(M+1)f_s \pm f_0$ because of sample period of the replacement sub-ADC. The change in SNDR is due to the use of an extra sub-ADC. The extra sub-ADC introduces timing-mismatch also. Therefore, it can improve or decrease the distortion in the TI-ADC.

The simulation with the full algorithm applied is shown in figure 5.12.

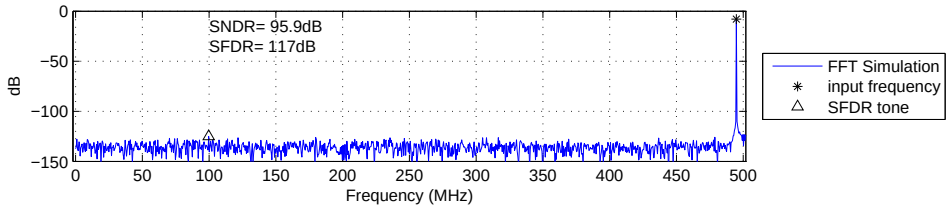


Figure 5.12: Spectrum of 4-channel TI-ADC, with correction algorithm, with timing-mismatch (Δt=[−0.009 −0.002 −0.008 0.004 0]ns), $f_0 \approx$ 494.6MHz, 2048 point FFT.

The SNDR and SFDR after calibration are respectively 95.9dB and 117.0dB, this is an improvement of 60dB in SNDR and 80dB in SFDR compared to the simulation with timing-mismatch in chapter 3. These results are equal to the simulation of a 4-channel TI-ADC without mismatch.

Chapter 6

Hardware Realization of a Demonstrator

To verify the results of the simulations in chapter 5 a demonstrator of the developed method is realized in hardware. The realization is described in this chapter. The first section discusses the configuration of the demonstration model with off-the-shelf components. The second section describes the details of the designed printed circuit board(PCB). The last section shows the design of the Field Programmable Gate Array(FPGA) software.

6.1 Configuration

The configuration for the demonstrator of the developed method is chosen comparable with the Simulink model. The number of channels is the same but the sample rate is scaled down with a factor thousand to be in a frequency range compatible with off-the-shelf components. Therefore, the realized demonstrator is a 4-channel TI-ADC at 1MSps. The demonstrator consists of two boards, a newly designed PCB which contains the sub-ADCs and the analog part of the system and a FPGA board for the digital part of the system. This is represented in figure 6.1. The FPGA board is a Xilinx Spartan-3E starter kit, which is connected to the PCB through a 100 pin connector of which 43 pins are useable I/O pins.

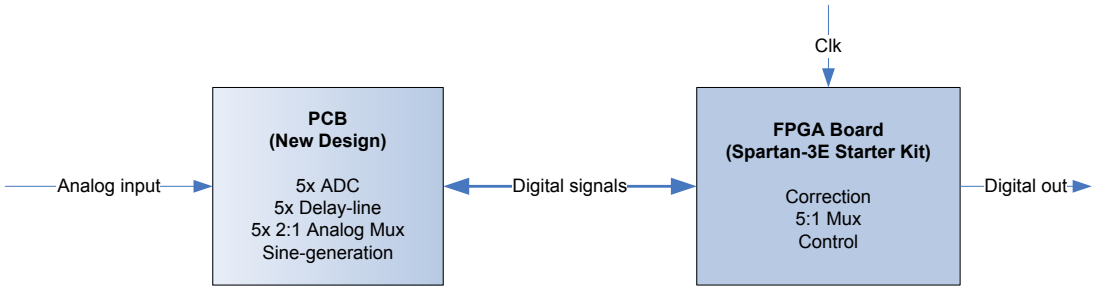


Figure 6.1: Overview Hardware Realization of the demonstrator.

6.2 Designed PCB

The designed PCB contains mainly analog components that are digitally controlled by the FPGA board. The analog components use a 5V supply and the FPGA uses a 3V3 supply. Therefore,

buffers are included on the PCB to communicate between the FPGA and the analog components. A block diagram of the designed PCB is shown is figure 6.2.

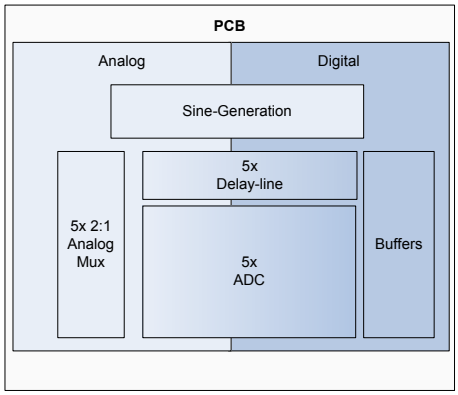


Figure 6.2: Block diagram of designed PCB.

In the following subsection the separate blocks of the block diagram are discussed with their sub-circuits and used components. The last subsection shows the designed PCB.

6.2.1 Hardware Realization of the System Blocks

sub-ADCs

The sub-ADCs require a manually tunable gain and offset, because they are not corrected by the algorithm and the mismatch should be small. Since the sample clock has to be controlled by the algorithm, the sample clock has to be external. The sample rate of the sub-ADCs has to be at least 250kSps. In order to distinguish the spurs from the quantization noise the sub-ADCs should have a resolution of at least 16-bit, according to the simulations results of chapter 5. Serial data is required because of the limited I/O pins available on the FPGA board.

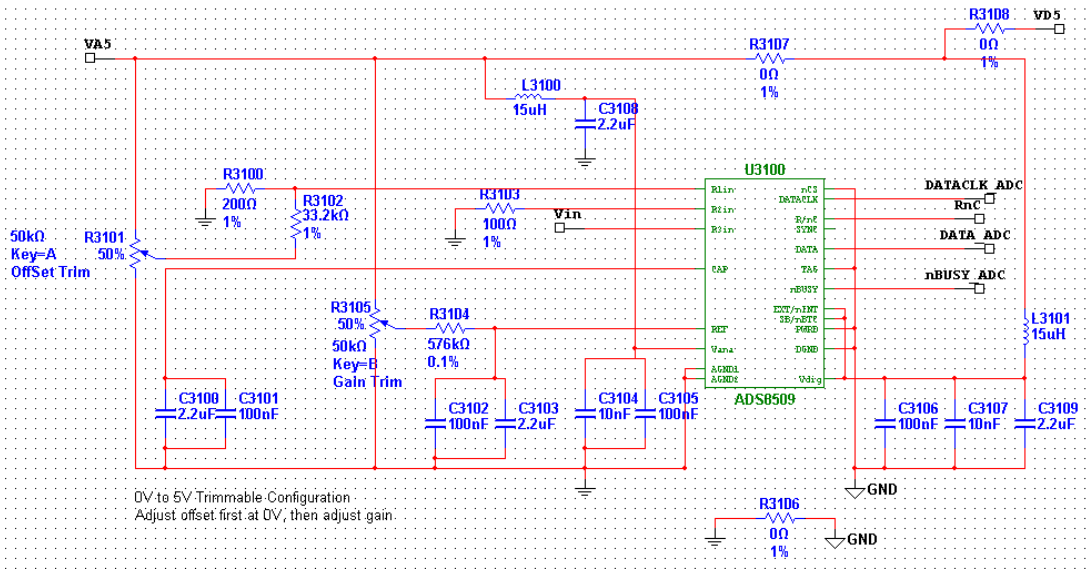


Figure 6.3: Sub-ADC circuit configured for 0 to 5V input with adjustable gain and offset.

Based on the requirements the Texas Instruments ADS8509 is chosen for all sub-ADCs. With a small passive circuit their gain and offset can be controlled by two potentiometers. The sub-

ADCs are configured for 0 to 5V single ended input (figure 6.3). The ADS8509 uses a 3-wire serial communication for the 16-bit digital output.

Controllable Delay Lines

The accuracy and range of the controllable delay lines define the accuracy and range of timing-mismatch correction. An accuracy of 0.1% of the sample time is used in the Simulink model of chapter 5 and shows good results. To be sure that the timing-mismatch spurs are clearly visible in the spectrum of the demonstrator a range of ±10% of the sample time is required.

For a 1MSps TI-ADC a time-mismatch of 0.1% is equal to a step size of 1ns. A range of ±10% results in a delay range of at least 200ns. Maxim-IC DS1023 is an 8-bit digitally controlled delay line(DCDL) with 256 steps available in five versions: 0.25ns steps, 0.5ns steps, 1ns steps, 2ns steps and 5ns steps. The ranges of the DCDLs are respectively 63.75ns, 127.5ns, 255ns, 510ns and 1275ns. The 1ns step version satisfies the requirements. Therefore, the DS1023-100 with 1ns steps is chosen.

Sine Wave Generation

The calibration signal required is a 200kHz sine wave with a peak-peak voltage of 5V and a 2.5V offset. The purity of the signal is less important as long as it is purely periodic and more or less linear around its zero crossings. The sine wave has to be synchronous with the sampling clock of the extra sub-ADC. Therefore, the sine wave for calibration is generated on board and is controlled by the FPGA just as the sampling clocks. The sine wave is generated by filtering a square wave. The FPGA controls a single pole dual throw (SPDT) analog switch (Intersil ISL84544) used as a 1-bit DAC, creating the square wave. The control signal is a 200kHz digital clock signal. The square wave from the DAC is filtered by a fifth order Butterworth LC filter with a corner frequency of 250kHz. The DC component of the generated sine wave is blocked by a high pass filter with a corner frequency of 10kHz. Then the DC offset is set to 2.5V and finally the sine is buffered with a AD744JR operational amplifier before it is fed to the analog multiplexers. The corresponding circuit is shown in figure 6.4. The fifth order Butterworth LC filter used is for

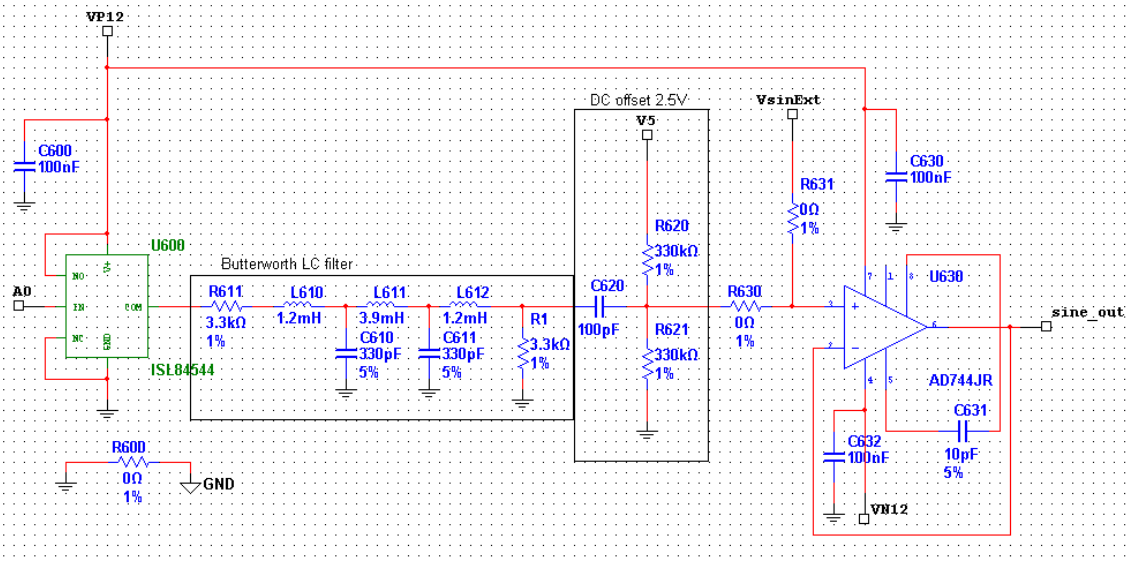


Figure 6.4: Sine wave generation circuit.

demonstration purpose, in a real system with area constraints the filter can be less complex as the exact waveform is not critical. Another option is to generate a digital sine wave and filter this with a RC filter. This options is not used for practical reasons.

2-to-1 Multiplexers

For the input 2-to-1 multiplexer a bandwidth of at least 500kHz and a low on-resistance is required. The Intersil ISL84544 SPDT analog switch is used as input multiplexer.

6.2.2 PCB

During the PCB design special attention was paid to electro-magnetic compatibility (EMC) related issues. All power lines to the integrated circuits (ICs) on the board are filtered and decoupled close the to ICs' power pins. A 4-layer PCB is used with one ground layer, containing an analog ground plane and a digital ground plane. The ground planes are connected close to each sub-ADC. The routing for each channel is as much identical as possible. The fabricated PCB is shown in figures 6.5 and 6.6.

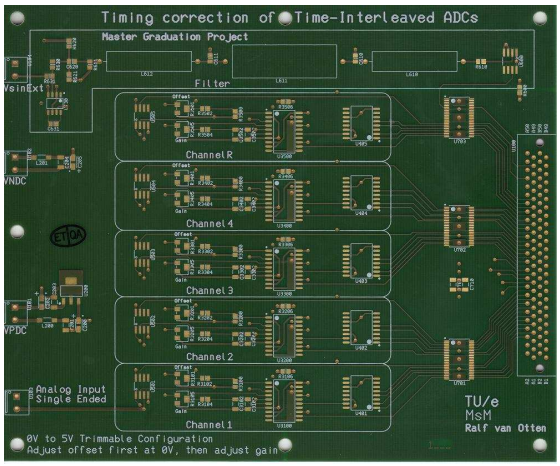


Figure 6.5: Fabricated PCB top view.

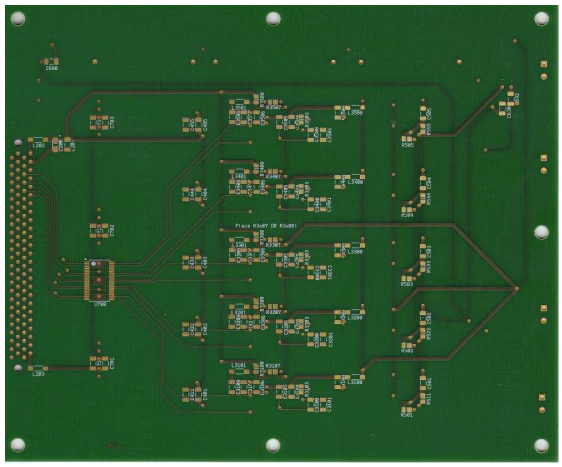


Figure 6.6: Fabricated PCB bottom view.

6.3 Designed FPGA Software

The FPGA is used for all digital communication and data processing needed for the 4-channel TI-ADC with timing correction. The FPGA generates the sample clocks, controls the phase of the sine wave, drives the input multiplexers, reads out the sub-ADCs and programs the DCDLs. The software is hierarchically built where the top schematic connects all blocks, synchronizing the full design.

The top schematic of the FPGA software is shown in figure 6.7. It contains the following blocks:

- DCMs: Digital Clock Managers
- Sample_clocks: Sample clock generator
- inpmuxdrv: Input multiplexers driver
- channel: Control for each channel
- outputmux: Output multiplexer
- serialdataout16b: 16-bit Parallel data to 3-wire serial data

The DCMs are used to generate three internal clocks from the onboard 50MHz clock. The channel block contains the reading of the sub-ADC, the correction algorithm and the programming of the DCDLs. Each of the blocks are discussed separately in the next subsections.

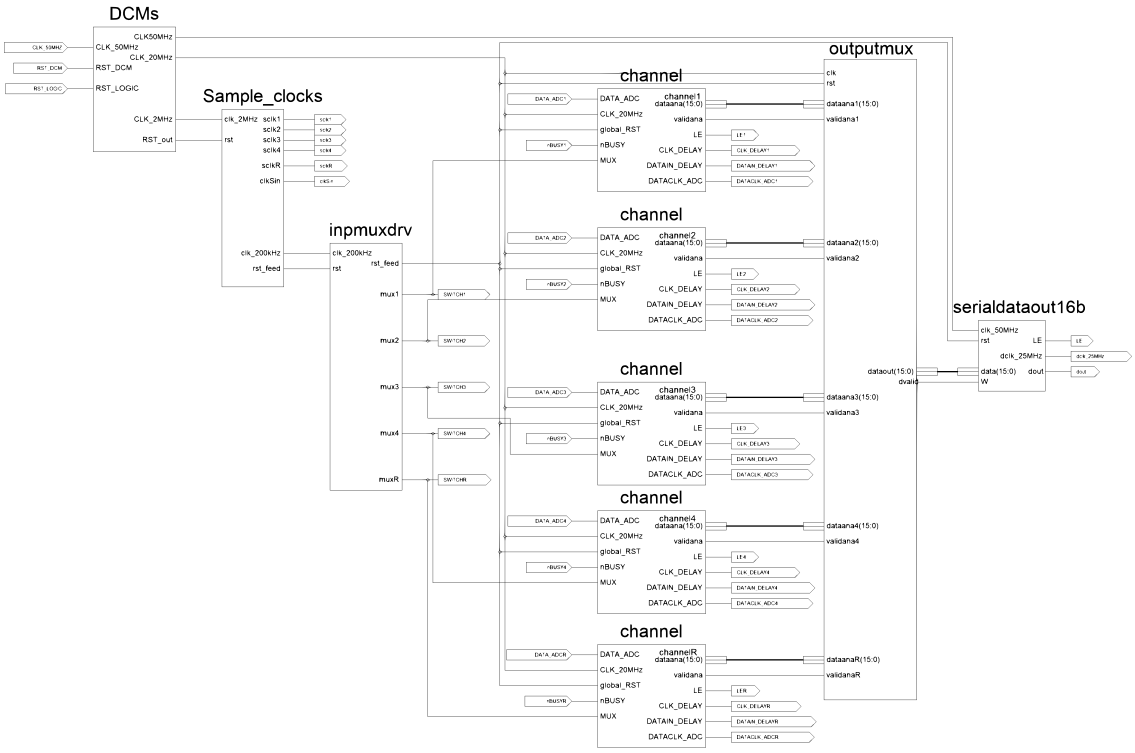


Figure 6.7: Top Schematic of FPGA software.

6.3.1 Digital Clock Managers

The FPGA board has a 50MHz clock onboard. With the use of DCMs three clocks are generated from this onboard clock. DCM is a hardware block inside the FPGA able to multiply and/or divide the clock frequency. The output clock of the DCM is 50% high and 50% low and in phase with the input clock. For the design a 2MHz clock is required for the generation of the sample clocks and the control of the sine wave. For the communication with the sub-ADCs and DCDLs the control block requires a clock of 20MHz. This is also used for the correction algorithm and the output multiplexer. The control for the serial 16-bit output requires a clock frequency of 50MHz. One DCM is not able to generate these three clocks at once; therefore, a cascade of two DCMs is used as shown in figure 6.8.

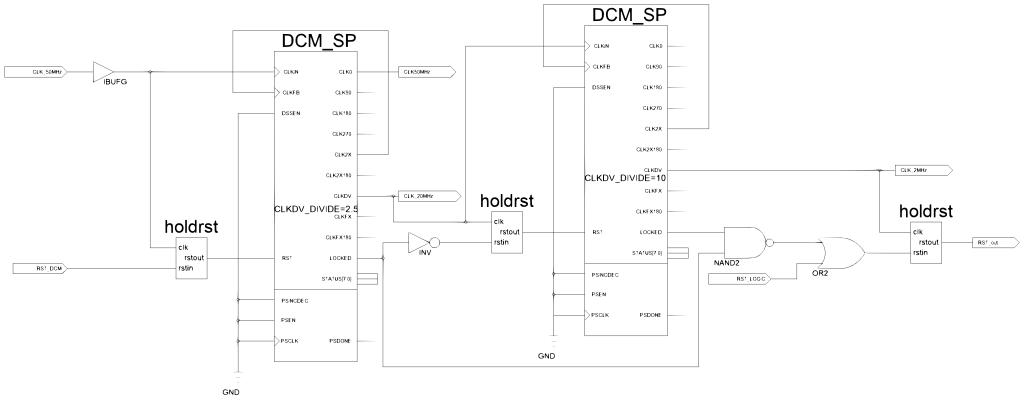


Figure 6.8: Schematic of DCMs.

The first DCM generates the 50MHz clock and 20MHz clock from the onboard clock. The second DCM generates the 2MHz clock by dividing the 20MHz clock by 10. The DCMs require the reset port to be active for at least three clock periods of the input clock. Therefore, a block called holdrst is designed that holds the reset for three clock periods after the reset turns inactive. The VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) design of the holdrst block is added in appendix B.1. The DCMs generate a locked signal when the output clocks are phase locked to the input signal. The inverse of the locked signal of the first DCM is used to reset the second DCM, the holdrst block is used to hold the signal for 3 clock cycles. All other logic is reset until both DCMs are locked for three clock cycles of the 2MHz clock. There are two external reset possibilities built in, one is to reset the full FPGA including the DCM and the other is to reset all logic except the DCMs.

6.3.2 Sample Clock Generator

The sample clock generator uses the 2MHz clock and a standard 4-bit counter to generate the four 90 degrees phase shifted sample clocks of 250kHz for ADC-1 to ADC-4 and the 2MHz clock with another 4-bit counter to generate three 200kHz signals with different phases. The 200kHz signals are the sample clock for ADC-R, the control of the sine wave, and a clock signal for the input multiplexer driver. The reset is held until all clocks are running correctly. The schematic overview is shown in figure 6.9.

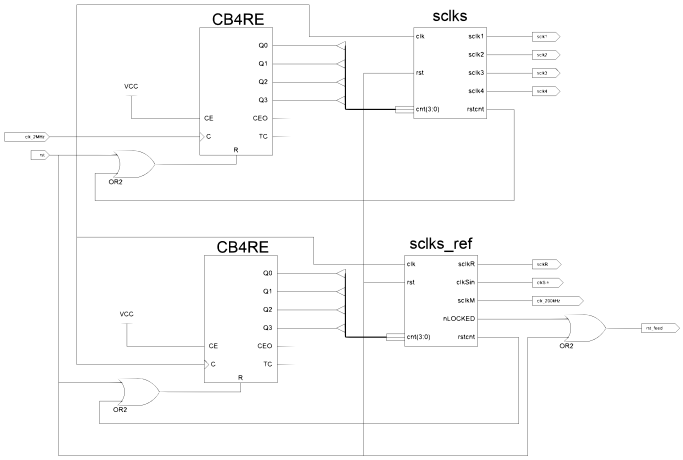


Figure 6.9: Schematic of Sample clock generator.

The first counter counts from 0 to 7 and is reset by the sclks block. The sclks block generates the four 250kHz sample clocks based on the value of the counter and resets the counter when it reaches 7. The VHDL design of the sclks block is added in appendix B.2.

The second counter counts from 0 to 9 and is reset by the sclks_ref block. Based on the values the sclks_ref block generates the three 200kHz output signal and holds the global reset until the counter is reset. The counter is reset when it reaches 9. The VHDL design of the sclks_ref block is added in appendix B.3.

6.3.3 Input Multiplexers Driver

The input multiplexers driver uses the 200kHz clock with a standard 8-bit counter to select the input for the sub-ADCs. The counter counts from 0 to 19 because after 20 periods of 200kHz the algorithm is repetitive. The counter is reset by the inmudrv driver block that drives the analog input multiplexers. The schematic overview is shown in figure 6.10.

The inmudrv block drives the input multiplexers based on the value of the counter. It resets the counter when it reaches 19. The VHDL design is added in appendix B.4.

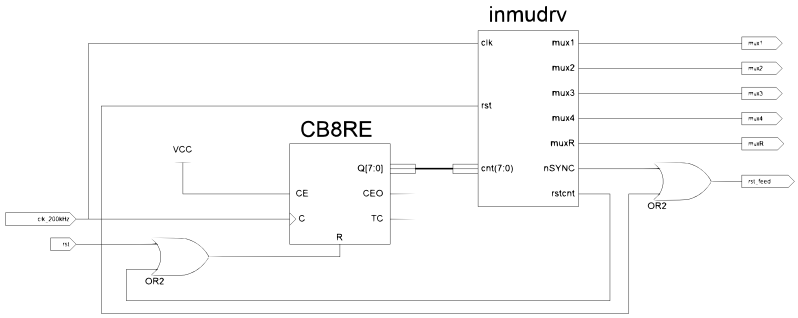


Figure 6.10: Schematic of input multiplexers driver.

6.3.4 Control for each Channel

The control for each channel is split into five blocks as shown in figure 6.11.

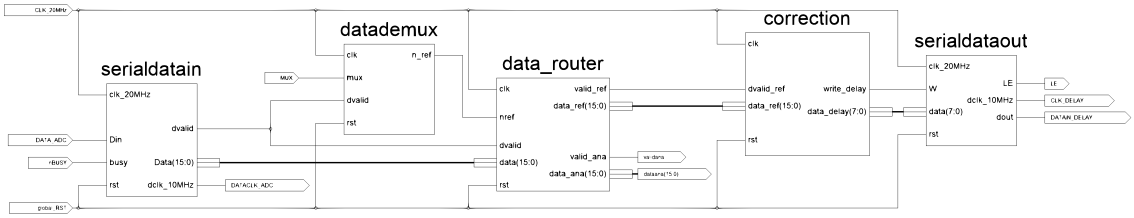


Figure 6.11: Schematic of Channel Control.

The 3-wire serial data from the sub-ADC is read by the serialdatain block and converted to parallel data. The datademux block detects if the new data is calibration data or normal data. The data_router splits the input data in calibration data and normal data. The correction block uses the calibration data to estimate the error and calculates the new delay. The serialdataout converts the 8-bit parallel data to 3-wire serial data and programs the DCDL. The blocks are further discussed below.

16-bit 3-wire Serial-to-Parallel Data Input

When the nbusy signal of the sub-ADC has a rising edge the serialdatain block generates 16 clock pulses at 10MHz and reads the 16-bit serial data. The 16-bit data is converted to a 16-bit parallel signal. After all 16-bits are read and parallelized the data valid signal is made active until there is a new rising edge on the nbusy signal and new data is read. The VHDL design is added in appendix B.5.

Calibration Data Detection

Because of the conversion time of the sub-ADCs, the data read when the input multiplexer passes the calibration signal is not the calibration data but the next data is the calibration data. The datademux block therefore waits for a falling edge of the control signal for the input multiplexer. After that the next data read by the serialdatain block is calibration data. When the data becomes invalid the datademux block waits again for the falling edge of the control signal for the input multiplexer. The VHDL design is added in appendix B.6.

Input Data Router

The data_router block sends the incoming data to the correction block when calibration data is detected, otherwise the data is send to the output multiplexer. The VHDL design is added in appendix B.7.

Correction

The correction block estimates the timing error and calculates the correction delay according to the algorithm described in chapter 5. The unsigned calibration data is first converted to signed data and the amplitude error is calculated. The amplitude error is converted to a timing error by dividing it by the slope of the calibration signal. The new delay is calculated by adding the timing error to the previous delay. Finally the calculated delay is sent to the data output block. The VHDL design is added in appendix B.8.

8-bit Parallel-to-3-wire Serial Data Output

The serialdataout block programs the DCDL with the new delay by generating a 10MHz clock for 8 pulses and shifting out the 8-bit parallel data. The VHDL design is added in appendix B.9.

6.3.5 Output Multiplexer

The output multiplexer multiplexes the converted data read from the sub-ADCs to 16-bit 1MSps parallel data. The multiplexer outputs the data in order of arrival at the input, but makes sure that the order is valid. This means that if for example the current data is from channel 2 the new data should be from channel 3 or channel R. But when channel R was active before channel 2 the new data must be from channel 3. The VHDL design is added in appendix B.10.

6.3.6 16-bit Parallel Data to 3-wire Serial Data Output

The serialdataout block outputs the 16-bit multiplexed data by generating a 25MHz clock for 16 pulses and shifting out the 16-bit parallel data. The VHDL design is added in appendix B.11.

Chapter 7

Measurement Results

This chapter shows the measurement results of the realized hardware system from chapter 6 and compares them with the simulation results of chapter 5. In the first section the results are shown when a large timing-mismatch is added on purpose. The second section shows results when there is only intrinsic timing-mismatch. In each section three input frequencies are used, $1391f_s/16384 \approx 84.9$kHz, $4601f_s/16384 \approx 280.8$kHz and $8101f_s/16384 \approx 494.4$kHz. In the third section the measurement results are compared with non-ideal simulation results. Spectra shown in this chapter are generated with a 16384 point FFT. The last section compares the results with the results in literature.

7.1 Large Timing Mismatch Added

In this section channel 1 has an initial mismatch of +12.8% (+128ns) and channel 3 has a mismatch of -12.8% (-128ns), which is generated by programming the error into the DCDLs on reset.

Input frequency 84.899902kHz

In figure 7.1 the single sided spectrum is shown when the input frequency is approximately 84.9kHz, (a) shows it for a normal 4-channel TI-ADC configuration without any correction method and (b) shows it for the 4-channel TI-ADC with the designed correction method.

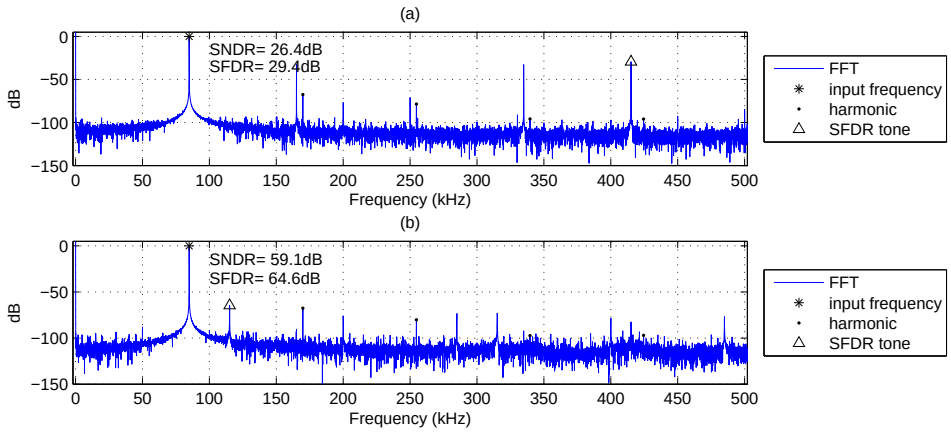


Figure 7.1: Spectrum of 4-channel 1MSps TI-ADC $f_0 \approx 84.9$kHz, Δt_i=[128 0 −128 0 (0)]ns; (a) no correction algorithm, (b) with correction algorithm.

In (a) the three spurs due to timing-mismatch are clearly visible (165.1kHz, 334.9kHz and

415.1kHz). The spurs at 250kHz and 500kHz are caused by offset-mismatch and a spur at 200kHz is caused by the reference sine wave through crosstalk in the input multiplexers. Without correction the SNDR is 26.4dB and the SFDR is 29.4dB.

In (b) the spurs are clearly spread and reduced by the algorithm as expected. The visible spurs of gain-mismatch and timing-mismatch are at 115.1kHz, 284.9kHz, 315.1kHz, 415.1kHz and 484.9kHz. The spurs of the offset-mismatch are moved to 200kHz and 400kHz. This is due to the addition of the replacement sub-ADC with a sample frequency of 200kHz. With the correction method the SNDR is 59.1dB and the SFDR is 64.4dB. This is an improvement in SNDR of 34.3dB and an improvement in SFDR of 25.2dB. The SFDR is limited by gain-mismatch and timing-mismatch.

Input frequency 280.82275kHz

In figure 7.2 the single sided spectrum is shown when the input frequency is approximately 280.8kHz, (a) shows it for a normal 4-channel TI-ADC configuration without any correction method and (b) shows it for the 4-channel TI-ADC with the designed correction method.

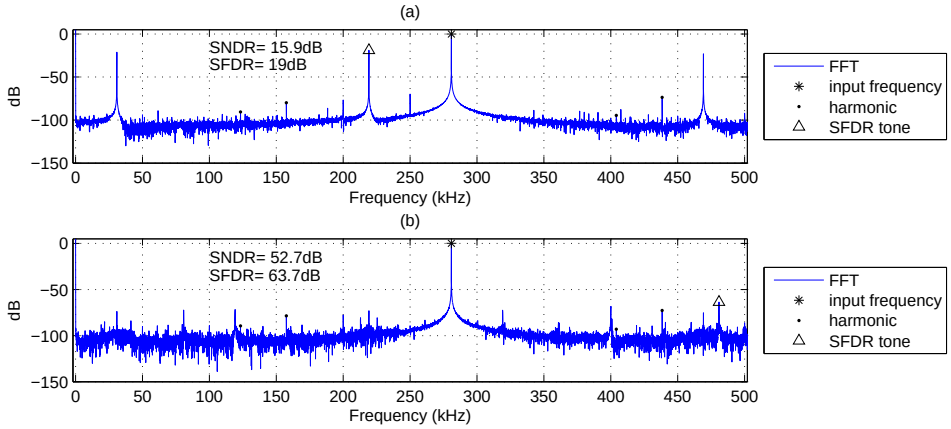


Figure 7.2: Spectrum of 4-channel 1MSps TI-ADC $f_0 \approx 280.8$kHz, $\Delta t_i = [128\ 0\ -128\ 0\ (0)]$ns; (a) no correction algorithm, (b) with correction algorithm.

In (a) the spurs caused by timing-mismatch are clearly visible (30.8kHz, 219.2kHz and 469.2kHz). Offset-mismatch spurs are visible at 250kHz and 500kHz. The crosstalk of the input multiplexers is visible at 200kHz. Without correction the SNDR is 15.9dB and the SFDR is 19dB.

In (b) the spurs are clearly spread and reduced by the algorithm as expected. The visible spurs of gain-mismatch and timing-mismatch are at 30.8kHz, 80.8kHz, 119.2kHz, 219.2kHz 319.2kHz and 469.2kHz and 480.8kHz. The visible spurs of the offset-mismatch are at 200kHz and 400kHz. With the correction method the SNDR is 52.7dB and the SFDR is 63.7dB. This is an improvement in SNDR of 36.8dB and an improvement in SFDR of 44.7dB. The SFDR is limited by gain-mismatch and timing-mismatch.

Input frequency 494.44580kHz

In figure 7.3 the single sided spectrum is shown when the input frequency is approximately 494.4kHz, (a) shows it for a normal 4-channel TI-ADC configuration without any correction method and (b) shows it for the 4-channel TI-ADC with the designed correction method.

In (a) three spurs due to timing-mismatch are clearly visible (5.6kHz, 244.4kHz and 255.6kHz). A spur at 250kHz is visible due to offset-mismatch and a spur at 200kHz is visible because of crosstalk in the input multiplexers. Without correction the SNDR is 10.9dB and the SFDR is 13.9dB.

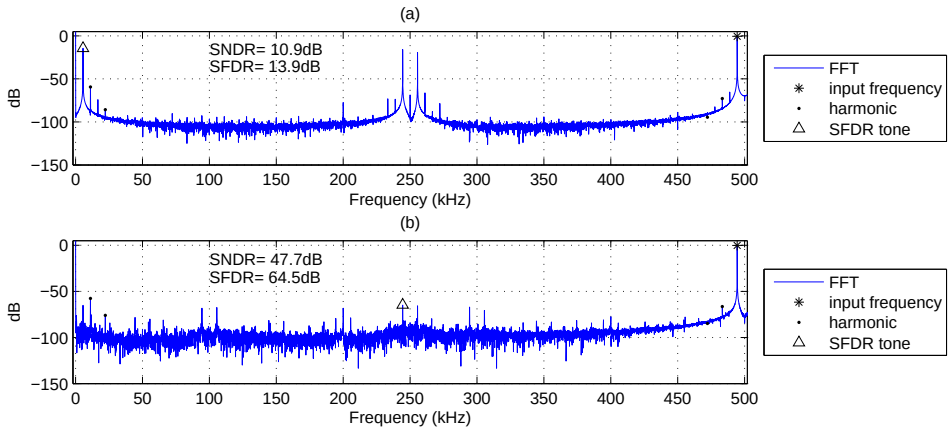


Figure 7.3: Spectrum of 4-channel 1MSps TI-ADC $f_0 \approx 494.4$kHz, $\Delta t_i = [128\ 0\ -128\ 0\ (0)]$ns; (a) no correction algorithm, (b) with correction algorithm.

In (b) the spurs are clearly spread and reduced by the algorithm as expected. The visible spurs of gain-mismatch and timing-mismatch are at 5.6kHz, 94.4kHz, 105.6kHz, 194.4kHz, 205.6kHz, 244.4kHz, 255.6kHz, 294.4kHz and 305.6kHz. The visible spurs of the offset-mismatch are at 200kHz and 400kHz. With the correction method the SNDR is 47.7dB and the SFDR is 64.5dB. This is an improvement in SNDR of 36.8dB and an improvement in SFDR of 50.6dB. The SFDR is limited by a spur due to gain-mismatch and timing-mismatch.

Summary

Without the algorithm the SFDR and SNDR decreas when the the input frequency increases, as expected. The algorithm improves the SNDR and SFDR for all measured frequencies. The SFDR reached for all three input frequencies is approximately constant at 64dB. Therefore, the SFDR is no longer limited by timing-mismatch but it is limited by gain-mismatch. The noise floor is increasing with the input frequency, this is probably caused by sampling jitter.

7.2 Intrinsic Timing Mismatch

In this section no errors are added on purpose, this means all DCDL are programmed with the same initial delay. Therefore, the timing-mismatch is only due to delays in the PCB and mismatches between components.

Input frequency 84.899902kHz

In figure 7.4 the single sided spectrum is shown when the input frequency is approximately 84.9kHz, (a) shows it for a normal 4-channel TI-ADC configuration without any correction method and (b) shows it for the 4-channel TI-ADC with the designed correction method.

In (a) the three spurs due to timing-mismatch are clearly visible (165.1kHz, 334.9kHz and 415.1kHz). The spurs at 250kHz and 500kHz are caused by offset-mismatch and a spur at 200kHz is caused by the reference sine wave through crosstalk in the input multiplexers. Without correction the SNDR is 59.8dB and the SFDR is 66.6dB.

In (b) the spurs are clearly spread and reduced by the algorithm as expected. The visible spurs of gain-mismatch and timing-mismatch are at 115.1kHz, 284.9kHz, 315.1kHz and 484.9kHz. The spurs of offset-mismatch are moved to 200kHz and 400kHz. With the correction method the SNDR is 59.3dB and the SFDR is 64.4dB. This is an decrease in SNDR of 0.5dB and an decrease in SFDR of 2.2dB. The spurs are clearly spread by the algorithm as expected, but the spurs have

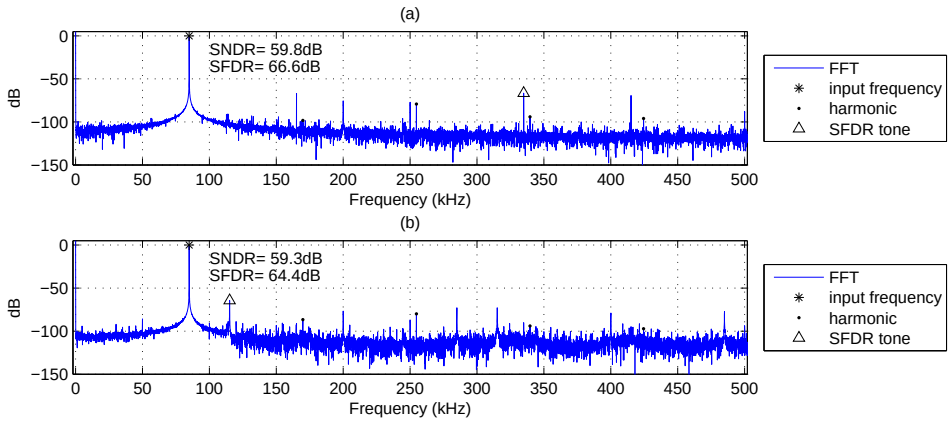


Figure 7.4: Spectrum of 4-channel 1MSps TI-ADC f_0≈84.9kHz, intrinsic timing-mismatch, (a) no correction algorithm, (b) with correction algorithm.

increased. The increase of the spurs are most likely caused by a mismatch between the replacement sub-ADC and the other sub-ADCs. The SFDR is limited by a spur due to gain-mismatch and timing-mismatch.

Input frequency 280.82275kHz

In figure 7.5 the single sided spectrum is shown when the input frequency is approximately 280.8kHz, (a) shows it for a normal 4-channel TI-ADC configuration without any correction method and (b) shows it for the 4-channel TI-ADC with the designed correction method.

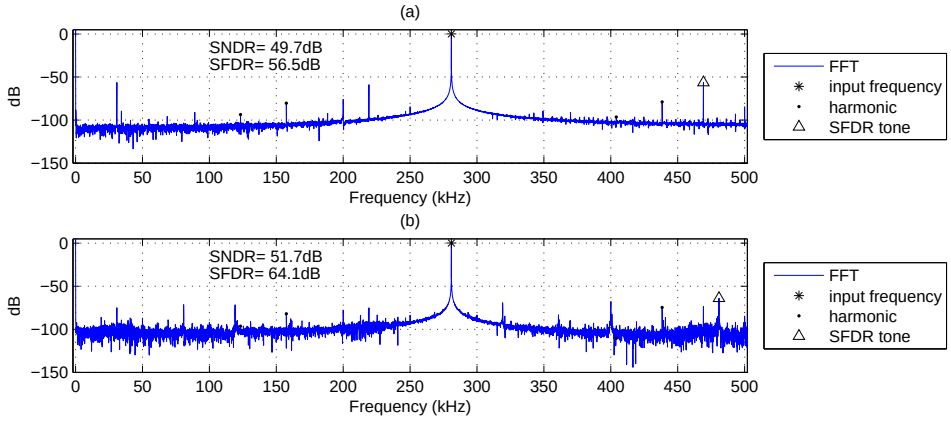


Figure 7.5: Spectrum of 4-channel 1MSps TI-ADC f_0≈280.8kHz, intrinsic timing-mismatch, (a) no correction algorithm, (b) with correction algorithm.

In (a) the spurs caused by timing-mismatch are clearly visible (30.8kHz, 219.2kHz and 469.2kHz). Offset-mismatch spurs are visible at 250kHz and 500kHz. The crosstalk of the input multiplexers is visible at 200kHz. Without correction the SNDR is 49.7dB and the SFDR is 56.5dB.

In (b) the spurs are clearly spread and reduced by the algorithm as expected. The visible spurs of gain-mismatch and timing-mismatch are at 30.8kHz, 80.8kHz, 119.2kHz, 219.2kHz 319.2kHz and 469.2kHz and 480.8kHz. The visible spurs of the offset-mismatch are at 200kHz and 400kHz. Without correction the SNDR is 49.7dB and the SFDR is 56.5dB, with the correction method the SNDR is 41.7dB and the SFDR is 64.1dB. This is an improvement in SNDR of 2.0dB and

an improvement in SFDR of 7.6dB. The SFDR is limited by a spur due to gain-mismatch and timing-mismatch.

Input frequency 494.44580kHz

In figure 7.6 the single sided spectrum is shown when the input frequency is approximately 494.4kHz, (a) shows it for a normal 4-channel TI-ADC configuration without any correction method and (b) shows it for the 4-channel TI-ADC with the designed correction method.

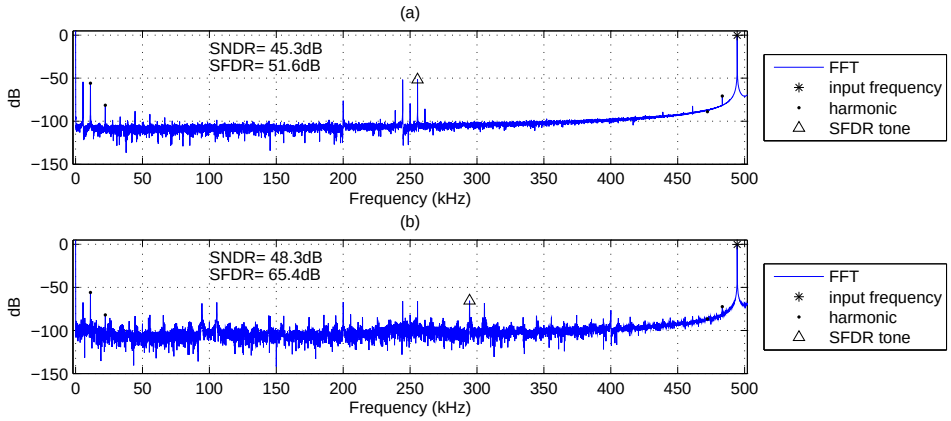


Figure 7.6: Spectrum of 4-channel 1MSps TI-ADC $f_0 \approx 494.4$kHz, intrinsic timing-mismatch, (a) no correction algorithm, (b) with correction algorithm.

In (a) three spurs due to timing-mismatch are clearly visible (5.6kHz, 244.4kHz and 255.6kHz). A spur at 250kHz is visible due to offset-mismatch and a spur at 200kHz is visible because of crosstalk in the input multiplexers. Without correction the SNDR is 45.3dB and the SFDR is 51.6dB.

In (b) the spurs are clearly spread and reduced by the algorithm as expected. The visible spurs of gain-mismatch and timing-mismatch are at 5.6kHz, 94.4kHz, 105.6kHz, 244.4kHz, 255.6kHz, 294.4kHz and 305.6kHz. The visible spurs of the offset-mismatch are at 200kHz and 400kHz. With the correction method the SNDR is 48.3dB and the SFDR is 65.4dB. This is an improvement in SNDR of 3.0dB and an improvement in SFDR of 13.8dB. The SFDR is limited by a spur due to gain-mismatch and timing-mismatch.

Summary

The SFDR reached for all three input frequencies is again limited to 64dB. The performance of the developed algorithm in terms of SFDR is therefore independent of the amount of timing-mismatch and independent of the input frequency but is limited by gain-mismatch.

7.3 Comparison with Non-Ideal Simulation Model

To verify the spurs in the measurement results, non-ideal simulations were done for each mismatch type.

Offset-Mismatch

Simulation was done with an offset-mismatch of -1mV (-0.02% of full scale) for channel 2 and 1mV for channel R. The other channels have no offset-mismatch. The offset in the demonstrator is controlled with a potentiometer; therefore, an offset-mismatch of $\pm$1mV is a reasonable error.

Gain-mismatch and timing-mismatch for all channels is zero. Figure 7.7 shows the simulation results for an input frequency of approximately 494.4kHz. The simulation shows that the offset-

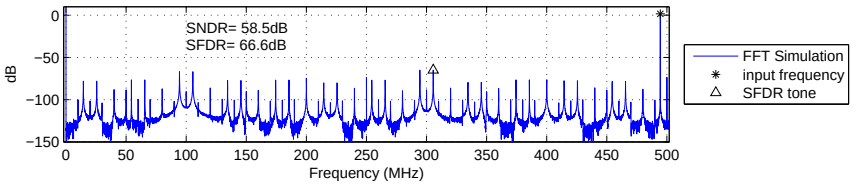


Figure 7.7: Spectrum of 4-channel 1MSps TI-ADC f_0≈494.4kHz with correction algorithm, offset-mismatch o_i=[0 −0.001 0 0 0.001]V

mismatch results in a timing-mismatch by the correction algorithm. This is because the correction algorithm estimates the timing-mismatch from the zero crossings of the calibration signal and the zero crossings depend on the offset.

Gain-Mismatch

Simulation was done with a gain-mismatch of -0.001 (-0.1%) for channel 2 and 0.001 for channel R. The other channels have no gain-mismatch. The gain in the demonstrator is controlled with a potentiometer; therefore, a gain-mismatch of ±0.1% is a reasonable error. Offset-mismatch and timing-mismatch for all channels is zero. Figure 7.8 shows the simulation results for an input frequency of approximately 494.4kHz. The gain spurs are at the same location as the spurs in

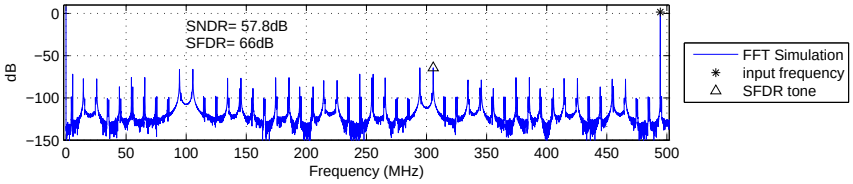


Figure 7.8: Spectrum of 4-channel 1MSps TI-ADC f_0≈494.4kHz with correction algorithm, gain-mismatch g_i=[1 0.999 1 1 1.001]

figure 7.6(b), the magnitude is different because the simulated gain-mismatch is not identical to the gain-mismatch in the demonstrator.

Timing-Mismatch

Simulation was done with a timing-mismatch of 1ns (step of DCDL) for channel 2 and 1ns for channel R. The other channels have no timing-mismatch. Offset-mismatch and gain-mismatch for all channels is zero. Figure 7.9 shows the simulation results for an input frequency of approximately 494.4kHz. The figure shows that the SFDR for this timing mismatch is already worse than

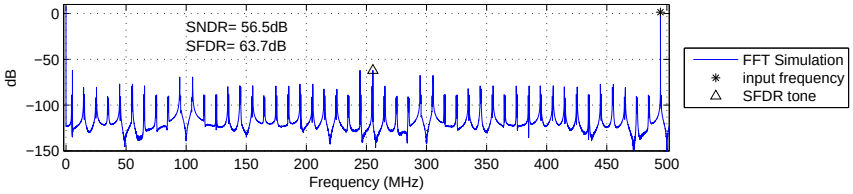


Figure 7.9: Spectrum of 4-channel 1MSps TI-ADC f_0≈494.4kHz with correction algorithm, timing-mismatch Δt_i=[0 1 0 0 1]ns

the SFDR of the demonstrator (see figure 7.6(b)). Therefore, the timing mismatch left in the demonstrator after correction is smaller than the resolution of the DCDLs.

Summary

The non-ideal simulations show that offset-mismatch generates a timing-mismatch. The timing mismatch left in the demonstrator is nevertheless smaller than the resolution of the DCDLs. The SFDR with a gain-mismatch of 0.1% in one channel and -0.1% in the replacement channel is already close to the SFDR of the demonstrator. To improve the results of the demonstrator a correction algorithm for offset-mismatch and gain-mismatch is required.

7.4 Comparison with Literature

Table 7.1 gives an overview of measurement results and the results found in literature. Comparison of the results is difficult because most results shown in literature are simulation results and the measurement results in literature are at a far higher rate and realized with an integrated ADC with on-chip correction instead of a PCB with off-the-shelve components. The developed method

Method	SFDR	SNDR	Channels	Bits	Type	Oversampling
Iroaga *et al.* [9]	37dB	27dB	3	10	Simulation	1
Iroaga *et al.* [9]	68dB	59dB	3	10	Simulation	2
Liu *et al.* [11]		75dB	2	14	Simulation	1
Harpe *et al.* [12]	62.5dB		2	10	1GSps on-chip	1.48(338.9MHz)
Mesadi [2]	56dB		4	10	Simulation	1
Developed	64.4dB	59.3dB	4	16	1MSps PCB	5.89(84.9kHz)
Developed	64.1dB	51.7dB	4	16	1MSps PCB	1.78(280.8kHz)
Developed	65.4dB	48.3dB	4	16	1MSps PCB	1.01(494.4kHz)

Table 7.1: Comparison of the measurement results with literature.

shows the best results in SFDR for input frequencies close to Nyquist. The SFDR of the method from Iroaga *et al.* shows better simulation results for an oversampling rate larger than 2X, but without oversampling the SFDR decreases fast. In terms of SNDR the method of Liu *et al.* gives better simulation results with a 2-channel TI-ADC but this is a foreground mode calibration.

Chapter 8

Conclusions and Recommendations

This chapter draws conclusions from the results found in chapter 7 and gives recommendations for improving the realization and the developed method.

8.1 Conclusions

This report describes a new method for timing correction in TI-ADCs. The developed method is a mixed-domain calibration in background mode. It uses one extra sub-ADC to replace a sub-ADC under test. The test is performed by sampling a known sine wave with a frequency below the Nyquist frequency of the TI-ADC. Based on the samples of the sine wave the timing-mismatch is calculated and the sample clock is corrected. The test is performed for each sub-ADC individually, including the extra sub-ADC, and runs continuously without interrupting the conversion.

The developed method was modeled in Simulink and the results were compared with an ideal model. The performance of the Simulink model was promising. Therefore, a demonstrator was realized in hardware to verify the simulation results in practice. The demonstrator consists of two parts: a PCB with analog off-the-shelf components and a FPGA board controlling the TI-ADC system and the correction calculations.

The timing-mismatch is significantly reduced within the resolution of the controllable delay. The performance of the demonstrator is limited to an SFDR of 64dB, because of gain-mismatch.

8.2 Recommendations

Logic Reduction

The amount of logic used by the algorithm can be reduced. The correction estimation uses separate logic for each channel while the correction estimation is done successively. Therefore, the same error estimation logic can be used for each channel.

Hardware Reduction

The fifth order Butterworth LC filter for the sine wave generation can be of lower order to reduce hardware. Another option is to generate a digital sine wave and filter this with a RC filter.

Performance Improvement

The developed method is designed concentrating on automatic correction of timing mismatch. After correction the SFDR performance of the method is limited by gain-mismatch. For an

optimal performance it is important to design an algorithm for correcting the gain-mismatch. The designed method is sensitive to offset-mismatches as well. Therefore, an algorithm for reducing the offset-mismatch should be added to improve the performance of the timing correction.

The analog input of each sub-ADC should be buffered after the input multiplexers, because it turned out that the switching of the input multiplexers influences the gain and offset of each sub-ADC.

IC implementation

The proposed method is described and simulated at a high level with ideal blocks. A transistor model with parasitics should be designed to be able to implement and verify the method in an IC eventually.

Acknowledgment

The author likes to thank Fitsum Mesadi for his preliminary research on timing error calibration in TI-ADCs. Georgi Radulov and Pieter Harpe thanks for reviewing the PCB design. The components and measurement equipment used during the master of science project were available thanks to Piet Klessens. Special thanks to Hans Hegt and Arthur van Roermund for their supervision during the project.

Bibliography

[1] W. Black and D. Hodges, "Time interleaved converter arrays," *Solid-State Circuits, IEEE Journal of*, vol. 15, no. 6, pp. 1022–1029, Dec 1980.

[2] F. B. Mesadi, "Timing Error Calibration in Time-Interleaved ADCs," Mixed-signal Microelectronics Group, Einhoven University of Technology, Internship report, Aug 2008.

[3] H. Jin and E. Lee, "A digital-background calibration technique for minimizing timing-error effects in time-interleaved ADCs," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 47, no. 7, pp. 603–613, Jul 2000.

[4] C. Vogel, "The impact of combined channel mismatch effects in time-interleaved ADCs," *Instrumentation and Measurement, IEEE Transactions on*, vol. 54, no. 1, pp. 415–427, Feb. 2005.

[5] N. Kurosawa, H. Kobayashi, K. Maruyama, H. Sugawara, and K. Kobayashi, "Explicit analysis of channel mismatch effects in time-interleaved ADC systems," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 48, no. 3, pp. 261–271, Mar 2001.

[6] K. Poulton, J. Corcoran, and T. Hornak, "A 1-GHz 6-bit ADC system," *Solid-State Circuits, IEEE Journal of*, vol. 22, no. 6, pp. 962–970, Dec 1987.

[7] J. Elbornsson, F. Gustafsson, and J.-E. Eklund, "Analysis of mismatch effects in a randomly interleaved A/D converter system," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, no. 3, pp. 465–476, March 2005.

[8] L. Wu and J. Black, W.C., "A low-jitter skew-calibrated multi-phase clock generator for time-interleaved applications," *Solid-State Circuits Conference, 2001. Digest of Technical Papers. ISSCC. 2001 IEEE International*, pp. 396–397, 470, 2001.

[9] E. Iroaga, B. Murmann, and L. Nathawad, "A background correction technique for timing errors in time-interleaved analog-to-digital converters," *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pp. 5557–5560 Vol. 6, May 2005.

[10] M. Seo, M. Rodwell, and U. Madhow, "A Low Computation Adaptive Blind Mismatch Correction for Time-Interleaved ADCs," *Circuits and Systems, 2006. MWSCAS '06. 49th IEEE International Midwest Symposium on*, vol. 1, pp. 292–296, Aug. 2006.

[11] Z. Liu, M. Furuta, and S. Kawahito, "Simultaneous Compensation of RC Mismatch and Clock Skew in Time-Interleaved S/H Circuits," *IEICE Trans Electron*, vol. E89-C, no. 6, pp. 710–716, 2006. [Online]. Available: http://ietele.oxfordjournals.org/cgi/content/abstract/E89-C/6/710

[12] P. Harpe, H. Hegt, and A. van Roermund, "Analog calibration of channel mismatches in time-interleaved ADCs," *Circuit Theory and Design, 2007. ECCTD 2007. 18th European Conference on*, pp. 236–239, Aug. 2007.

List of Figures

List of Tables

Appendices

Appendix A

MATLAB m-files

A.1 Ideal TI-ADC

```
% Model for 4-channel ideal TI-ADC with quantization.
figure(1)
Fs=1e9;
Ts=1/Fs;
NFFT=16384;
Runtime=(NFFT-1)*Ts;

t=0:Ts:Runtime;
x0=0:4*Ts:Runtime;
x1=Ts:4*Ts:Runtime;
x2=2*Ts:4*Ts:Runtime;
x3=3*Ts:4*Ts:Runtime;
y=zeros(1,length(t));

% Gain
g=0.99;

% fin=round(749*(Fs/NFFT)); % ~45.7153MHz
 fin=round(7490*(Fs/NFFT)); % ~457.71533MHz

y0=g*sin(2*pi*fin*(x0));
y1=g*sin(2*pi*fin*(x1));
y2=g*sin(2*pi*fin*(x2));
y3=g*sin(2*pi*fin*(x3));

% Multiplex output
for i=1:length(y0)
 y(i*4-3)=y0(i);
 y(i*4-2)=y1(i);
 y(i*4-1)=y2(i);
 y(i*4)=y3(i);
end

% Quantize signal
ydig=round(2^15*y); %16-bit signed
ydigscaled=ydig/2^15;
```

```
% Plot output signal
subplot(2,1,1)
plot(t*1e6,ydigscaled);
xlim([0 3/fin*1e6])
ylabel('Amplitude')
ylim([-1 1])
set(gca,'YTick',-1:0.5:1)
grid on
title('(a)');
xlabel('Time (\mus)')

% Calculate Spectrum
Y = fft(ydigscaled,NFFT)/NFFT;
f = (Fs/1e6)*linspace(0,1,NFFT);
Ydb = 20*log10(abs(Y(1:NFFT)));
% Plot single-sided amplitude spectrum.
subplot(2,1,2)
plot(f,Ydb,'b')
grid on
ylim([-150 0])
xlim([-10 510])
title('(b)');
xlabel('Frequency (MHz)')
ylabel('dB')
set(gca,'YTick',-150:25:0)
```

A.2 TI-ADC with Mismatches

```
% Model for 4-channel TI-ADC with Mismatches and quantization.
figure(2)
Fs=1e9;
Ts=1/Fs;
NFFT=16384;
Runtime=(NFFT-1)*Ts;

t=0:Ts:Runtime;
x0=0:4*Ts:Runtime;
x1=Ts:4*Ts:Runtime;
x2=2*Ts:4*Ts:Runtime;
x3=3*Ts:4*Ts:Runtime;
y=zeros(1,length(t));

% Gain
g=0.99;

% fin=round(749*(Fs/NFFT)); % ~45.7153MHz
 fin=round(7490*(Fs/NFFT)); % ~457.71533MHz

% Offset
 do=[-0.002 0.0033 -0.0021 -0.004];
% do=[0 0 0 0];
```

```
% Gain-mismatch
 dg=[0.004 0.989-0.99 1.009-0.99 0.006];
% dg=[0 0 0 0];

% Timing-mismatch
 dt=[-0.009e-9 -0.002e-9 -0.008e-9 0.004e-9];
% dt=[0 0 0 0];

y0=(g+dg(1))*sin(2*pi*fin*(x0+dt(1)))+do(1);
y1=(g+dg(2))*sin(2*pi*fin*(x1+dt(2)))+do(2);
y2=(g+dg(3))*sin(2*pi*fin*(x2+dt(3)))+do(3);
y3=(g+dg(4))*sin(2*pi*fin*(x3+dt(4)))+do(4);

% Multiplex output
for i=1:length(y0)
 y(i*4-3)=y0(i);
 y(i*4-2)=y1(i);
 y(i*4-1)=y2(i);
 y(i*4)=y3(i);
end

% Quantize signal
ydig=round(2^15*y); %16-bit signed
ydigscaled=ydig/2^15;

% Plot output signal
subplot(2,1,1)
plot(t*1e6,ydigscaled);
xlim([0 3/fin*1e6])
ylabel('Amplitude')
ylim([-1 1])
set(gca,'YTick',-1:0.5:1)
grid on
title('(a)');
xlabel('Time (\mus)')

% Calculate Spectrum
Y = fft(ydigscaled,NFFT)/NFFT;
f = (Fs/1e6)*linspace(0,1,NFFT);
Ydb = 20*log10(abs(Y(1:NFFT)));
% Plot single-sided amplitude spectrum.
subplot(2,1,2)
plot(f,Ydb,'b')
grid on
ylim([-150 0])
xlim([-10 510])
title('(b)');
xlabel('Frequency (MHz)')
ylabel('dB')
set(gca,'YTick',-150:25:0)
```

A.3 Embedded m-files

A.3.1 Signal Routers

Router 0

```
%#eml
% Routing table for Multiplexer 0
function y = mux0(u)
if mod(u,4)==0&&mod(u,5)~=0 %(u==4)||(u==8)||(u==12)||(u==16)
    y=1;
else
    y=0;
end
```

Router 1

```
%#eml
% Routing table for Multiplexer 1
function y = mux1(u)
if mod(u-1,4)==0&&mod(u,5)~=0 %(u==1)||(u==9)||(u==13)||(u==17)
    y=1;
else
    y=0;
end
```

Router 2

```
%#eml
% Routing table for Multiplexer 2
function y = mux2(u)
if mod(u-2,4)==0&&mod(u,5)~=0 %(u==2)||(u==6)||(u==14)||(u==18)
    y=1;
else
    y=0;
end
```

Router 3

```
%#eml
% Routing table for Multiplexer 3
function y = mux3(u)
if mod(u-3,4)==0&&mod(u,5)~=0 %(u==3)||(u==7)||(u==11)||(u==19)
   y=1;
   else    y=0;
end
```

Router R

```
%#eml
% Routing table for Multiplexer R
function y = muxR(u)
if mod(u,5)==0;%(u==0)||(u==5)||(u==10)||(u==15)
   y=1;
    else    y=0;
end
```

A.3.2 Output Multiplexer

```
%#eml
% Routing table for the output multiplexer
function y = outmux(u)
if mod(u,4)==0&&mod(u,20)~=0||u==0
    y=0;
elseif mod(u-1,4)==0&&mod(u-5,20)~=0||u==25
    y=1;
elseif mod(u-2,4)==0&&mod(u-10,20)~=0||u==50
    y=2;
elseif mod(u-3,4)==0&&mod(u-15,20)~=0||u==75
    y=3;
else y=4;
end
```

Appendix B

FPGA Software VHDL-files

B.1 Hold Reset

```
----------------------------------------------------------------------------------
-- Company: TU/e
-- Engineer: Ralf van Otten
-- Design Name:    Hold reset for at least 3 clock periods.
-- Module Name:    holdrst - Behavioral
-- Target Devices:  Spartan 3E
-- Description: Hold asyncreset for at least 3 clock periods.
-- Dependencies: async reset
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity holdrst is
    Port ( clk : in  STD_LOGIC;
           rstin : in  STD_LOGIC;
           rstout : out  STD_LOGIC);
end holdrst;
architecture Behavioral of holdrst is
begin
hldrst: process(clk,  rstin)
variable resetcount : integer:= 3;
begin
if rstin = '1' then
rstout <= '1';
resetcount := 3;
elsif (clk'event and clk = '1') then -- after reset hold reset for 3 clock cycles
if resetcount = 0 then
rstout <= '0';
else
resetcount := resetcount -1;
end if;
end if;
end process hldrst;
end Behavioral;
```

B.2 Sample Clocks

```
----------------------------------------------------------------------------------
-- Company: TU/e
-- Engineer: Ralf van Otten
-- Design Name:   Sample clocks normal ADCs
-- Module Name:    sclks - Behavioral
-- Target Devices: Spartan 3E
-- Description:  4 Phaseshifted 250kHz clks.
-- Dependencies: Synchronus reset, 2 MHz clock in. and 4 bit counter.
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity sclks is
    Port ( cnt : in  STD_LOGIC_VECTOR (3 downto 0);
           clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           sclk1 : out  STD_LOGIC;
           sclk2 : out  STD_LOGIC;
           sclk3 : out  STD_LOGIC;
           sclk4 : out  STD_LOGIC;
           rstcnt : out  STD_LOGIC);
end sclks;
architecture Behavioral of sclks is
attribute buffer_type : string;
attribute buffer_type of clk : signal is "BUFG";
begin
sclk: process(clk)
begin
if (clk'event and clk = '1') then
 if rst = '1' then
   sclk1 <= '1';
sclk2 <= '1';
sclk3 <= '0';
sclk4 <= '0';
rstcnt <= '1';
 else
-- Phase 0
if (cnt<=2) then
sclk1 <= '0';
else
sclk1 <= '1';
end if;
-- Phase 90
if (cnt >=2 and cnt<=4) then
sclk2 <= '0';
else
sclk2 <= '1';
end if;
-- Phase 180
if (cnt>=4 and cnt<=6) then
sclk3 <= '0';
```

```
else
sclk3 <= '1';
end if;
-- Phase 270
if (cnt >=6 or cnt<=0) then
sclk4 <= '0';
else
sclk4 <= '1';
end if;
-- Counter counts from 0 to 7. Wait for 6 because of synchronus reset
if cnt = 6 then
   rstcnt <= '1';
else
   rstcnt <= '0';
end if;
  end if;
end if;
end process sclk;
end Behavioral;
```

B.3 Sample Clock R, Sine Control and 200kHz

```
----------------------------------------------------------------------------------
-- Company: TU/e
-- Engineer: Ralf van Otten
-- Design Name:   Sample clock Replacement ADC and Reference Sine
-- Module Name:    sclks_ref - Behavioral
-- Target Devices: Spartan 3E
-- Description: 250 kHz clock for replacement ADC.
-- 250 kHz phase shifted square wave to be filtered to sinewave.
-- Dependencies: sync reset, 2MHz clock in, and 4 bit counter.
----------------------------------------------------------------------------------
--BUFFER_TYPE constraint used
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity sclks_ref is
    Port ( cnt : in  STD_LOGIC_VECTOR (3 downto 0);
           clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           sclkR : out  STD_LOGIC;
           sclkM : out  STD_LOGIC;
           clkSin : out  STD_LOGIC;
           rstcnt : out  STD_LOGIC;
  nLOCKED : out STD_LOGIC);
end sclks_ref;
architecture Behavioral of sclks_ref is
attribute buffer_type : string;
attribute buffer_type of clk : signal is "BUFG";
begin
clks : process(clk)
begin
```

```
if (clk'event and clk = '1') then
 if rst = '1' then
   sclkR <= '1';
sclkM <= '1';
clkSin <= '1';
rstcnt <= '1';
nLOCKED <= '1';
 else
-- Phase 0
if (cnt<=2) then
sclkR <= '0';
else
sclkR <= '1';
end if;
-- 200kHz
if (cnt<=4) then
sclkM <= '0';
else
sclkM <= '1';
end if;
-- Phase Sine
if (cnt >=1 and cnt<=5) then
clkSin <= '0';
else
clkSin <= '1';
end if;
-- Counter counts from 0 to 9, wait for 8 because of sync reset.
if cnt = 8 then
   rstcnt <= '1';
nLOCKED <= '0';
else
   rstcnt <= '0';
end if;
  end if;
end if;
end process clks;
end Behavioral;
```

B.4 Input Multiplexers Driver

```
----------------------------------------------------------------------------------
-- Company: TU/e
-- Engineer: Ralf van Otten
-- Design Name:  Input multiplexer driver.
-- Module Name:    inmudrv - Behavioral
-- Target Devices: Spartan 3E
-- Description: Drive inputmultiplexers in correct order.
-- Dependencies: sync reset, 250kHz clock in. 4 bit counter.
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity inmudrv is
    Port ( cnt : in  STD_LOGIC_VECTOR (7 downto 0);
           clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           rstcnt : out  STD_LOGIC;
           mux1 : out  STD_LOGIC;
           mux2 : out  STD_LOGIC;
           mux3 : out  STD_LOGIC;
           mux4 : out  STD_LOGIC;
           muxR : out  STD_LOGIC;
  nSYNC : out STD_LOGIC);
end inmudrv;
architecture Behavioral of inmudrv is
attribute buffer_type : string;
attribute buffer_type of clk : signal is "BUFG";
begin
inpmuxdrv : process(clk)
begin
if (clk'event and clk = '1') then
 if rst = '1' then
   mux1 <= '0';
mux2 <= '0';
mux3 <= '0';
mux4 <= '0';
muxR <= '1';
rstcnt <= '1';
nSYNC <= '1';
 else
   -- Multiplexer 4
if cnt=4 or cnt=8 or cnt=12 or cnt=16 then
mux4 <='1';
else
mux4 <='0';
end if;
   -- Multiplexer 1
   if cnt=1 or cnt=9 or cnt=13 or cnt=17 then
mux1 <='1';
else
mux1 <='0';
end if;
   -- Multiplexer 2
   if cnt=2 or cnt=6 or cnt=14 or cnt=18 then
mux2 <='1';
else
mux2 <='0';
end if;
   -- Multiplexer 3
   if cnt=3 or cnt=7 or cnt=11 or cnt=19 then
mux3 <='1';
else
mux3 <='0';
end if;
   -- Multiplexer Replacement
   if cnt=0 or cnt=5 or cnt=10 or cnt=15 then
```

```
muxR <='1';
nSYNC <= '0';
else
muxR <='0';
end if;
-- Counter counts from 0 to 19, wait for 18 because of sync reset.
if cnt=18 then
rstcnt <= '1';
else
rstcnt <= '0';
end if;
 end if;
end if;
end process inpmuxdrv;
end Behavioral;
```

B.5 16-bit 3-wire Serial-to-Parallel Data Input.

```
----------------------------------------------------------------------------------
-- Company: TU/e
-- Engineer: Ralf van Otten
-- Design Name:   Serial to 16 bit Parrallel converter.
-- Module Name:    serialdatain - Behavioral
-- Target Devices: Spartan 3E
-- Description:  Convert Serial data in to 16 bit parrallel data.
--Generate 10MHz serial clock.
-- Dependencies: sync reset, 20 MHz clock in, nbusy
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity serialdatain is
    Port ( clk_20MHz : in  STD_LOGIC;
  rst : in  STD_LOGIC;
           dclk_10MHz : out  STD_LOGIC;
           Din : in  STD_LOGIC;
           busy : in  STD_LOGIC;
           dvalid : out  STD_LOGIC;
           Data : out  STD_LOGIC_VECTOR (15 downto 0));

end serialdatain;
architecture Behavioral of serialdatain is
attribute buffer_type : string;
attribute buffer_type of clk_20MHz : signal is "BUFG";
   type state_type is (Finished, idle, txBit, CheckFinished);
   signal state : state_type;
signal busysync : STD_LOGIC;
signal Dataint : STD_LOGIC_VECTOR(15 downto 0);
   signal bitnr : unsigned(3 downto 0) := X"F";

begin
sync : process (clk_20MHz)
```

```
begin
if (clk_20MHz'event and clk_20MHz = '1') then
busysync <= busy;
end if;
end process sync;
serin : process (clk_20MHz)
begin
   if (clk_20MHz'event and clk_20MHz = '1') then
      if rst = '1' then
         Data <= (others => '0');
dclk_10MHz <= '0';
bitnr <= X"F";
Dataint <= (others => '0');
dvalid <= '1';
state <= Finished;
      else
  case state is
         -- Wait for new data
 when idle =>
            dclk_10MHz <= '0';
            if(busysync = '1') then
              state <= txBit;
              dvalid <= '0';
            else
              state <= idle;
            end if;
 -- clock high
          when txBit =>
            dclk_10MHz <= '1';
            state <= checkFinished;
          -- Check if finished and collect data
          when checkFinished =>
            Dataint <= Dataint(14 downto 0) & Din;
--Data(bitnr) <= Din;
            dclk_10MHz <= '0';
            if(bitnr = 0) then
              state <= Finished;
            else
              state <= txBit;
              bitnr <= bitnr-1;
            end if;
 -- Conversion Finished wait for data unvalid
          when Finished =>
   Data <= Dataint;
dvalid <= '1';
dclk_10MHz <= '0';
            if(busysync = '1') then
              state <= Finished;
            else
              state <= idle;
              bitnr <= X"F";
            end if;
          when others => null;
        end case;
```

```
      end if;
   end if;
end process serin;
end Behavioral;
```

B.6 Data Demultiplexer.

```
----------------------------------------------------------------------------------
-- Company: TU/e
-- Engineer: Ralf van Otten
-- Design Name:   Data demultiplexer.
-- Module Name:    datademux - Behavioral
-- Target Devices: Spartan 3E
-- Description:  Select if data is not from reference data.
-- Dependencies: sync reset, clk
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity datademux is
    Port ( clk : in  STD_LOGIC;
           mux : in  STD_LOGIC;
           dvalid : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           n_ref : out  STD_LOGIC);
end datademux;
architecture Behavioral of datademux is
attribute buffer_type : string;
attribute buffer_type of clk : signal is "BUFG";
   type state_type is (S0, S1, S2, S3);
   signal state : state_type;
begin
indemux : process (clk)
begin
   if (clk'event and clk = '1') then
      if rst = '1' then
n_ref <= '1';
state <= S0;
      else
  case state is
         -- Sine on ADC
 when S0 =>
            if(mux = '1') then
              state <= S1;
              n_ref <= '1';
            else
              state <= S0;
            end if;
 -- Next data is from Sine
          when S1 =>
            if(mux = '0') then
              state <= S2;
```

```
              n_ref <= '0';
            else
              state <= S1;
            end if;
          -- Wait for data valid
          when S2 =>
            if(dvalid = '1') then
              state <= S3;
            else
              state <= S2;
            end if;
 -- Wait for data invalid
          when S3 =>
            if(dvalid = '1') then
              state <= S3;
            else
              state <= S0;
              n_ref <= '1';
            end if;
          when others => null;
        end case;
      end if;
   end if;
end process indemux;
end Behavioral;
```

B.7 Data Router.

```
----------------------------------------------------------------------------------
-- Company: TU/e
-- Engineer: Ralf van Otten
-- Design Name:   Data router
-- Module Name:    data_router - Behavioral
-- Target Devices: Spartan 3E
-- Description:  Route data to correction algorithm or to output.
-- Dependencies: sync reset, clk
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity data_router is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           dvalid : in  STD_LOGIC;
           nref : in  STD_LOGIC;
           data : in  STD_LOGIC_VECTOR (15 downto 0);
           data_ref : out  STD_LOGIC_VECTOR (15 downto 0);
           data_ana : out  STD_LOGIC_VECTOR (15 downto 0);
           valid_ref : out  STD_LOGIC;
           valid_ana : out  STD_LOGIC);
end data_router;
architecture Behavioral of data_router is
```

```
attribute buffer_type : string;
attribute buffer_type of clk : signal is "BUFG";
   type state_type is (Finished, Idle, CheckData);
   signal state : state_type;
begin
route : process (clk)
begin
   if (clk'event and clk = '1') then
      if rst = '1' then
         data_ref <= (others => '0');
data_ana <= (others => '0');
valid_ref <= '0';
valid_ana <= '0';
state <= Finished;
      else
  case state is
          -- Wait for valid data
 when Idle =>
            valid_ref <= '0';
valid_ana <= '0';
            if(dvalid = '1') then
              state <= CheckData;
            else
              state <= Idle;
            end if;
 -- Check if data is from analog signal or Sine and route data
          when CheckData =>
            if(nref = '1') then
              state <= Finished;
  data_ana <= data;
  valid_ana <= '1';
            else
              state <= Finished;
  data_ref <= data;
  valid_ref <= '1';
            end if;
 -- Wait for data invalid
          when Finished =>
            if(dvalid = '1') then
              state <= Finished;
            else
              state <= Idle;
            end if;
          when others => null;
        end case;
      end if;
   end if;
end process route;
end Behavioral;
```

B.8 Correction.

```
-- Company: TU/e
-- Engineer: Ralf van Ottem
-- Design Name:  Correction algorithm
-- Module Name:    correct - Behavioral
-- Target Devices: Spartan 3E
-- Description:  Generate data for the delay lines.
-- Dependencies: sync reset, clk
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.Numeric_Std.ALL;
entity correction is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           dvalid_ref : in  STD_LOGIC;
           data_ref : in  STD_LOGIC_VECTOR (15 downto 0);
           data_delay : out  STD_LOGIC_VECTOR (7 downto 0);
           write_delay : out  STD_LOGIC);
end correction;
architecture Behavioral of correction is
attribute buffer_type : string;
attribute buffer_type of clk : signal is "BUFG";

constant C32768: signed(17 downto 0) := "001000000000000000";--X"8000"; --2.5V
constant Cmult: signed(15 downto 0) := "1111111001110000";
   type state_type is (Finished, idle, substr, multi, feedback, add128);
   signal state : state_type;
signal data_new : signed(7 downto 0);
signal data_old : signed(7 downto 0);
signal data_sub16 : signed(15 downto 0);
   signal mult: signed(31 downto 0);
   signal data_temp : unsigned(15 downto 0);
begin
correct : process (clk)
 variable data_sig : unsigned(17 downto 0);
 variable data_sub18 : signed(17 downto 0);
 variable data_new9 : signed(8 downto 0);
begin
   if (clk'event and clk = '1') then
      if rst = '1' then
         data_delay <= X"80";-- Initial output delay 128ns
data_new <= X"80";-- Initial delay 128ns
write_delay <= '0';
state <= finished;
      else
  case state is
          when idle =>
-- wait until new data is valid and store it.
write_delay <= '0';
if dvalid_ref='1' then
  state <= substr;
  data_old <= data_new;
```

```
  data_temp <= unsigned(data_ref);
else
  state <= idle;
end if;
          when substr =>
-- create signed data from unsigned data and shift around zero
data_sig(17 downto 16) := "00";
data_sig(15 downto 0) := data_temp;
data_sub18 := signed(data_sig)-C32768;-- -2.5V
data_sub16 <= data_sub18(15 downto 0);
state <= multi;
          when multi =>
-- Calculate delay from input data
mult <= (signed(data_sub16) * Cmult);-- inp*(-0.02441)
state <= feedback;
          when feedback =>
   -- Add measure delay to old delay.
data_new <= data_old + (mult(21 downto 14)); -- feedback
state <= add128;
          when add128 =>
-- Make data unsigned
data_new9(8) := data_new(7);
data_new9(7 downto 0) := data_new;
data_delay <= STD_LOGIC_VECTOR(data_new9(7 downto 0));
state <= Finished;
          when Finished =>
-- Write delay and wait for data to be unvalid.
write_delay <= '1';
if dvalid_ref = '1' then
  state <= Finished;
else
  state <= idle;
end if;
          when others => null;
        end case;
end if;
   end if;
end process correct;
end Behavioral;
```

B.9 8-bit Parallel-to-3-wire Serial Data Output.

```
----------------------------------------------------------------------------------
-- Company: TU/e
-- Engineer: Ralf van Otten
-- Design Name:   8 bit Parrallel to serial converter.
-- Module Name:    serialdataout - Behavioral
-- Target Devices: Spartan 3E
-- Description:  Convert 8 bit parrellel data to serial data
--  with 10 MHz serial clock and enable.
-- Dependencies: sync reset, 20 MHz clock in.
----------------------------------------------------------------------------------
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity serialdataout is
    Port ( clk_20MHz : in  STD_LOGIC;
           dclk_10MHz : out  STD_LOGIC;
           data : in  STD_LOGIC_VECTOR (7 downto 0);
           dout : out  STD_LOGIC;
           LE : out  STD_LOGIC;
           W : in  STD_LOGIC;
           rst : in  STD_LOGIC);
end serialdataout;
architecture Behavioral of serialdataout is
 attribute buffer_type : string;
 attribute buffer_type of clk_20MHz : signal is "BUFG";
    type state_type is (Finished, idle, txBit, CheckFinished);
    signal state : state_type;
 signal data_in : STD_LOGIC_VECTOR (7 downto 0);
    signal bitnr : unsigned(2 downto 0) := "111";
begin
serpar8 : process (clk_20MHz)
begin
   if (clk_20MHz'event and clk_20MHz = '1') then
      if rst = '1' then
         dout <= '0';
dclk_10MHz <= '1';
bitnr <= "111";
LE <= '0';
data_in <= (others => '0');
state <= Finished;
      else
  case state is
          -- Wait until parrallel data ready for conversion
          when idle =>
            dclk_10MHz <= '1';
            if(W = '1') then
              state <= txBit;
              LE <= '1';
  data_in <= data;
            else
              state <= idle;
            end if;
 -- Next bit out, clk low
          when txBit =>
            dclk_10MHz <= '0';
            state <= checkFinished;
            dout <= data_in(7);
 -- clk high check finished and select next bit
          when checkFinished =>
            dclk_10MHz <= '1';
            if(bitnr = 0) then
              state <= Finished;
            else
              state <= txBit;
```

```
              bitnr <= bitnr-1;
  data_in <= data_in(6 downto 0) & '0';
            end if;
 -- Conversion Finished wait for data to be invalid
          when Finished =>
   LE <= '0';
dclk_10MHz <= '1';
            if(W = '1') then
              state <= Finished;
            else
              state <= idle;
              bitnr <= "111";
            end if;
          when others => null;
        end case;
      end if;
   end if;
end process serpar8;
end Behavioral;
```

B.10 Output Data Multiplexer.

```
----------------------------------------------------------------------------------
-- Company: TU/e
-- Engineer: Ralf van Otten
-- Design Name:  Output 5 to 1 Multiplexer 16 bit.
-- Module Name:    outputmux - Behavioral
-- Target Devices: Spartan 3E
-- Description: Output 16 bit data in correct order.
-- Dependencies: Sync reset, 20MHz clock in.
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity outputmux is
    Port ( clk : in STD_LOGIC;
  rst : in STD_LOGIC;
  validana1 : in  STD_LOGIC;
            dataana1 : in  STD_LOGIC_VECTOR (15 downto 0);
            validana2 : in  STD_LOGIC;
            dataana2 : in  STD_LOGIC_VECTOR (15 downto 0);
            validana3 : in  STD_LOGIC;
            dataana3 : in  STD_LOGIC_VECTOR (15 downto 0);
            validana4 : in  STD_LOGIC;
            dataana4 : in  STD_LOGIC_VECTOR (15 downto 0);
            validanaR : in  STD_LOGIC;
            dataanaR : in  STD_LOGIC_VECTOR (15 downto 0);
  dvalid : out STD_LOGIC;
  dataout : out  STD_LOGIC_VECTOR (15 downto 0));
end outputmux;
architecture Behavioral of outputmux is
attribute buffer_type : string;
```

```
attribute buffer_type of clk : signal is "BUFG";
   type state_type is (ADC1, ADC2, ADC3, ADC4, ADCR, nextchannel, nextR);
   type ADC_type is (ADC1, ADC2, ADC3, ADC4, ADCR);
   signal state : state_type;
signal old_state : ADC_type;
begin
outputmultiplexer : process (clk)
begin
   if (clk'event and clk = '1') then
      if rst = '1' then
         dataout <= (others => '0');
dvalid <= '0';
state <= ADC1;
old_state <= ADCR;
      else
  case state is
          -- Choose next Channal to Output, certainly not replacement
 when nextchannel =>
if (old_state = ADC1 and validana2 = '1') then
  state <= ADC2;
  dataout <= dataana2;
     dvalid <= '0';
elsif (old_state = ADC2 and validana3 = '1') then
  state <= ADC3;
  dataout <= dataana3;
     dvalid <= '0';
elsif (old_state = ADC3 and validana4 = '1') then
  state <= ADC4;
  dataout <= dataana4;
     dvalid <= '0';
elsif (old_state = ADC4 and validana1 = '1') then
  state <= ADC1;
  dataout <= dataana1;
     dvalid <= '0';
else
  state <= nextchannel;
end if;
 -- Next channal probably Replacement
 when nextR =>
   if (old_state = ADC1 and validana2 = '1') then
  state <= ADC2;
  dataout <= dataana2;
     dvalid <= '0';
elsif (old_state = ADC2 and validana3 = '1') then
  state <= ADC3;
  dataout <= dataana3;
     dvalid <= '0';
elsif (old_state = ADC3 and validana4 = '1') then
  state <= ADC4;
  dataout <= dataana4;
     dvalid <= '0';
elsif (old_state = ADC4 and validana1 = '1') then
  state <= ADC1;
  dataout <= dataana1;
```

```
     dvalid <= '0';
elsif(validanaR = '1') then
  state <= ADCR;
  dataout <= dataanaR;
     dvalid <= '0';
else
  state <= nextR;
       dvalid <= '1';
end if;
 -- Channel 1 to Output
 when ADC1 =>
      dvalid <= '1';
            old_state <= ADC1;
if(validanaR = '1') then
              state <= nextchannel;
else
              state <= nextR;
end if;
 -- Channel 2 to Output
          when ADC2 =>
      dvalid <= '1';
            old_state <= ADC2;
if(validanaR = '1') then
              state <= nextchannel;
else
              state <= nextR;
end if;
 -- Channel 3 to Output
          when ADC3 =>
      dvalid <= '1';
            old_state <= ADC3;
if(validanaR = '1') then
              state <= nextchannel;
else
              state <= nextR;
end if;
 -- Channel 4 to Output
          when ADC4 =>
      dvalid <= '1';
            old_state <= ADC4;
if(validanaR = '1') then
              state <= nextchannel;
else
              state <= nextR;
end if;
 -- Channel R to Output and select next channel
          when ADCR =>
      dvalid <= '1';
            if(validana1 = '1' and old_state = ADC3) then
              state <= ADC1;
  old_state <= ADCR;
  dataout <= dataana1;
  dvalid <= '0';
            elsif (validana2 = '1' and old_state = ADC4) then
```

```
              state <= ADC2;
  old_state <= ADCR;
  dataout <= dataana2;
  dvalid <= '0';
            elsif (validana3 = '1' and old_state = ADC1) then
              state <= ADC3;
  old_state <= ADCR;
  dataout <= dataana3;
  dvalid <= '0';
            elsif (validana4 = '1' and old_state = ADC2) then
              state <= ADC4;
  old_state <= ADCR;
  dataout <= dataana4;
  dvalid <= '0';
else
  state <= ADCR;
            end if;
          when others => null;
        end case;
      end if;
   end if;
end process outputmultiplexer;
end Behavioral;
```

B.11 16-bit Parallel-to-3-wire Serial Data Output.

```
----------------------------------------------------------------------------------
-- Company: TU/e
-- Engineer: Ralf van Otten
-- Design Name:   16 bit Parrallel to serial converter.
-- Module Name:    serialdataout - Behavioral
-- Target Devices: Spartan 3E
-- Description:  Convert 16 bit parrellel data to serial data
--  with 25 MHz serial clock and enable.
-- Dependencies: sync reset, 50 MHz clock in.
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity serialdataout16b is
    Port ( clk_50MHz : in  STD_LOGIC;
           dclk_25MHz : out  STD_LOGIC;
           data : in  STD_LOGIC_VECTOR (15 downto 0);
           dout : out  STD_LOGIC;
           LE : out  STD_LOGIC;
           W : in  STD_LOGIC;
           rst : in  STD_LOGIC);
end serialdataout16b;
architecture Behavioral of serialdataout16b is
attribute buffer_type : string;
attribute buffer_type of clk_50MHz : signal is "BUFG";
   type state_type is (Finished, idle, txBit, CheckFinished);
```

```
   signal state : state_type;
signal data_in : STD_LOGIC_VECTOR (15 downto 0);
signal bitnr : unsigned(3 downto 0) := X"F";
begin
P2S16b : process (clk_50MHz)
begin
   if (clk_50MHz'event and clk_50MHz = '1') then
      if rst = '1' then
         dout <= '0';
dclk_25MHz <= '1';
bitnr <=  X"F";
LE <= '0';
data_in <= (others => '0');
state <= Finished;
      else
  case state is
          -- Wait until parrallel data ready for conversion
 when idle =>
            dclk_25MHz <= '1';
            if(W = '1') then
              state <= txBit;
              LE <= '1';
  data_in <= data;
            else
              state <= idle;
            end if;
 -- Next bit out, clk low
          when txBit =>
            dclk_25MHz <= '0';
            state <= checkFinished;
            dout <= data_in(15);
 -- clk high check finished and select next bit
          when checkFinished =>
            dclk_25MHz <= '1';
            if(bitnr = 0) then
              state <= Finished;
            else
              state <= txBit;
              bitnr <= bitnr-1;
  data_in <= data_in(14 downto 0) & "0";
            end if;
 -- Conversion Finished wait for data to be invalid
          when Finished =>
   LE <= '0';
dclk_25MHz <= '1';
            if(W = '1') then
              state <= Finished;
            else
              state <= idle;
              bitnr <=  X"F";
            end if;
          when others => null;
        end case;
      end if;
```

```
   end if;
end process P2S16b;
end Behavioral;
```