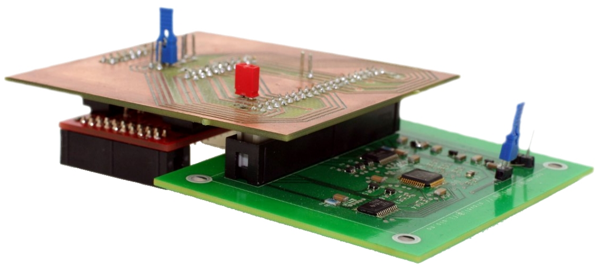
UNIVERSITY OF OSLO
Department of Informatics

Low power bitstream running cross-correlator / convolver

Master thesis

Olav E. Liseth



May 11, 2009

Abstract

Cross-correlation gives a measure of the similarity between two signals in the time domain and is desirable in pattern matching and recognition, but other solutions are often sought after due to the heavy computational load. The work includes processing of signals in their oversampled $\Delta\Sigma$ domain, bitstream processing, and asynchronous techniques. Employing these techniques resulted in a novel signal processing solution.

This master thesis presents a power efficient implementation of a time-domain bit-stream cross-correlator suitable for integration in CMOS. The developed generic SKILL script can generate layouts with different dimensions and one version is fabricated in 90 nm CMOS. Cross-correlation can be used for heartbeat detection in an ECG analysis and the implemented chip is tested with real-world Electrocardiography (ECG) beat detection. The power dissipated by the chip is compared to the power dissipated by a microcontroller performing equal computations. The bitstream cross-correlator has an estimated improvement in power dissipation by a factor of 84.

Contents

Acknowledgements

I want to thank my supervisor Tor Sverre Lande for always motivating me to perform and to challenge my curiosity in a wide variety of subjects. His guidance has given an excellent balance between direction and confidence in my independent work.

Great thanks to my co-supervisor Håkon A. Hjortland for providing technical skills and guidance in almost all the fields encountered throughout this work. Special credit for providing microcontroller firmware and for correcting the computational performance analysis of the implemented chip.

I owe great thanks to Daniel, Øyvind, Kristian, Bård, Dag and all the other guys that have been in and out the lab during the past years. Both social and technical discussions have improved the performed work. Thanks to Arvind for his picture-taking abilities.

Thank you Jill for always being there, for bringing me dinner during chip tape-out and for supporting me during rough times. Your excellent lingual skills have raised the standard of this thesis.

Last, but not least, I want to thank the entire Microelectronic Systems research group at the University of Oslo for providing a splendid academic environment.

Acronyms

ADC	Analog-to-Digital Converter
CIC	Cascaded Integrator-Comb
DAC	Digital-to-Analog Converter
DAQm	Data Acquisition module
DDC	Digital-to-Digital Converter
DI	Delay-Insensitive
DRC	Design Rule Check
DSM	Delta-Sigma Modulator
DSP	Digital Signal Processing
ECG	Electrocardiography
ENOB	Effective Number Of Bits
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FO4	Fanout-Of-4
FOM	Figure Of Merit
IIR	Infinite Impulse Response
lsb	least significant bit
MISO	Master Input, Slave Output
MIPS	Million Instructions Per Second
MOSI	Master Output, Slave Input
msb	most significant bit
MUX	Multiplexer

NTF	Noise Transfer Function
OSR	Oversampling Ratio
PCB	Printed Circuit Board
PCM	Pulse-Code Modulation
PDM	Pulse-Density Modulation
RMS	Root Mean Square
SACD	Super Audio CD
SPI	Serial Peripheral Interface
SMD	Surface Mounted Device
SNR	Signal-to-Noise Ratio
SQNR	Signal-to-Quantization-Noise Ratio
SS	Slave Select
TQFP	Thin Quad Flat Pack
USB	Universal Serial Bus
WSN	Wireless Sensor Networks

1 Introduction

1.1 Motivation

Cross-correlation gives a measure of the similarity between two signals in the time domain. Cross-correlation is desirable in pattern matching and recognition, but other solutions are often sought due to the heavy computational load.

A novel architecture for running cross-correlation is proposed in [Land 07]. It is possible to increase the efficiency compared to traditional cross-correlation by using simple logic on the bitstream produced by a delta-sigma, $\Delta\Sigma$ (or sigma-delta), modulator. The goal of this master thesis is to implement the suggested bitstream running cross-correlation method and to demonstrate a practical area of application. The efficiency of the bitstream cross-correlation is compared to other solutions in terms of power, area, accuracy and speed. The used data processing method is discussed and parallels are drawn to similar processing methods.

The third wave of computing, Ubiquitous computing, leads to that technology vanishes into the background of our lives [Weis 91]. Small, seamlessly integrated computers around and even within us means that charging and changing of batteries is a less favored operation. Wireless Sensor Networks (WSN) are a part of a ubiquitous future and have raised significant research activity lately. A WSN consists of many sensor nodes, each node being autonomous, typically equipped with one or more sensors, a radio transceiver, an energy source and a small microcontroller. The nodes are often small in size and use batteries. Each node has a limited communication range, but long range communication is possible by sending data through other nodes to complete the transmission to the base station. Energy is consumed by sensing, data processing and data communication, and often the most energy-expensive operation is data transmission. In WSNs, and most small, portable, battery operated devices, the scarcest resource is energy. Local data processing can reduce the amount of data to be transmitted with overall energy savings as the profit.

The vision of a ubiquitous interaction between human and machine requires sensing of the analog surroundings of a device and analog-to-digital converters are used for further digital processing of data. A popular and power efficient converter architecture is the $\Delta\Sigma$ converter, which uses oversampling to produce a one bit representation of the sampled signal. The value of the sampled signal is a local average of several bits from the bitstream produced. The oversampling rate is one factor to acquire the desired resolution in the converter at the cost of a higher clock rate. The bitstream is usually decimated to the Nyquist rate, resulting in a multibit signal. An example of efficient use of bitstream processing was proposed in [Malo 91], the methodology presented concerns the detection of input signals by the correlation technique. A commercial use of bitstreams is found in the Super Audio CD (SACD), developed by Sony and Philips Electronics. The SACD is an optical audio disc where the audio is stored as a bitstream [Jans 03].

Many methods for simplification of the cross-correlation operation are found in the literature. Examples are the weighted correlation of differences methods [Alpe 86] and average magnitude cross difference methods [Lind 88]. Both were considered to be more computationally efficient as they do not require the intense multiplicative processes associated with cross-correlation.

Convolution is the single most important technique in Digital Signal Processing (DSP) [Smit 97]. Its applications include statistics, image and signal processing, electrical engineering and differential equations. Convolution is the same mathematical operation as cross-correlation, the only difference being that one of the signals of interest is reversed.

A power efficient running bitstream cross-correlator/convolver benefits from the simple 1 bit representation of the signal. This representation of the signal is already present in many relevant applications, but rarely utilized. Cross-correlation and convolution are two of the most used processing tools in the digital signal processing toolbox and accurate and efficient computation methods are desired.

The old concept of clockless circuitry is again a subject of research. A system without a clock can be both faster and more energy efficient than its clocked counterpart. There are, however, many caveats and challenges in designing an *asynchronous* system. The clockless part of the implemented circuit is discussed and placed within asynchronous theory, theories expound when the asynchronous concept was evolved in the late 1950s.

1.2 Field of application

The processing method discussed in this thesis is feasible in a wide variety of applications. The article [Last 04] proposes a new heartbeat detection approach based on the fundamentals of cross-correlation. In the article, cross-correlation is used to identify different waveforms within a heartbeat. The thorough study uses ECG data from a large online database. This thesis will investigate how bitstream running cross-correlation performs compared to the ordinary cross-correlation scheme applied in the article.

2 Clockless computation

Synchronization is the coordination of events to operate a system in unison. There are at least two different solutions to synchronize a digital system, the most common approach being to perform all events in the system synchronously. This involves a system clock, a concept every computer customer has some relation to. There are many advantages to synchronous computation, which today is the only solution to synchronization for most electronics developers.

As a chain is only as strong as its weakest link, a digital system can only be as fast as its slowest component. Optimization and pipelining of a system can equalize the time spent in each computation block, but various factors prevents the next clock tick from striking at the most optimum time. Instead of operating the whole system at the same pace, asynchronous solutions are possible. The same optimization and pipelining are necessary, but each computation element indicates when outputs are valid and when inputs are read. The time spent on each computational step is adapted to the actual computation time.

Moore's law states that the number of transistors on integrated circuits are doubled every eighteen months. Shrinking manufacturing processes will typically reduce the power dissipation for a given circuit. Smaller transistors

1. allow a higher transistor count per IC

   and

2. result in lower capacitances which allows for higher clock frequencies

These two factors lead to the downside of Moore's law; the power consumption of computer nodes doubles every eighteen months [Feng 03].

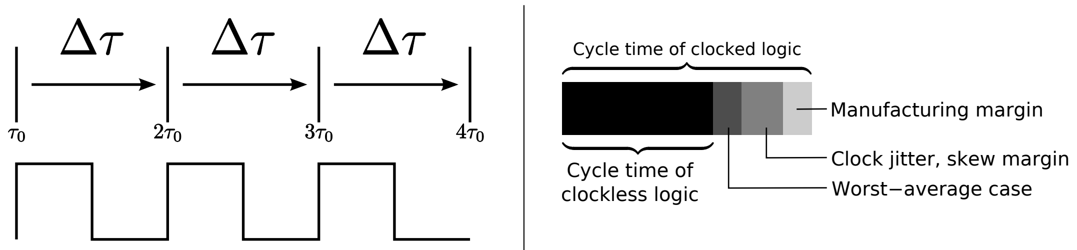
The growing need of power efficient circuits has inspired developers to revisit the old concept of the clockless chip. Most digital circuits designed today are synchronous, meaning that a system clock synchronizes computation. Research in clockless design goes back to the mid 1950s [Mull 59] and principles presented in the paper are still valid today. Clockless, or asynchronous, circuits do not require a system clock to compute or exchange data. Asynchronous circuit elements pass the result as soon as calculations are finished. This form of communication seems promising, but has several pitfalls. Many factors have to be considered when choosing between a synchronous, asynchronous or hybrid design. Lack of tools and expertise makes this a difficult choice.

The implemented circuit is described in chapter 5 and some techniques similar to asynchronous computing are used. To understand the differences and to see the similarities, a basic understanding of asynchronous techniques is important.

2.1 Asynchronous techniques, a brief overview

The major difference between asynchronous and synchronous design is the way data is exchanged between computational elements. Synchronous, or clocked, design requires a global signal which sets a time limit for every subcircuit's completion of a specific task. The clock defines a timing constraint which all circuit elements must follow. Fig. 2.1(a) illustrates the timing constraint of clocked logic, where $\Delta\tau$ is the time available, illustrated in Fig. 2.1(b). The time cycle consists of, in addition to the logic execution average time, the extra time it takes to run the worst case logic (worst – average case), variations in clock operations and manufacturing differences.

Asynchronous logic does not need the extra overhead clocked logic requires. Instead, handshaking is used. A circuit element indicates when its outputs are valid and when its inputs have been read. The time used for every computational element is kept to a minimum, regardless of process and environmental variations.



(a) Synchronous computation.

(b) $\Delta\tau$, synchronous time cycle [Geer 05].

Figure 2.1: Synchronous time cycle principle.

2.1.1 Handshake protocols

Various handshake protocols compatible in asynchronous circuits exist. Handshaking is a well-known term in synchronous logic, used for synchronization, to establish a connection between two devices, as well as other functions. Handshaking ensures that a register does not accept data from its predecessor before its successor has stored the data that the register was previously holding. The 4-phase protocol is illus-

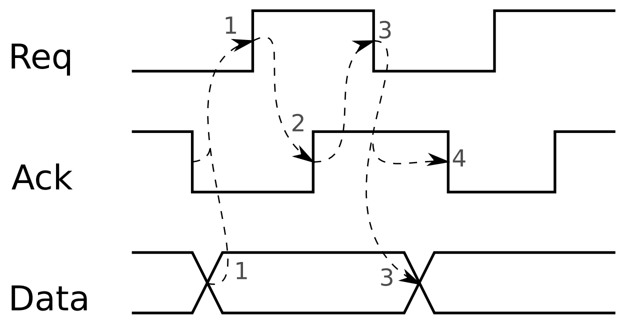


Figure 2.2: The 4-phase protocol.

trated in Fig. 2.2. There are 4 steps and two control lines involved in the handshaking process[Spar 01]:

1. The sender issues data and then sets request high.
2. The receiver latches data and then sets acknowledge high.
3. The sender responds by taking request low, data is no longer guaranteed to be valid.
4. The receiver acknowledges this by taking acknowledge low.

At this point, the sender may initiate the next communication cycle. This protocol is reliable regardless of the time delay in the wires between the two parties; the protocol is Delay-Insensitive (DI).

The 4-phase protocol has a disadvantage in the redundant return to zero transition. Instead, a transition on the request and acknowledge lines can ensure proper behavior. A transition from 0 to 1 has the same meaning as a transition from 1 to 0. This results in two less steps and the handshaking method is named the 2-phase protocol.

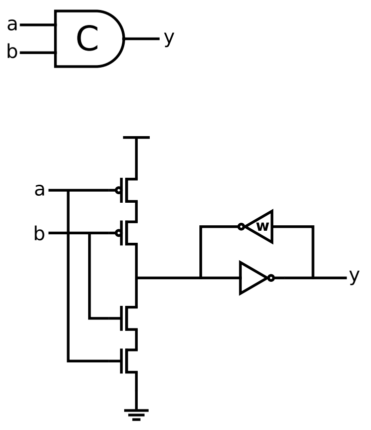
The clock is substituted by handshaking and each computational block indicates when data shall be passed on and received. Several restrictions apply and some considerations and solutions follow.

2.1.2 The Muller C-element

Implementation of handshaking logic requires that signals need to be valid at all times. One possible effect of a temporary transition on the request line, is duplication of data.

If the output of an AND gate changes from 0 to 1, it can be concluded that both inputs are 1. However, a conclusion about both inputs cannot be made for a transition on the output from 1 to 0. At least one input has changed, but there is no information about which one. Input signal transitions that are not indicated on the output are the source of hazards and should be avoided.

A fundamental component in asynchronous circuits is the Muller C-element, Fig. 2.3. For every transition on the output, a conclusion about the input can be made. When both inputs are 0, the output is set to 0, while the output is set to 1 when both inputs are 1.



a	b	y
0	0	0
0	1	no change
1	0	no change
1	1	1

Figure 2.3: The Muller C-element: Symbol, possible implementation and truth table. The letter w indicates that the transistors of the inverter are weak.

2.1.3 The Muller pipeline

The Muller pipeline relays handshakes and is, in one form or another, the control backbone of almost all asynchronous circuits [Spar 01]. Fig. 2.4 shows the Muller pipeline, which is built with inverters and Muller C-elements.

Considering the i^{th} C-element, C[i], the following pseudocode summarizes the behavior of the Muller pipeline:

```
If C[i-1] > C[i] and C[i-1] != C[i+1]:
  swap(C[i], C[i-1])
```

The i^{th} C-element will input and store a 1 only if its predecessor, C[i-1], is 1 and its successor, C[i+1] is 0. Or a more generic explanation, the value of the predecessor

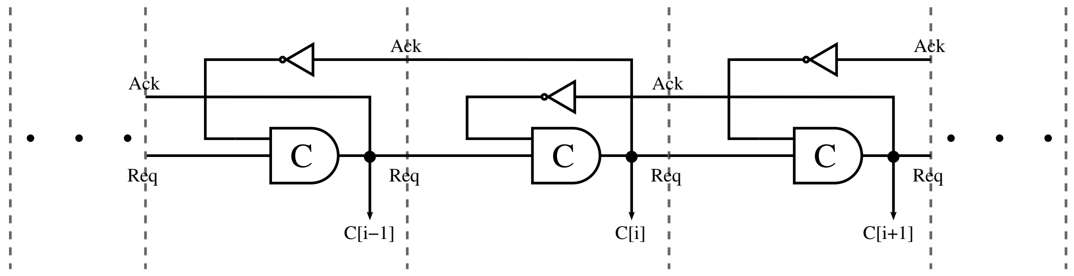


Figure 2.4: The Muller pipeline.

will propagate to the current element, if the value is the opposite of the value of the successor.

If a right hand environment does not respond to a handshake, the ripple of handshake will stop and the pipeline will fill. Following the above pseudocode, the Muller C-element outputs are $(\ldots, 0, 1, 0, 1, \ldots)$, every other element outputs a logic high.

The Muller pipeline works correctly regardless of delays in gates and wires; the Muller pipeline is Delay-Insensitive.

Circuit implementation style, 4-phase bundled-data

The 4-phase bundled-data protocol uses a Muller pipeline to generate local clock pulses. Fig. 2.5 shows how the Muller pipeline controls a pipeline with combinational logic. To

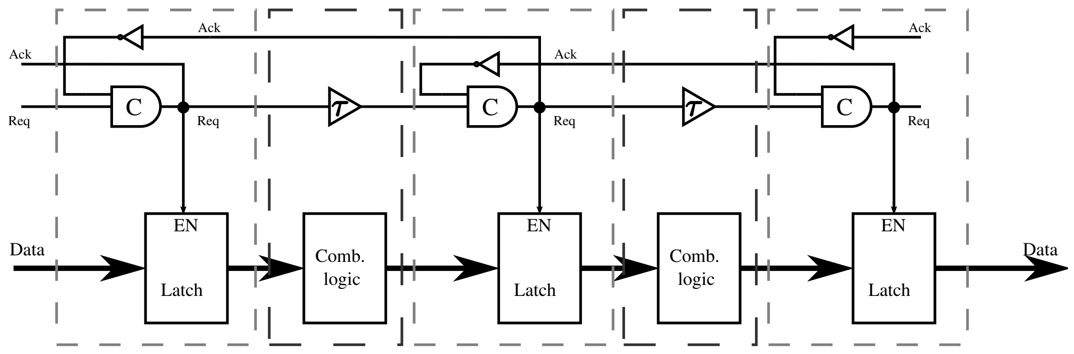


Figure 2.5: A 4-phase bundled data pipeline.

ensure completion of each combinational stage, a matching delay, τ in the figure, has to be inserted in the request signal path.

This pipeline implementation is particularly simple, but has some drawbacks:

- When it fills, only every other latch stores data.
- A handshake cycle involves communication with both neighbors, which reduces speed.

An implementation of the 4-phase bundled-data protocol gives a good understanding of asynchronous circuits, alternative implementations are faster and have a better occupancy when full.

2.2 Power dissipation

One possible advantage of asynchronous computation is less power consumption. The dynamic power dissipation of an asynchronous system can be considerable lower than in a clocked system. Equations in this section are mainly derived from [Kim 03].

2.2.1 Dynamic power dissipation

A transistor dissipates a small amount of power when it is switched on and off and even when it is off [West 05]. The first case is called dynamic power dissipation and is mainly caused by charging the load capacitance. Current typically flows from V_{dd} to charge the load and then from the load to GND during discharge. For a traditional inverter, the charging of the load capacitance to V_{dd} will take place when the input is low, and discharging the capacitance when the input is high. In addition, V_{dd} and GND will be short circuited for a short period while the input to the inverter is between V_{tn} and $V_{dd} - |V_{tp}|$, where V_{tn} and V_{tp} denote the threshold voltage for a NMOS and PMOS transistor respectively. This results in a current pulse from V_{dd} to GND and is caused by the switching time being larger than zero and both the pullup and pulldown transistors are partially turned on. The dynamic power has traditional been the dominant source of power consumption.

2.2.2 Static power consumption

Leakage current will occur when a transistor is turned off, known as static power dissipation. Ideally, no current flows through a turned off transistor. The two principal components of static power consumption are [Kim 03]:

- Subtreshold leakage, a weak inversion current across a transistor.
- Gate leakage, a tunneling current through the gate oxide insulation.

The components of static power dissipation were in older processes said to be zero. The current leakage increases when the threshold voltage and gate-oxide thickness decreases and is today an important design constraint.

The subtreshold leakage current is proportional to e^{-V_t}. An increase in the threshold voltage, because it appears as a negative exponent, can have a dramatic effect on the leakage current. Unfortunately, increasing V_t will reduce the operation frequency of a CMOS logic circuit, which is given by:

$$f \propto (V - V_t)^{\alpha} / V \tag{2.1}$$

where V is the transistor's supply voltage and the exponent α is an experimentally derived constant. A transistor with a high threshold voltage can not deliver enough current to maintain the maximum operation frequency. Using doping techniques or applying a bias voltage on the substrate of a transistor can increase the threshold voltage, which allows a designer to compromise between speed and power in different parts of their design.

A solution to reduce the gate leakage current is the use of high-κ dielectrics to better insulate the transistor's gate from the channel. This can bring gate leakage under control by 2010 [Kim 03].

Both main components in static power depend on the width of a transistor and obviously the number of transistors.

2.2.3 Overall power consumption

The static power consumption was predicted to exceed the dynamic power consumption as technology drops below 65 nm feature size [Kim 03]. The next equation defines overall power consumption as the sum of dynamic and static power:

$$P = ACV^2 f + V I_{leak} \tag{2.2}$$

In the first term, A is the fraction of gates actively switching, C is the total capacitance load of all gates and f is the operating frequency. The product of these factors constitutes the dynamic power consumption. The second term models the static power lost due to leakage current, I_{leak}. The equation ignores power loss due to the momentary short circuit when a gate switches, which is relatively small.

The V^2 factor in Eq. 2.2 suggests reducing supply voltage as the most effective method to decrease power consumption. The reduction of supply voltage leads to a lower operation frequency, which can be compensated with parallel or pipelined implementations. Transistor count and size are also important contributors to both static and dynamic power. Furthermore, a reduction of the number of them switching will drastically reduce the dynamic power.

2.2.4 Clock power dissipation

The vast majority of digital circuits use a system clock. The clock binds the entire chip to run at the speed of the slowest component, every logic subpart of the chip has to wait until the next tick from the clock. Clock driver circuits occupy a large amount of the limited chip area and consume a large percentage of the total power usage. In [Davi 97] it is said that the DEC Alpha CPU clock driver circuit occupies about 10% of the chip area and consumes over 40% of the power dissipated by the chip. The clock provides synchronization information, but does not principally contribute to the computed result. The clock distribution is power consuming even during idle periods and extensive effort is devoted to clock power saving circuits.

2.3 Synchronous vs. asynchronous

There are many valid arguments for the use of both synchronous and asynchronous solutions. Some of the reasons that clocked processors have dominated the industry since the 1960s [Geer 05] are that developers saw them as more reliable, capable of higher performance, and easier to design, test and run than their clockless counterparts.

2.3.1 Clockless advantages

There are many advantages associated with asynchronous logic, but implementing such a system is not necessarily favorable. In addition to faster computation, two distinct advantages are evident:

Power dissipation

The previously described handshake protocols use one line to request and one line to acknowledge for every bit exchanged. A traditional clock only uses one line to synchronize communication between computing blocks. Claiming that the clockless solution is power saving can seem faulty at first glance. The fact is that the many handshakes are most likely to use more dynamic power than a system clock solution. The bright side is that asynchronous systems more than offset this [Geer 05] because each subcircuit only uses power when it performs computation, while synchronous systems also use power in idle periods. Power is not only used to drive the clock circuits, power is also dissipated when the clock charges and discharges gates and lines in every subcircuit resulting in dynamic power dissipation. In asynchronous circuits, dynamic power dissipation is proportional to data processed.

Less electromagnetic interference (EMI)

In clocked designs, data moves at every tick of the clock, causing voltage spikes. This results in electromagnetic interference at the clock frequency and long clock lines aggravate the emitted EMI. In clockless designs, the current flow is spread out in time. The frequency and strength of the voltage spikes are reduced, minimizing the emitted EMI. Noise errors due to EMI are reduced on-chip and on nearby electronics. Low EMI is especially important in mixed mode design, where the analog part is sensitive to noise, e.g. an analog to digital converter.

2.3.2 Clockless challenges

The design of asynchronous logic is far from mature and there are many caveats bound to the technology. Many decades have been used to optimize clocked logic to the utmost. Extensive effort is essential to build up knowledge, building tools and experience on clockless logic. There is still a long way before research on clockless logic reaches the knowledge level of clocked logic. Some of the challenges seen today follow.

Integrating clocked with clockless logic

Few or none complete asynchronous systems are available on the market and therefore clocked and clockless logic need to interface. Clockless circuits request and acknowledge data at their own speed, while their clocked counterpart require valid data by each clock tick . Special circuits are needed to align the data.

Lack of tools and experience

Most of the integrated circuit software is developed to create clocked circuits. Designers often have to create their own tools or implement whole systems manually, which is time consuming and leads to high development costs. Some programming languages and synthesis tools exist.

The limited number of courses at colleges and universities concerning asynchronous design will not favor the field of clockless computation in the near future.

Full speed

Debugging asynchronous logic is complicated as it always runs at full speed. As there is no clock frequency to lower, one cannot slow down the system to detect when and where the circuit fails. This problem will come into view in later chapters.

Performance analysis

It is important that a new method of comparing performance is introduced. The performance of asynchronous logic is the outcome of environmental conditions and the data pattern to be processed and is therefore less defined than traditional clocked circuits. This is an important marketing consideration, where only the most effective circuitry will survive in this competitive market.

2.4 MiniMIPS — An asynchronous microprocessor

There are few or none commercially successful clockless chips, but fully implemented examples exist.

In 1997, the Caltech group completed the design of the MiniMIPS [Mart 97] — an asynchronous clone of a MIPS R3000 microprocessor. The goal was to implement the R3000 instruction set. The asynchronous counterpart should behave as its inheritor, but architectural similarity to existing MIPS microprocessors was not required. This permits maximum asynchronous optimization.

A speed and energy performance analysis was published in 2001. The performance of the MiniMIPS was measured in Million Instructions Per Second (MIPS), which should be used with skepticism as it is not comparable between CPU architectures. (Different interpretations of the acronym exist, "Meaningless Indicator of Processor Speed" among others.) The use of MIPS on circuits using the same instruction set and under the same conditions is considered reliable. The publication states that the asynchronous microprocessor is approximately four times as fast as a commercial synchronous MIPS R3000 in the same technology. Further, it is claimed that correction of a layout error would deliver five times the performance of the clocked version.

Energy efficiency is measured by the product $E \cdot t^2$, where E is the average energy of an instruction execution, and t is the average cycle time. The asynchronous design had an improvement in Et^2 by a factor of 90. The microprocessor was designed for high speed performance and special care to energy efficiency was not emphasized. The low power advantage of asynchronous design was assumed to give a power efficient result. A redesign of the MiniMIPS could deliver up to 400 times improvement in Et^2 compared to the implemented circuit. For details, see [Mart 01].

3 Bitstream representation

A $\Delta\Sigma$ data converter is a popular data conversion technique which we will exploit. $\Delta\Sigma$ conversion utilizes oversampling to move the in-band noise to higher frequencies and consists of a $\Delta\Sigma$ modulator, which produces a bitstream, and a low pass filter or decimator, which removes the out-band noise and downsamples the signal to the Nyquist sampling rate. The $\Delta\Sigma$ kind of bitstreams is also known as Pulse-Density Modulation (PDM). Pulse-Code Modulation (PCM) is the conventional digital modulation of an analog signal, usually presented as a binary code. The output of a $\Delta\Sigma$ converter is PCM coded, the bitstream is rarely used and is decimated to PCM code before any processing is done on the signal.

The bitstream cross-correlation advantage is significant with low oversampling ratios [Land 07]. A $\Delta\Sigma$ modulator with high precision and a low oversampling ratio can be challenging to make. A digital-to-digital six bit to one bit second-order modulator with an Oversampling Ratio (OSR) of 8 and a normalized input range of $-1 \leq x \leq 1$ is used to produce the bitstream in software. This resembles the modulator implemented as a side project on the processed chip. The $\Delta\Sigma$ conversion technique is explained as a Digital-to-Digital Converter (DDC), but the principles are similar to the Analog-to-Digital Converter (ADC) and Digital-to-Analog Converter (DAC).

3.1 Delta-Sigma data converters

$\Delta\Sigma$ data converters are used in an increasing range of modern electronic components and the most widespread use is in data conversion. A $\Delta\Sigma$ converter can achieve very high resolutions while using low-cost CMOS processes. Although the $\Delta\Sigma$ concepts have existed since the early 1960s [Inos 62], it is only in the recent years the technique has become common and popular because of the improvements in silicon technology.

The field of $\Delta\Sigma$ converters are large and a brief overview is given to get an understanding of the bitstream data representation. Two excellent books on the topic are [Nors 97, Schr 05].

3.1.1 Quantization

Quantization is the heart of all digital modulators and is the process of approximating a continuous range of values or, in our case, a set of discrete values by a smaller set of values. A $\Delta\Sigma$ modulator usually employ two-level quantization and a example is

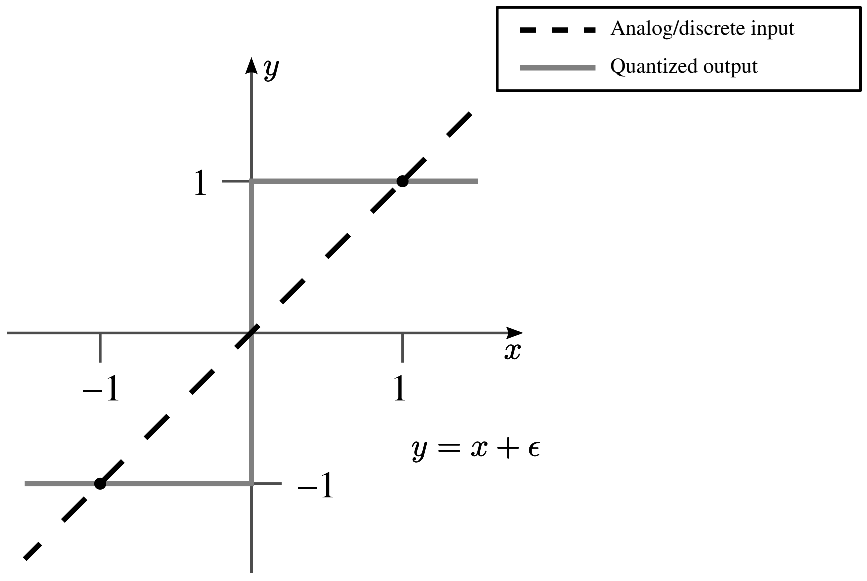


Figure 3.1: Two-level quantization.

plotted in Fig. 3.1. The input range is normalized to $-1 \leq x \leq 1$ and the level spacing, Δ, is 2. The two-level quantizer simply outputs 1 if its input is higher than a threshold at half of the input range and -1 if the input is lower. The quantized signal y is a linear function of the input x with an error ϵ; That is,

$$y = x + \epsilon \tag{3.1}$$

When x is in the valid input range, the error is bounded by $\pm\Delta/2$.

3.1.2 Delta-Sigma modulation

The $\Delta\Sigma$ modulator is the core of $\Delta\Sigma$ converters and produces a bitstream. A block diagram of a first-order digital to digital $\Delta\Sigma$ modulator is shown in Fig. 3.2. The input to the circuit, x_i, feeds to the quantizer via an integrator. The quantized output feeds back to subtract from the input. This negative feedback loop forces the average of the output to track the average input. A difference between them accumulates in the

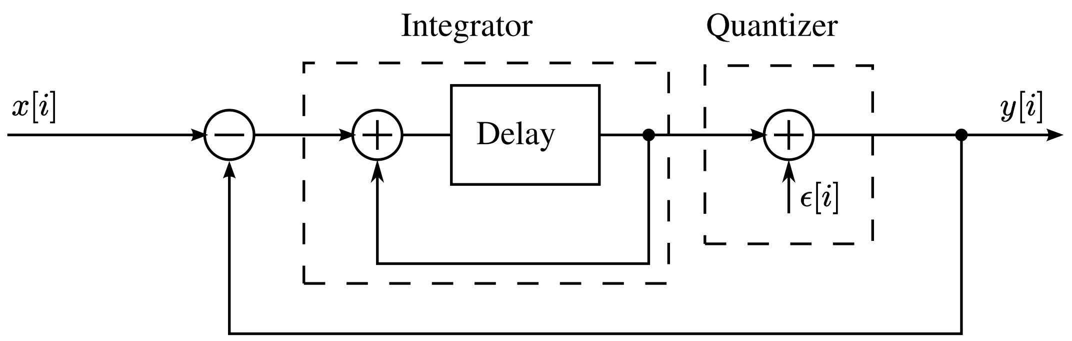


Figure 3.2: A block diagram of a first-order ΔΣ modulator.

integrator and eventually corrects itself. A analysis of Fig. 3.2 gives the output of the ΔΣ modulator,

$$y[i] = x[i-1] + (\epsilon[i] - \epsilon[i-1]) \tag{3.2}$$

The output contains a delayed and otherwise unchanged replica of the input signal and a differentiated version of the quantization error. A key aspect for the understanding of ΔΣ modulators is noise shaping: The differentiation of the quantization error ϵ, suppresses it at frequencies that are low compared to the sampling rate, f_s. The in-band noise is attenuated and moved to higher frequencies, the process called noise shaping. The output noise due to the quantization error is $q[i] = \epsilon[i] - \epsilon[i-1]$, as Eq. 3.2 shows. A Z-transformation of the output noise gives the discrete frequency domain representation of the quantization error:

$$Q(z) = E(z)\left(1 - z^{-1}\right) \tag{3.3}$$

where $E(z)$ is the frequency-domain quantization error of the ΔΣ modulator. The factor $1 - z^{-1}$ represents a high-pass filter; high frequency components of the quantization error passes, while low frequencies are attenuated.

In the frequency domain, the power spectral density (PSD) of the quantization error is found by replacing z by $e^{j2\pi f/f_s}$ in Eq. 3.3, where f_s is the sampling frequency.

$$S_q(f) = (2\sin(\pi f/f_s))^2 S_e(f) \tag{3.4}$$

$S_e(f)$ is the one sided PSD of the quantization error. By setting some conditions on the input signal, the error ϵ is approximated with white noise and is uncorrelated with the signal itself, thus $S_e(f)$ is constant. The Noise Transfer Function (NTF) of a first order ΔΣ modulator as a function of frequency is then proportional to Eq. 3.4. Fig. 3.3 shows the NTF for a first and second order ΔΣ modulator. A second order modulator suppresses the quantization noise within the signal band, i.e. at low frequencies, more

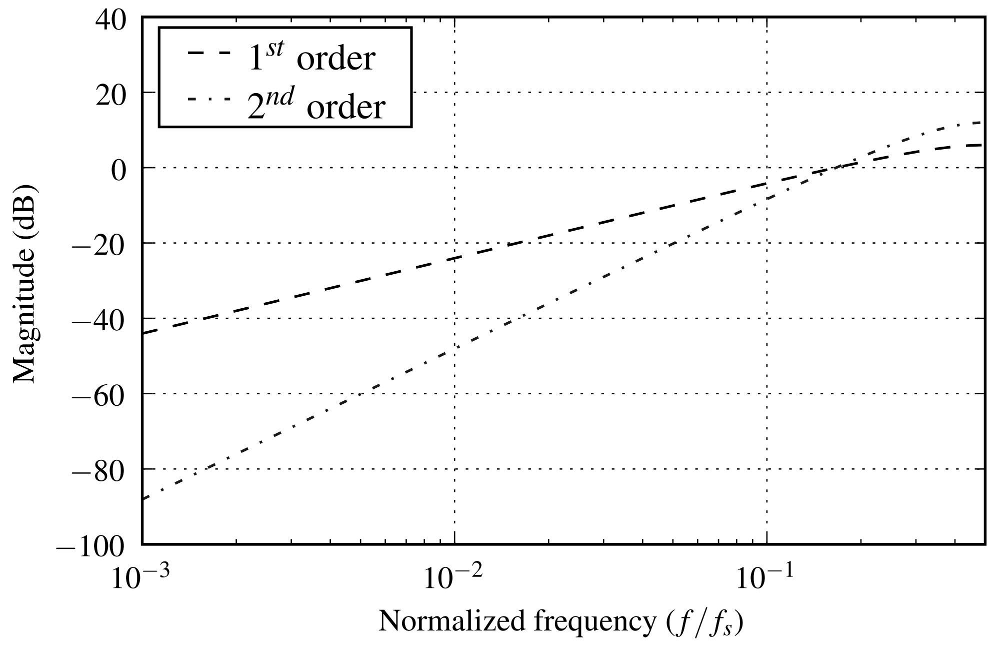


Figure 3.3: Noise transfer functions of a first and second order ΔΣ modulator.

efficiently. The drawback is that the second order modulator provides more gain at higher frequencies, resulting in higher total noise power.

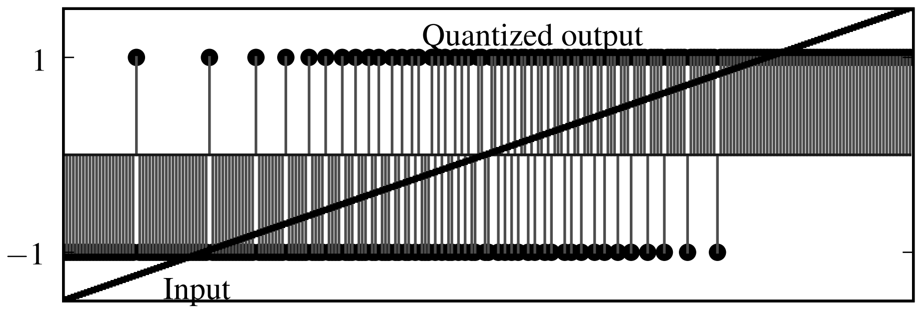
A ΔΣ modulator outputs a one bit representation of the sampled signal. The bitstream is equivalent to the output of a multibit ADC/DDC, differencing in how the signal is represented. A 1 bit representation is possible because the ΔΣ modulator oversamples the signal to a higher extent than a multibit converter. The OSR is defined by the ratio of the sampling frequency f_s to the Nyquist frequency $2f_0$,

$$OSR = \frac{f_s}{2f_0} \tag{3.5}$$

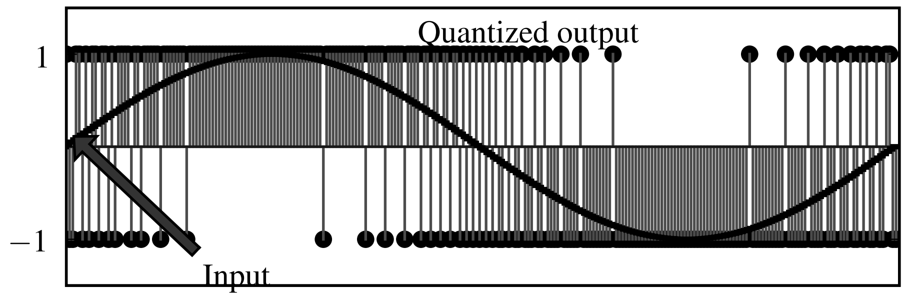
The bitstream is somewhat a serial representation of the signal of the cost of a higher clock rate, hence the one bit label is a bit of a misnomer. The higher bit rate allows a local average of the bitstream to represent the corresponding original sample.

The response of a ΔΣ modulator to a ramp input and a sinusoid is plotted in Fig. 3.4, the local average of the bitstream approximates the input value.

The implemented converter includes a second-order ΔΣ modulator. The first-order modulator described has the advantage of simplicity, robustness and stability, but the



(a) Ramp input.



(b) Sinusoidal input.

Figure 3.4: Response of a second-order ΔΣ modulator.

acquired resolution is inadequate for most applications. A second-order ΔΣ modulator can be created by replacing the quantizer in Fig. 3.2 with another copy of the first-order modulator. This results in lower in-band noise and higher resolution. As in all circuits including a feedback, instability in higher order ΔΣ modulators is a problem. Modulators with a higher order than two can not simply be made by adding further stages.

The in-band Signal-to-Quantization-Noise Ratio (SQNR) limits the resolution of a ΔΣ modulator. Assuming that the input signal to the previously described second-order ΔΣ modulator is a sine wave of amplitude 1, the SQNR depends on the OSR [Schr 05]:

$$SQNR = \frac{15(OSR)^5}{2\pi^4} \tag{3.6}$$

Assuming a uniform distribution of input signals, the relationship between the Effective Number Of Bits (ENOB) of a ΔΣ modulator and the OSR is given by

$$SNR(\text{dB}) = 20\log(2^{ENOB}) \approx 6.02ENOB \tag{3.7}$$

It is now possible to calculate the achievable ENOB of a second-order ΔΣ modulator with an OSR of 8. Eq. 3.6 gives $SQNR \approx 34$ dB, which gives $ENOB \approx 5.7$, nearly 6-bit resolution.

3.1.3 Decimation

The output of the ΔΣ modulator is a bitstream including the input signal and different noise components. The bitstream is the encoding representation used in the cross-correlator. At some point, the out-of-band noise has to be attenuated to reveal the result. The purpose of the decimation filter is to remove noise outside the signal band and to downsample the signal to the Nyquist rate. A simple decimation filter is the accumulate and dump circuit. If its input samples are x_i occurring at the sampling rate f_s and output samples are y_k occurring at the Nyquist rate $2f_0$, then

$$y_k = \frac{1}{N}\sum_{i=N(k-1)}^{Nk-1} x_i \tag{3.8}$$

where the decimation ratio N is the oversampling ratio OSR. OSR bits are summed and averaged, decreasing the data rate, while increasing the bit size of each sample. This filter has a frequency response based on a sinc function.

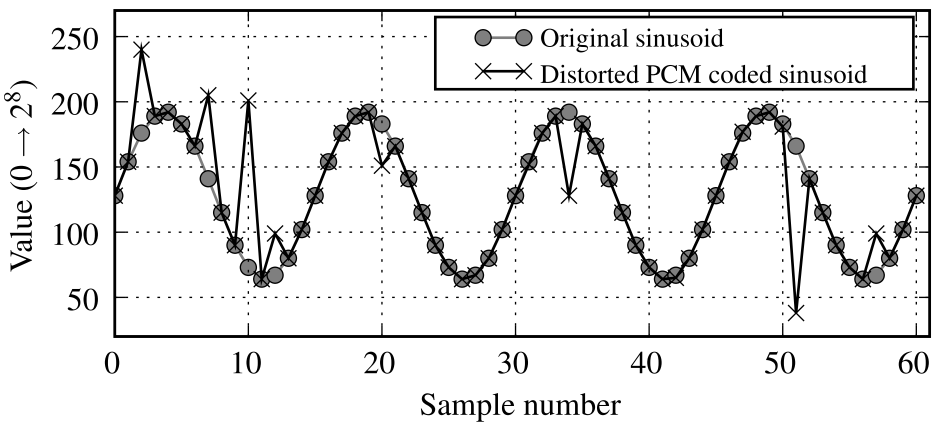
It has been shown that a close to optimum decimation filter function is a filter represented by $(L+1)$ products of sinc functions [Nors 97, p. 30], where L is the order of the ΔΣ modulator. Each filter in the decimator outputs a signal with a intermediate lower oversampling ratio than the previous, eventually resulting in the chosen data rate.

3.2 Bitstreams

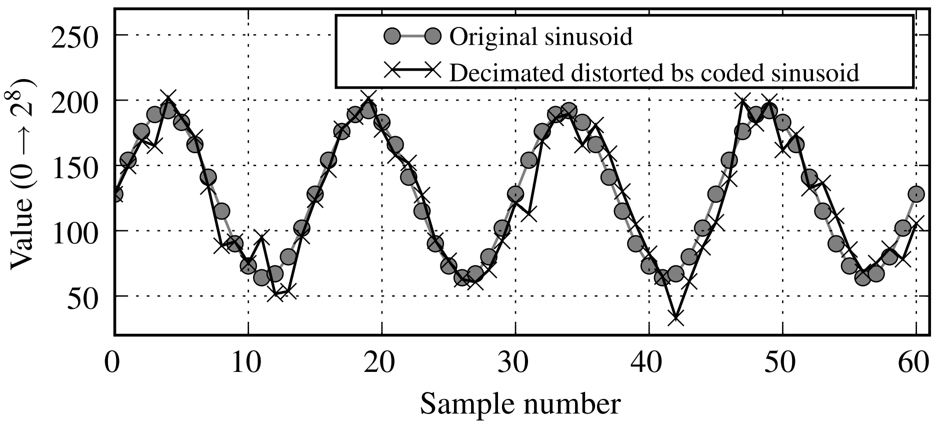
A simple way to get a understanding of how a bitstream represents a number, is to decimate a sequence of samples. For simplicity, let $N = OSR = 4$ and the bitstream excerpt $x = [0,1,0,0,1,0,0,1]$ includes very little noise. The outputs from the decimator, y_1 and y_2, can be found by using Eq. 3.8. y_1 and y_2 are as close to the input to the modulator as possible. Several other bitstreams would have produced the same output as x. A real world example is much more complex, including different noise components and higher order modulators and decimators. A detailed understanding of bitstreams is not intuitive and the bitstream is most often decimated to PCM code without any processing. The bitstream is just a step in the data conversion.

3.2.1 Robustness

The possibility of faults in digital circuits increases as transistor lengths decrease into the nanometric scale. The bit sequence in most coding schemes are ordered. Bits in a PCM coded value can e.g. be sorted from most significant bit (msb) to least signif-



(a) A PCM coded sinusoid with and without 20% of its bits inverted.



(b) A decimated bitstream coded sinusoid with and without 20% of its bits inverted.

Figure 3.5: Inverting of 20% of the bits randomly chosen in a PCM coded signal (a) and in a decimated bitstream coded signal (b).

icant bit (lsb), where the msb has more significance than the next. Inverting of the msb will cause a large error in the signal. A bitstream does not have bits more or less significant than the next, they all have the same weight. This makes a $\Delta\Sigma$ bitstream more error prone than many other coding forms. The advantage is not evident at low OSRs. Fig. 3.5 shows the inverting of 20% of the bits in a sinusoid with a resolution of eight bits. The inverted bits are randomly chosen, Fig. (a) shows how large impact an inversion of the msbs in a PCM coded signal has. The decimated bitstream plotted in Fig. (b) is modulated with $OSR = 8$, before the same bits are inverted and the bitstream is decimated back to PCM code. This signal is also clearly distorted, but has a better Signal-to-Noise Ratio (SNR) than the first example. The bitstream error tolerance is increasing with a higher OSR, which makes the coding form interesting in error vulnerable processes or harsh environments.

3.3 Bitstream operations

Mathematical operations on a bitstream are not intuitive like operations on PCM coded signals. Increasing a PCM coded value affects the less significant data bits, but a bitstream does not have lsbs. The bitstream operations needed are composed in the following sections.

3.3.1 Cross-correlation

The cross-correlation of two real discrete functions, $f[i]$ and $g[i]$, is defined as

$$y[i] = f[i] \star g[i] = \sum_{k=-\infty}^{\infty} f[k]g[i+k] \tag{3.9}$$

where the sum is over the appropriate values of k.

This means that for each shift in time, all the n samples defined by k have to be multiplied and summed together. We would typically need the calculation in real-time and the multiplication has to be done n times the sampling frequency or in parallel. Multiplication is a complex computation. The traditional cross-correlation principle in hardware is shown in Fig. 3.6.

It is possible to calculate the running cross-correlation in a more efficient way by encoding the signal as a bitstream.

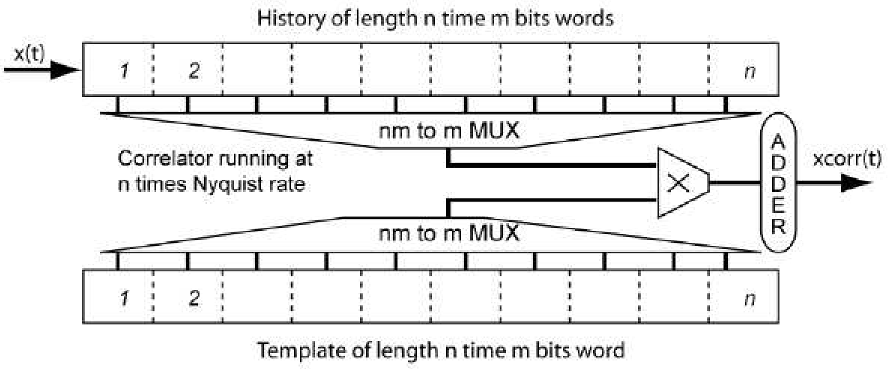


Figure 3.6: Traditional cross-correlation principle. *Picture taken from [Land 07]*

Bitstream multiplication

Multiplication between bitstreams has a significant advantage compared to its multibit counterpart and can be carried out using a basic logic gate. The rules of binary multiplication are the same as the truths of the AND gate. From Eq. 3.9 it is reasonable to assume that we should AND the incoming bitstream with the template. Several logic operations on $\Delta\Sigma$ bitstreams were derived in [Malo 91] and the AND gate was indicated to be the bitstream equivalent to PCM multiplication. The calculations were performed under the prerequisite of a modulator with input dynamic range from 0–1. The correct gate in our case can be found by viewing the sigma-delta modulation in a probabilistic term. The dynamic range of the implemented modulator is normalized to $-1 \leq x \leq 1$, where x is the input. The probability that the output of the modulator, X, is 1 or 0 is then given by $P(X = 1) = (1 + x)/2$ or $P(X = 0) = 1 - P(X = 1) = (1 - x)/2$, respectively. The modulation of two input signals x and y is regarded as uncorrelated and results in two bitstreams, X and Y. The XNOR of the two bitstreams results in:

$$\begin{aligned} P(X\overline{\oplus}Y = 1) &= P(X = 0) \cdot P(Y = 0) + P(X = 1) \cdot P(Y = 1) \\ &= \frac{1-x}{2} \cdot \frac{1-y}{2} + \frac{1+x}{2} \cdot \frac{1+y}{2} \\ &= \frac{1}{2}(1 + xy) \end{aligned}$$

which indicates that operations which usually require complex digital circuitry, can be done with a single logic gate when processing bitstreams.

Bitstream summing

Bubble sort is a simple software sorting method. It repeatedly traverses an array of unsorted elements, comparing two elements at a time and swapping them if needed until complete. This is the oldest and slowest sorting algorithm with a worst case complexity proportional to n^2, where n is the number of elements being sorted. Bubble sort of a bitstream can be beneficial when implemented in hardware, in contrast to the software version. Elements in the array are sorted in parallel when implemented in hardware and has a worst case complexity proportional to n. The sorted array is the thermometer code representation of the sum of the bitstream. Thermometer code is named so because it works similarly to a mercury thermometer. The mercury rises to a point and no mercury is present above this point. Similarly in thermometer code, the number represented is the number of 1s appearing in succeeding order. A 4 bit example is listed in Table 3.1.

Table 3.1: Thermometer code representation.

Thermometer base	Decimal base
0000	0
0001	1
0011	2
0111	3
1111	4

3.3.2 Convolution

The convolution of two discrete functions, $f[i]$ and $g[i]$, is defined as

$$y[i] = f[i] \star g[i] = \sum_{k=-\infty}^{\infty} f[k]g[i-k] \tag{3.10}$$

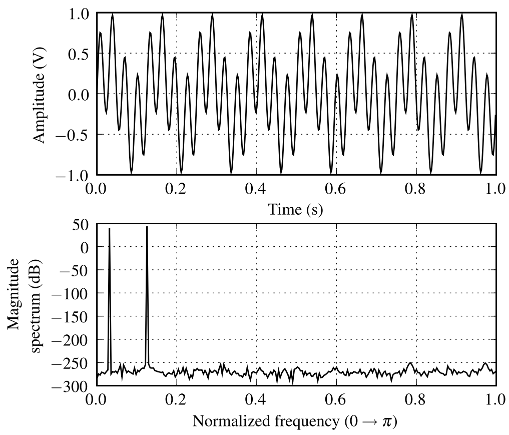
where the sum is over the appropriate values of k. This is similar to the formula for cross-correlation. If $g'[i] = g[n-i]$ where $0 \leq i \leq n$ and n is the length of g, i.e. the order of g' is the reverse of g. We can now write Eq. (3.10) as $y[i] = \sum_{k=-\infty}^{\infty} f[k]g'[i+k]$ which is the same as Eq. (3.9). This means that the running cross-correlator can be used in the large variety of applications a convolver gives us.

Digital filter

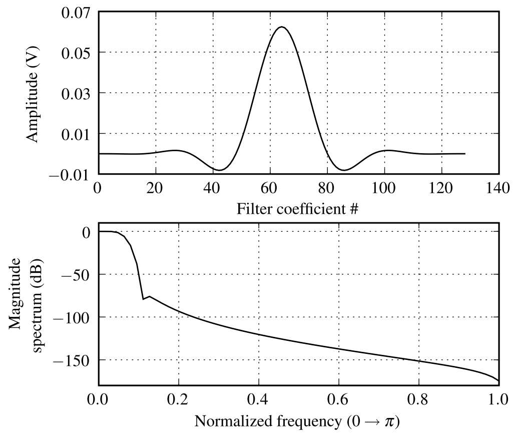
A linear time-invariant (LTI) system transforms an input signal to an output signal based on an algorithm. Convolution is the general formula for doing this for any LTI system [McCl 03]. In contrast to an Infinite Impulse Response (IIR) filter, Finite Impulse Response (FIR) filters do not require feedback. Any FIR filter can be realized by $\Delta\Sigma$ modulating the inverse sequence of the filter coefficients. A digital filter can give superior behavior compared to an analog filter, i.e. less headroom and a steeper roll-off factor, especially on ICs without the use of discreet components. The design of low frequency filters are especially advantageous and convolution is often the only solution.

A low-pass filter is often used to remove high frequency noise. As an example, consider the signal plotted in Fig. 3.7(a). The signal is the sum of two sinusoids with different amplitude and frequency.

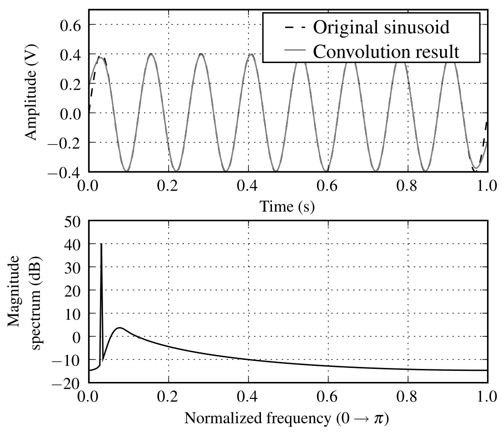
It is possible to extract the two original sinusoids of the signal by digital filtering. The low-pass filter in Fig. 3.7(b) has a cutoff frequency which will attenuate one of the sinusoids. The result of convolving the filter with the input signal is plotted in Fig 3.7(c), the time domain representation of the original sinusoid of interest is plotted together with the filter output for comparison. The frequency response of the filter output shows how the second sinusoid is attenuated.



(a) Time and frequency domain representation of the sum of two sinusoids.



(b) Time and frequency domain representation of the digital filter.



(c) Time and frequency domain representation of the low frequency sinusoid in the original signal.

Figure 3.7: Input signal (a), low-pass filter (b) and filtered signal (c).

4 System overview

The system as a whole is the result of the collective effort between Daniel Mo [Mo 09] and the writer. This chapter describes the full system, while implementation and operation of the cross-correlator is explained in great detail in Chapter 5. Developing and implementing the $\Delta\Sigma$ converter part of the system (FIR filter, CIC interpolator, DSM, CIC decimator in Fig. 4.1) and the later described clock divider is solely the work of Mo.

The chip is interfaced with a simple Serial Peripheral Interface (SPI) bus, allowing multibit inputs and outputs to be written or read serially. A block schematic showing the main signal paths of the system is shown in Fig. 4.1. Multiplexers (MUXs)

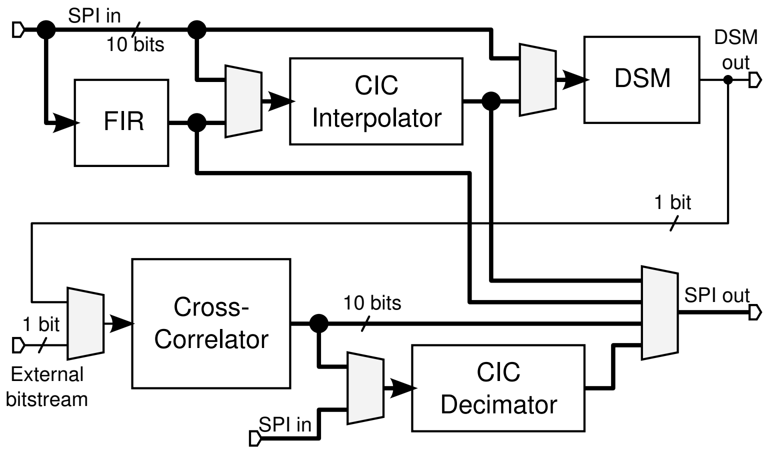


Figure 4.1: Block schematic of the system showing main signal paths.

allow for individual testing of each block, several blocks combined or the whole system all together. Only the SPI in and out pins are used for the full signal path, giving a simple interface where both input and output signals are assumed to be Nyquist rate PCM encoded.

4.1 Background of bitstream cross-correlation

The work in this thesis is based on the Letter “Running cross-correlation using bitstream processing” [Land 07]. The statement that the proposed cross-correlation technique is power efficient is based on a estimated transistor count of a traditional and a bitstream cross-correlator. An appropriate Figure Of Merit (FOM) for digital microelectronics is:

$$\text{FOM} = \text{clockspeed} \cdot \text{transistorcount}$$

The FOM_M of a multiplier based cross-correlator, as previously shown in Fig. 3.6, is derived from different recent state of the art articles and is an optimistic estimate excluding control logic. FOM_M is a function of the length of the cross-correlation window and number of bits in each sample of the signal. The FOM_B of a bitstream cross-correlator is derived on the same principles and is a function of the OSR used to produce the bitstream, in addition to the variables of FOM_M. It is assumed that the $\Delta\Sigma$ bitstream is already available. This rough transistor count will give an estimate on the rela-

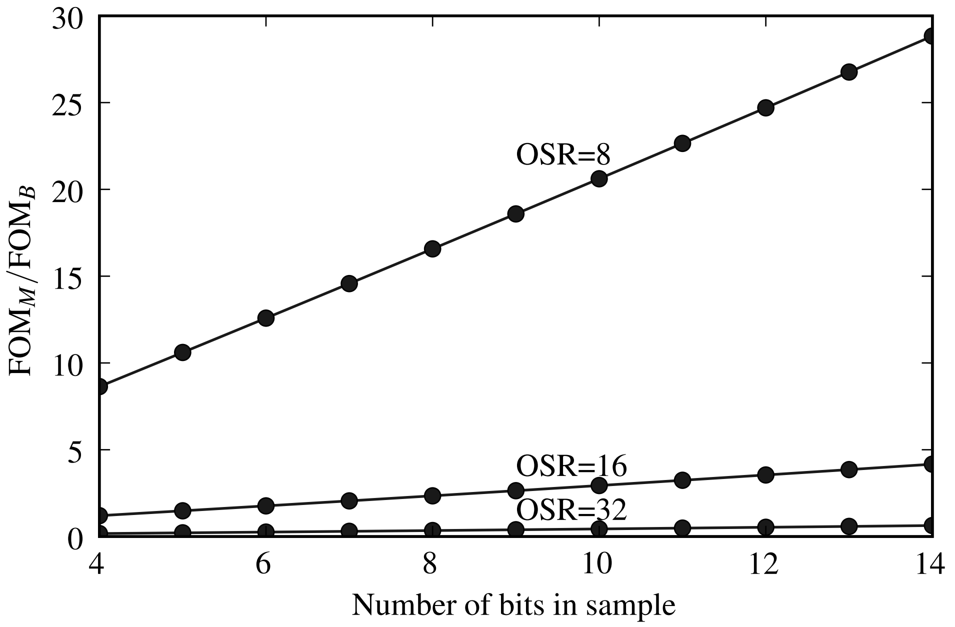


Figure 4.2: Relative improvements of bitstream cross-correlation.

tive improvements of bitstream processing. Chapter 5 will show that a bubble register length of 1024 is the upper limit of this implementation because of chip die area and, more importantly, the current drawn at the clock edges. Note that a smaller and more power efficient solution is straight forward implementable by using standard cells. The $\text{FOM}_{M/B} = \text{FOM}_M/\text{FOM}_B$ is a function of register length n. The bubble register length is set to 1024 , giving $n = 1024/\text{OSR}$.

$\text{FOM}_{M/B}$ is plotted in Fig. 4.2. The bitstream advantage is significant for low OSR and increases with the number of bits in the sample. High resolution $\Delta\Sigma$ modulators with a low OSR can be difficult to make.

4.2 System building blocks

Fig. 4.3(a) shows the layout of the implemented chip. All elements from Fig. 4.1 are identified, in addition to a clock divider. The clock divider distributes and divides the higher SPI clock rate to the different computation blocks.

The chip is produced in STMicroelectronics' 90 nm technology. The fabricator fills the die with metal layers to fulfill certain density rules and the microphotograph of the chip does not reveal much of the implemented structures. Fig. 4.3(b) shows clearly the two top metal layers, which mainly are used to distribute the supply voltages and will therefore outline each transistor in the design.

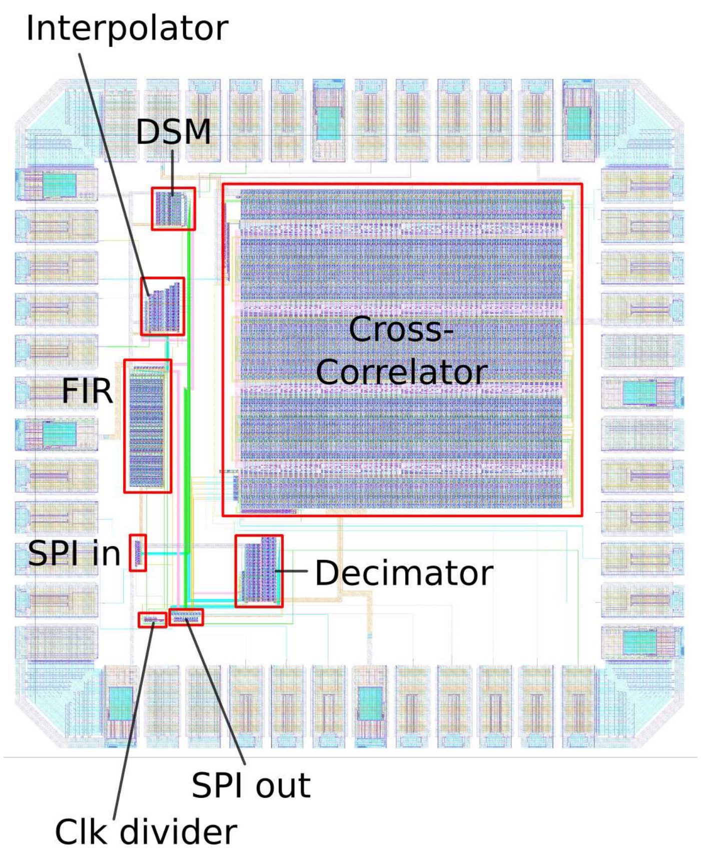
The building blocks in the system follow.

Serial Peripheral Interface

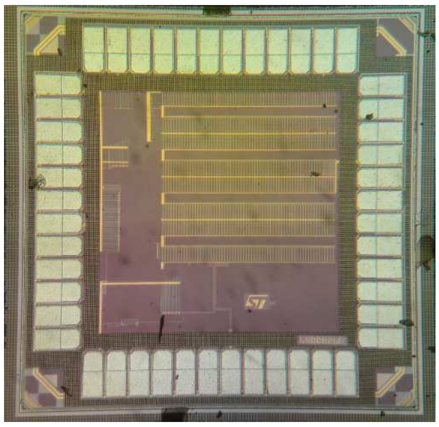
The implemented synchronous serial data link does not follow the naming conventions and SPI interface standard named by Motorola, but was designed to interface as a slave to another SPI device in master mode. Several of the chip inputs and outputs can be used as the SPI standard Master Output, Slave Input (MOSI) and Master Input, Slave Output (MISO) and there is no Slave Select (SS) to enable/disable the SPI interface on the chip. During each SPI clock cycle, a full duplex data transmission occurs:

- the master sends a bit on the MOSI line; the slave reads it from that same line.
- the slave sends a bit on the MISO line; the master reads it from that same line

The chip SPI input is simply a register reading ten bits serially, clocking out ten bits in parallel at a reduced speed. Similarly, the SPI output register latches ten bits in parallel, clocking out ten bits in serial at an extended speed.



(a) Chip layout. Chip size is 1 ×1 mm.



(b) Microphotograph of the chip die.

Figure 4.3: Chip layout and microphotograph.

Clock divider

The clock divider circuits provide operating frequencies for the different subcircuits. The clock domains are derived from the chip input pin *SPI_clk*. The *SPI_clk* is used for the increased SPI interface clock and is divided by ten to get the internal clock signal *clk_hi*. This clock is used for the oversampled version of the input signal and the bitstream. To achieve an oversampled version of the input signal, two other clock frequencies are required. The clock domains are summarized in table 4.1 and the circuits using them are briefly described throughout this section. The clocking scheme is rather

Table 4.1: The different on-chip clock domains.

Internal clock name	Clock frequency	Used in circuit
SPI_clk	10	SPI in and out
clk_hi	1	SPI in and out, CIC interpolate and decimate, DSM and cross-correlator
clk_div4	1/4	FIR
clk_div8	1/8	FIR, CIC interpolate and decimate

complex and will not be explained in further detail.

Finite Impulse Response filter and Cascaded Integrator-Comb interpolator

The chip input is expected to be at the Nyquist sampling rate and held for eight *SPI_clk* cycles and an upsampling of the signal is required before the $\Delta\Sigma$ modulation. The incoming signal is upsampled by zero padding and filtered to get a smooth interpolation. Multi stage filtering allows for different filter specifications at different upsampling rates. The total frequency response of the filters is the product of each of the filters' responses.

The incoming signal is upsampled by a factor of two and FIR filtered. The FIR filter has a sharper transition between the pass and stop band than the following Cascaded Integrator-Comb (CIC) filter, but is more hardware demanding. The upsampling stage will result in spurious signal components outside the signal band, known as aliasing.

The CIC filter in the second upsampling stage attenuates unwanted high frequency components to a larger extent than the FIR filter. The CIC filter is considerably smaller than the FIR filter and has a lower power consumption. The CIC filter constitutes the completion of the upsampling and outputs a oversampled version of the input signal with an OSR of eight.

Modulation, bitstream operations and decimation

The input to the $\Delta\Sigma$ modulator is the oversampled Nyquist signal at a rate of *clk_hi*. The modulator produces the bitstream used for the cross-correlator and is implemented as discussed in chapter 3.

The cross-correlator is fed with a bitstream at the rate of one tenth of the SPI clock, *clk_hi*. The bitstream can be read from the on-chip modulator or externally. The output from the cross-correlator is a ten bit word including high frequency noise from the $\Delta\Sigma$ noise shaping and can be decimated on-chip or read directly. The decimation filter is of third order, as required to decimate the bitstream produced from the second order modulator.

5 CMOS Implementation

A running cross-correlator using bitstream processing has been implemented in 90 nm technology. The theoretical bitstream operations were discussed in section 3.3. This chapter will describe the implementation of the bitstream operations and system.

The implemented template register is 1024 bits and the circuit contains 1024 almost identical slices. Implementation of the actual cross-correlation and bubble sorting are equal for every slice and only the thermometer to binary decoder varies from one slice to another. To manually implement this would be a time-consuming task and the possibility for errors is large. SKILL is a LISP-like CAD system extension language and was used for producing most of the schematic and layout of this system. The programming language is poorly documented, most of the knowledge to produce both schematics and layout with SKILL was obtained from "SKILL Language Reference" [SKIL 03] and online user groups. The produced SKILL code makes it possible to generate various register lengths and to choose the dimensions of the circuit, but the pre- and postlogic for the layout are hard coded due to time limitations.

The circuit, with the exception of some of the control logic, is built up from NAND and inverter standard cells delivered by STMicroelectronics. This is done because of limitations in the available standard-cell library. The die area used by the cross-correlator could be minimized by using other standard cells and/or customized cells. By using all available standard cells from STMicroelectronics instead of only NAND and inverter, the area was reduced by a factor of 0.56. The outcome of only using the two available cells was in-depth knowledge about every building block, but using more gates led to a less power efficient design.

5.1 Circuit overview

The cross-correlation is basically implemented in three stages. Bitstream multiplication is done by comparing the two bitstreams with XNOR gates. Bitstream counting is done by using bubble sort, which gives a thermometer coded result. The last stage is to convert the thermometer code to its binary representation.

The three stages are started and ended in the course of one clock cycle and parts of the computation are done asynchronously, i.e. the clockless subcircuit relaxes the clock frequency to be run at the bitstream frequency. The downside of doing this is the severe timing restrictions which apply. One operation has to be completed before the next one can start. Thorough simulations were done to ensure the proper sequential behavior and generous time margins were added to each computation step. Fig. 5.1 shows

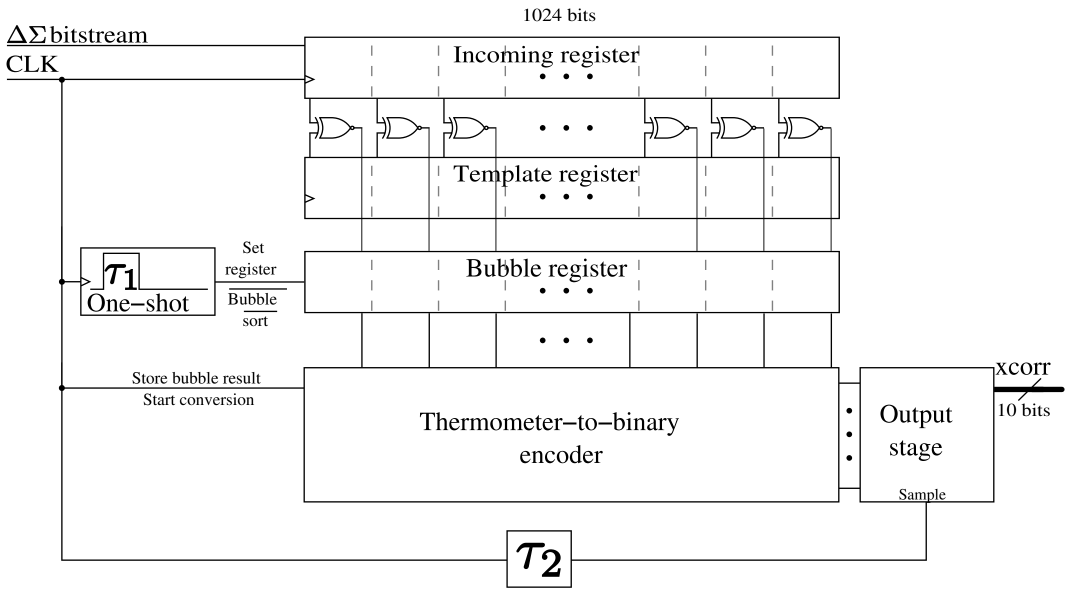


Figure 5.1: Circuit block diagram.

the block diagram of the circuit, excluding some output logic. Initially, the template bitstream which the incoming signal will be compared against, is clocked into the 1024 bits template register. The incoming signal is shifted into the incoming register and every bit in the two registers are multiplied by the XNOR gates at the start of the clock cycle. The bubble register contains 1024 SR-latches which will hold the multiplication results from the XNOR operation. The one-shot circuit controls the first important time delay. The circuit gives a high output for a short time, long enough to complete the multiplication and to latch the result into the bubble register. The pulse width is defined by the time delay τ_1 in Fig. 5.1. τ_1 has to be long enough to allow the first stage to complete, but should be kept to a minimum, maximizing the time available for the next stage.

Stage two starts when the output from the one-shot circuit goes low. The bubble register starts to sort the 1024 bit array, which is done asynchronously, still within the same clock cycle. To make this possible, the duration of time delays are again the success factor. Short time delays internally in the bubble register could cause loss of information and long time delays will add up to an unbearable sorting time. The high values in the register are shifted to the right, finally forming the result of the cross-correlation presented in thermometer code.

The worst case sorting time depends on the internal time delays in the bubble register and the array to be sorted. The sorting time and the time allowed for the first stage to complete, τ_1, restricts the maximum clock frequency of the whole circuit. The second stage has to finish before the next rising clock edge.

The second stage is completed at the subsequent rising clock edge and the third computation stage is initiated. The inputs to the thermometer to binary encoder are an array consisting of one unique transition from low to high. This transition is identified and converted to the correct binary value. The outputs from the encoder are floating nodes and have to be sampled after a short time delay defined by τ_2 in Fig. 5.1. During the lower half of the clock cycle, the output nodes of the thermometer to binary encoder are precharged to logic high.

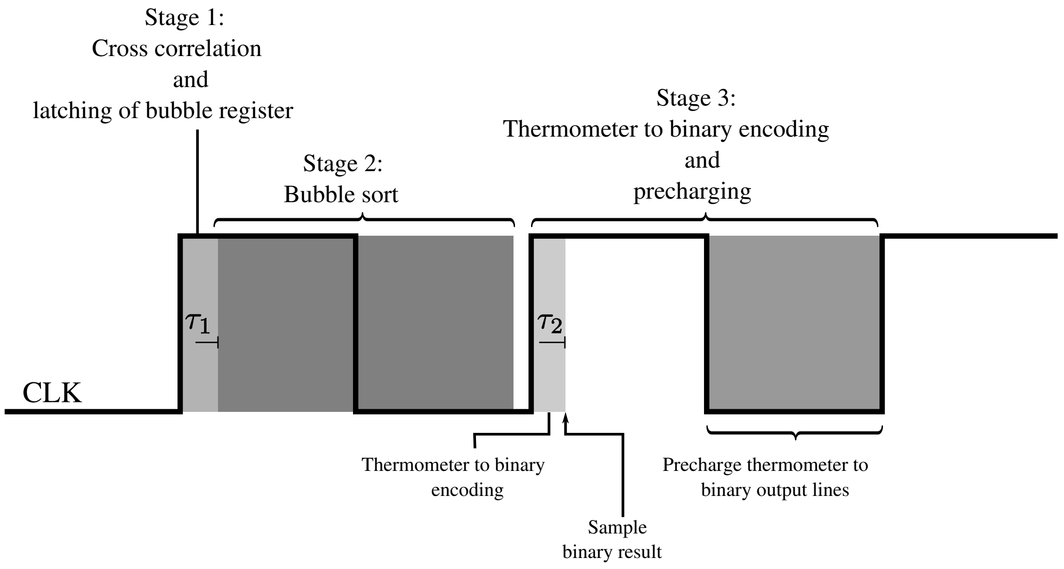


Figure 5.2: Clock timing diagram, not in scale.

Fig. 5.2 shows a coarse timing diagram for each of the main computations in the circuit. All tasks are started every clock period, but left out for clarity. Notice the unused time after the 2nd stage. This margin is similar to the previously presented margins for synchronous logic.

The next sections present the different subcircuits and time critical computing steps are discussed. The time constraints are based on Monte Carlo simulations on the actual schematics or, in most cases, models of the actual schematics.

5.1.1 Bitstream multiplication

The calculations in section 3.3.1 showed that the correct multiplication operator between to bits in a bitstream is the XNOR operation. Fig. 5.3 shows the two registers and the bitstream multiplication between them. It is possible to check both registers for correct behavior by reading the last bit.

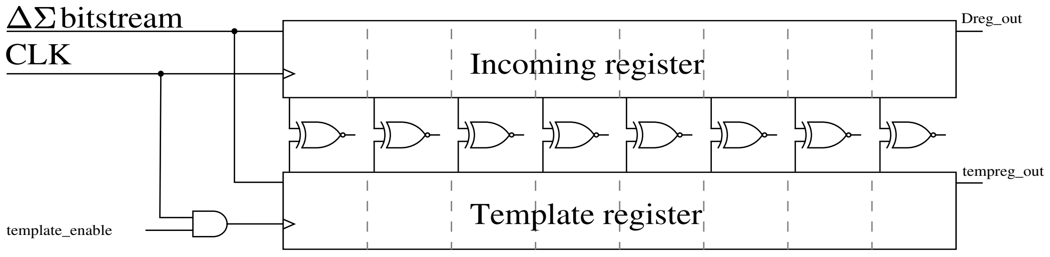


Figure 5.3: The bitstream multiplication.

The incoming signal is compared to the template and an array of bits will appear on the XNOR gates. The multiplications are done during the time delay τ_1 before the results are latched into the bubble register. The multiplication could have been calculated in its own clock cycle and the critical estimates regarding the time delay could be dismissed, but this solution would require another latching step.

Timing – Presetting the bubble register

Shifting of the register and the logic XNOR operations have to be complete before the next stage is initiated. In addition, the bubble register has to be preset with the computed array. The time when the multiplication results are latched into the bubble register is defined by the delay τ_1 in Fig. 5.1. The $\Delta\Sigma$ bitstream is shifted in the bitstream register on rising clock edge. τ_1, has to be longer than the delay from the clock edge to the last bubble register is set.

A model of one multiplication cell and one bubble sort latch was created and a Monte Carlo simulation was executed. The Monte Carlo analysis generates a set of random samples for process and mismatch parameters and simulates the system for each set. The system as a whole is impossible to simulate with the data power available, insufficient memory is encountered during initial setup, even before the netlist is produced.

All driving circuits and component loads are carefully modeled. Resistance and capacitance of wires were omitted. The goal of this simulation is to find an appropriate length of the two delays:

τ_{1min}	Actual computation time for Stage 1
τ_1	Allowed computation time for Stage 1

The length of the delay τ_{1min} is simulated and an element with a delay τ_1, where $\tau_1 > \tau_{1min}$, is constructed.

The time from the clock reaches $0.9 \cdot V_{dd}$ to the output of the latch in the bubble register reaches $0.9 \cdot V_{dd}$ is simulated 1000 times with process and mismatch variation. This gives the time from rising clock edge, through the completion of the bitstream multiplication, to presetting of the bubble register. The number of simulations is a trade-off between accuracy and time, 1000 simulations gives a good enough estimate of the time delay. The simulation indicates an estimated mean $\mu_{\tau_{1min}} = 199$ ps and an estimated standard deviation $\sigma_{\tau 1min} = 20$ ps. The simulation is plotted in Fig. 5.4 together with the normal distribution $Y \sim= N(\mu_{\tau 1min}, \sigma_{\tau 1min})$.

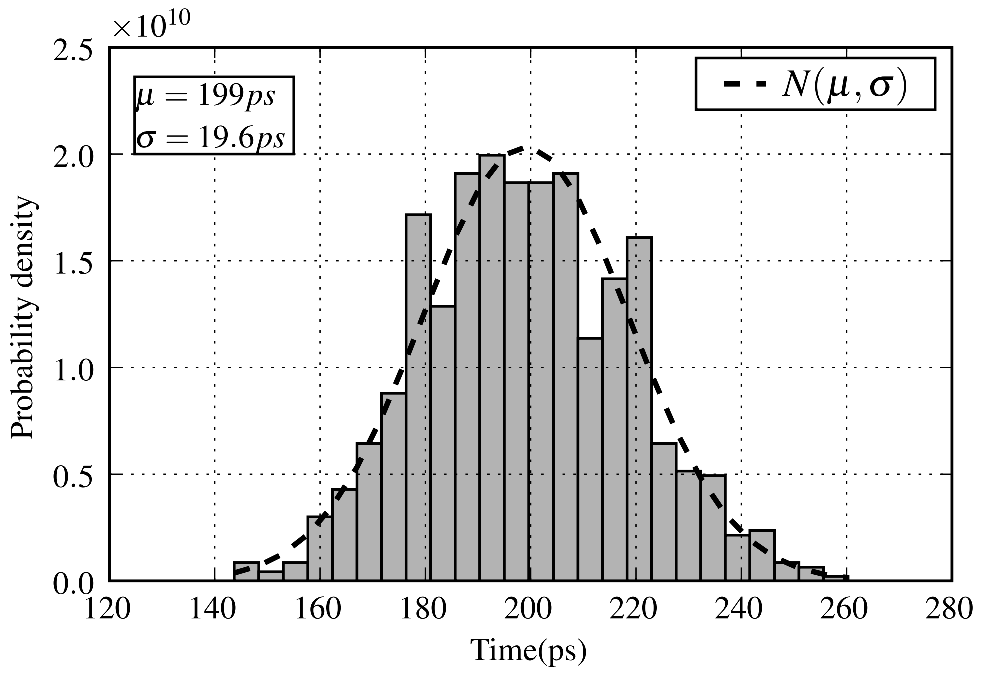


Figure 5.4: Statistical analysis of the time delay from rising clock edge to the bubble register is set.

The normal distribution involves two parameters, μ and σ, where μ is the mean of the distribution and σ is the standard deviation:

$$f(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, -\infty < x < \infty$$

Since the Monte Carlo simulates the system a large number of times with random parameters, the normal distribution is a good estimate of the probability distribution of $\tau_{1_{min}}$. The following calculations are based on simulations and estimates of the actual values which are not explicitly addressed for better readability.

The actual delay is normally distributed, $X \sim N(\mu_{\tau 1}, \sigma_{\tau 1})$ and is realized with inverters. Simulation shows that the standard deviation for a delay in the relevant scope is similar to τ_{1min}. The same simulation indicated the correlation coefficient between the delays, $\rho \approx 0.7$. The correlation measures the direction and strength of the linear relationship between two quantitative variables. The difference $D = X - Y$ between the delays is normally distributed, with mean and standard deviation:

$$\begin{aligned}
\mu_D &= \mu_{\tau 1} - \mu_{\tau_{1min}} \\
\sigma_D &= \sqrt{\sigma_{\tau 1}^2 + \sigma_{\tau 1min}^2 - 2\rho\sigma_{\tau 1}^2\sigma_{\tau 1min}^2} \\
&\overset{\sigma_{\tau 1} = \sigma_{\tau 1min}}{=} \sigma_{\tau 1min}\sqrt{2\left(1 - \rho\sigma_{\tau 1min}^2\right)} \approx 28.3 \text{ ps}
\end{aligned}$$

$D \sim N(\mu_D, \sigma_D)$. The mean, $\mu_{\tau 1}$, which results in 99.85% possibility for $\tau_1 > \tau_{1min}$ is calculated. Because D is one-sided, this is ensured when $P(D < 0) = 100\% - 99.7\% = 0.003$.

$$\begin{aligned}
P(X < Y) &= P(X - Y < 0) = 0.003 \\
&= P\left(\frac{D - \mu_D}{\sigma_D} < \frac{0 - \mu_D}{\sigma_D}\right) = 0.003 \\
&= P(N(0,1) < -2.75) = 0.003
\end{aligned}$$

Which implies:

$$\begin{aligned}
-\frac{\mu_D}{\sigma_D} &= -2.75 \\
\mu_D &\approx 78 \text{ ps}
\end{aligned}$$

Finally, μ_τ is found:

$$\mu_\tau 1 = \mu_{\tau 1min} + \mu_D = 199 \text{ ps} + 78 \text{ ps} = 277 \text{ ps}$$

The bubble register will be preset when its control signal is high and start bubbling when it is low. The output from the one-shot circuit in Fig. 5.1 goes high on the clock rising edge and low after the predefined time delay, τ_1. The delay has to be long enough to allow completion of the bitstream multiplication and preset the bubble register. This is fulfilled within a probability of 99.85% by implementing a delay with a mean of 277 ps.

The actual implemented delay of τ_1 is simulated to have a mean $\mu = 344$ ps and a standard deviation $\sigma = 19$ ps. Because of time restrictions when implementing the circuit, the requirement to the delay was $1.5 < \mu_{\tau 1}/\mu_{\tau 1 min} < 2$ and ended quite randomly to be $\mu_{\tau 1}/\mu_{\tau 1 min} \approx 1.7$. This allows a variation in τ_1 of more than $5\sigma_D$.

The fulfilled calculations and simulations show that the width of the one-shot pulse gives enough headroom to always ensure that the bubble register is preset and is short enough to maximize the sorting time in the next stage.

5.1.2 Bitstream summing

The number of high outputs from stage one need to be summed to complete the cross-correlation. A standard approach to this problem could be to store the array of bits and count them by shifting them out of the register. This would delay computation and/or require a higher clock rate, resulting in a higher power dissipation. Another way to sum the array is to sort all the high values to the right and then convert the now thermometer coded array to its binary value. This sorting, even combined with the cross-correlation, can be done in one clock cycle. The bubble register contains a regular SR-latch register with an AND gate and a time delay as shown in Fig. 5.5. In

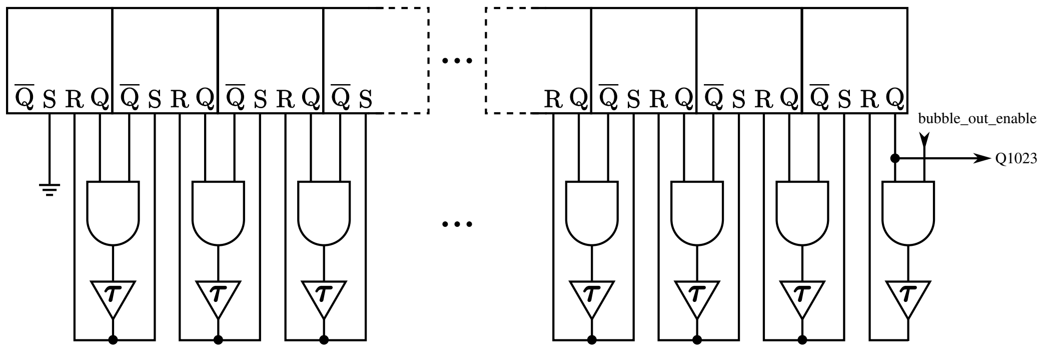

$\overline{Q}$ S R Q $\overline{Q}$ S R Q $\overline{Q}$ S R Q $\overline{Q}$ S
R Q $\overline{Q}$ S R Q $\overline{Q}$ S R Q $\overline{Q}$ S R Q
bubble_out_enable
Q1023
T

Figure 5.5: Continuous bubble sorter

addition, logic to preset the latches in the bubble register is implemented. At normal operation, the input signal *bubble_out_enable* in Fig. 5.5 is grounded. Setting this signal

]

to high, will bubble the data in the register to the output pin *Q1023*. The data can be read and controlled for correct behavior with a high frequency oscilloscope.

Table 5.1 presents the SR-latch operation. The state of $R = S = 1$ will not fulfill the

Table 5.1: SR-latch operation.

S	R	Action
0	0	Keep state
0	1	$Q = 0, \overline{Q} = 1$
1	0	$Q = 1, \overline{Q} = 0$
1	1	Restricted combination

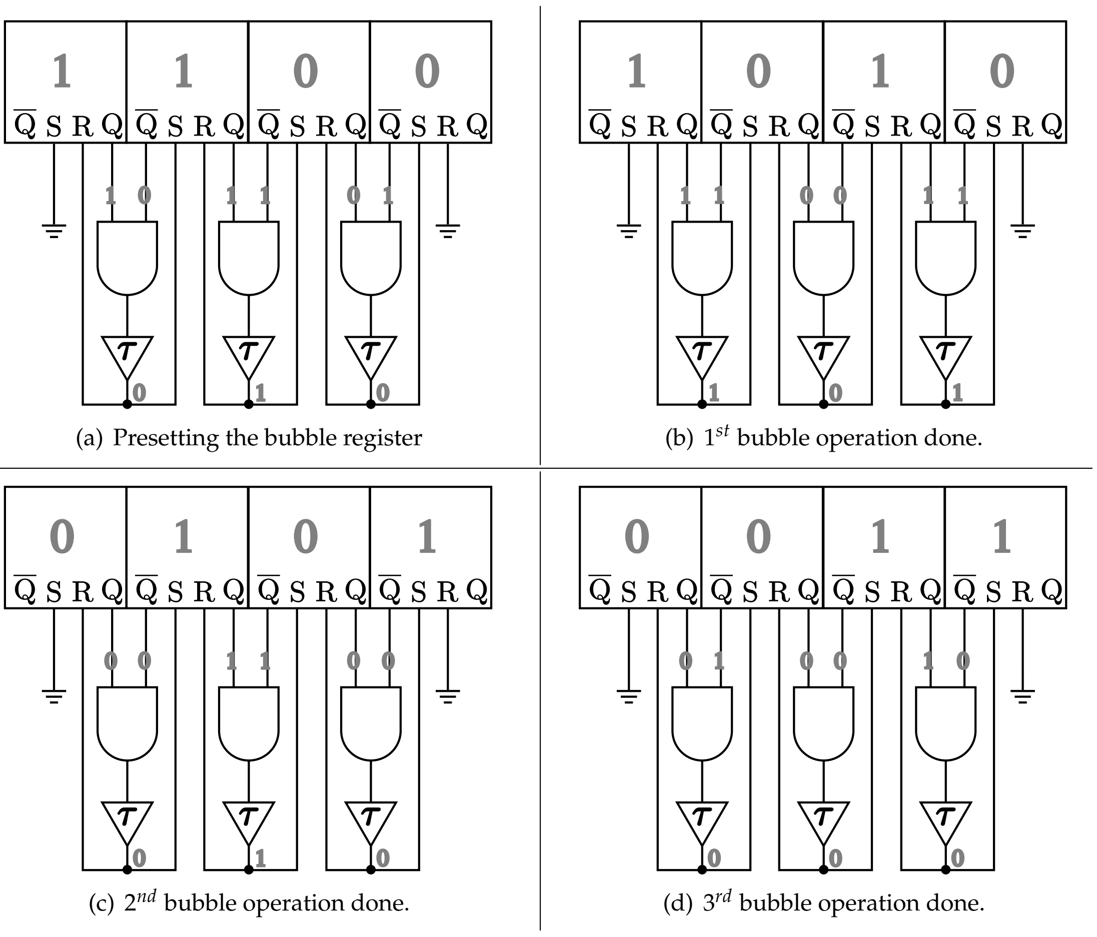
logic equation $Q = \textbf{not}\ \overline{Q}$ and is therefore a restricted combination. This state can also cause a race condition, where the final state can not be predicted.

Considering i^{th} bubble element, Q[i], the following pseudocode summarizes the behavior of the bubble register:

```
while not sorted:
  for all bubble elements, i, in register at once:
    if Q[i] > Q[i+1]:
      swap(Q[i], Q[i+1])
```

The $(i+1)^{th}$ bubble element will input and store a 1 only if its value is 0, $Q[i+1] = 0$, and its predecessor, Q[i], is 1. This results in a ripple of logic 1s from left to right in the register. It is also possible to think of the sorting as a ripple of logic 0s from right to left in the register. The circuit has a aesthetic and symmetric behavior which will be thoroughly described throughout this section.

The bubble sort of a four bit bubble register is illustrated in Fig. 5.6. Four bit is enough to demonstrate the different sorting operations. Fig 5.6(a) shows the initial state of the bubble register. Here, the SR-latches are preset with the values from the bitstream multiplication. The logic value of each node is indicated in gray. Only a logic 1 followed by a logic 0 in the register will result in two high inputs to the accompanying AND gate which will reset the latch originally valued at logic 1 and set the next. In other words, the logic 1 is bubbled one latch to the right sequentially, i.e. without a clock. The next bubble operation, illustrated in (b), involves bubbling of two bit since both high bits have a consecutive logic 0. Note that the described behavior is explained in discrete time; in reality each bubble will occur at slightly different times. The rightmost bit in (c) has reached its stable state and the latch can be neither reset nor set. The last unsorted bit will be bubbled in the next and final bubble operation. (d) visualizes the final stable state of the bubble register. All inputs to the SR-latches are zero and all



(a) Presetting the bubble register

(b) 1st bubble operation done.

(c) 2nd bubble operation done.

(d) 3rd bubble operation done.

Figure 5.6: Sorting of a four bit array.

outputs are kept, meaning that the array is sorted and ready to be encoded. This array is one example of the most time consuming patterns to sort. $n - 1$ bubble operations are used to reach the final stable state, where n = register length.

The graph in Fig. 5.7 shows the outputs from a bubble register at a given time, t. The data plotted are slightly simplified simulation results, to improve clarity. The plot is valid for one of the cases in the previously discussed example and shows how the logic 1s will propagate through the bubble register as a wave, eventually filling the register from right to left.

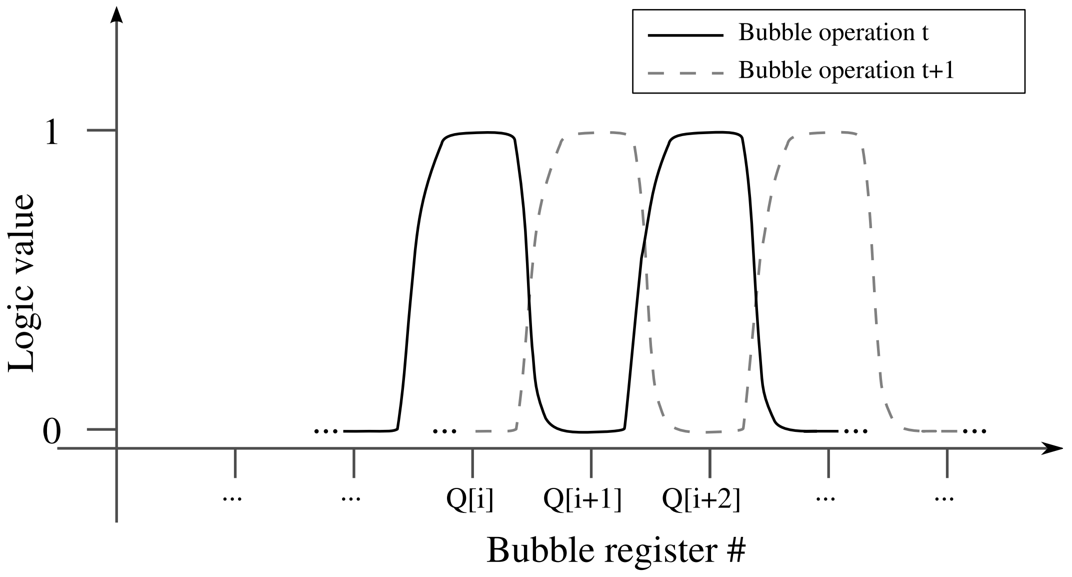


Figure 5.7: Asynchronously sorting logic values in the bubble register.

Timing – Bubble sort

The sorting is the most time consuming process and depends on the template bit length. The transient response of sorting a '1' and fifteen '0' is plotted in Fig. 5.8. The '1' is bubbled through the register until the sorted stable state is reached. The time delay between each bubble is a combination of the time delay in the two inverters, the AND gate and the latch set time seen in Fig. 5.6. Every bubble, except the first and the last, takes 156.6 ps.

The time a value is kept in the SR-latch depends on the time delay, τ_3 in Fig. 5.5. The delay has to be longer than the difference in the latch set and reset time to avoid metastability in the latch. A short delay could cause one latch to be reset without setting the next one. This delay is less defined than the previously calculated τ_1. The difference in the latch set and reset time has a mean of 5 ps with a worst case value of 11 ps. The delay in the AND gate should be more than long enough to ensure stability in the latch, but some bubble operations fail when running a Monte Carlo simulation on the bubble register.

τ_3 is realized with two inverters, in addition to the delay of the AND gate, and simulation confirms the operation of the circuit. The delay time of one bubble operation will limit the maximum operating frequency of the whole system. A four bit bubble register was created to make the statistical analysis of the bubble sort possible. The time

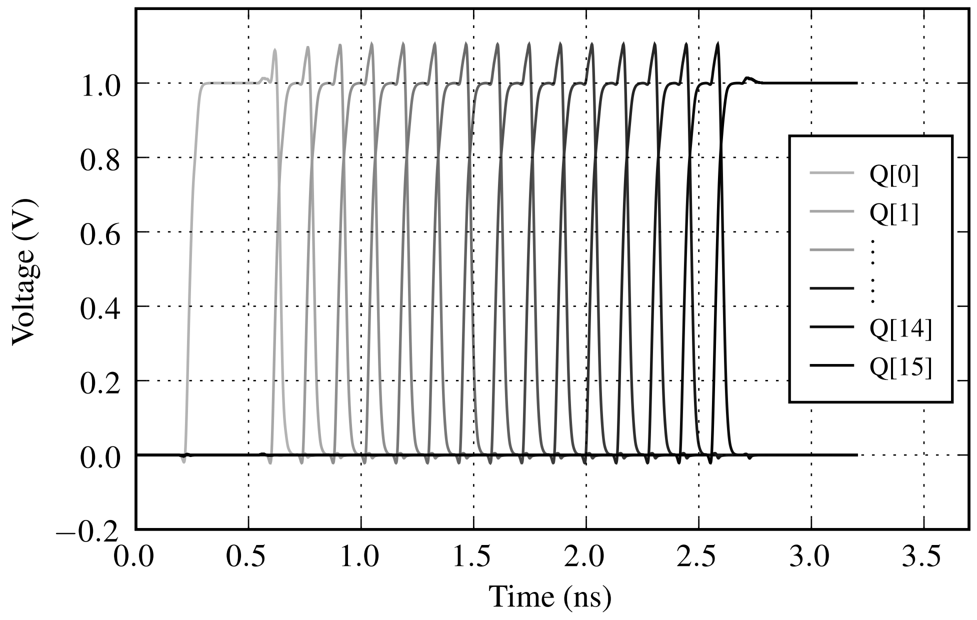


Figure 5.8: Transient response of a 16 bit bubble sort.

from the second and the third bit was measured with mean $\mu = 144$ ps and standard deviation, $\sigma = 14.1$ ps.

5.1.3 Thermometer to binary encoder

A straightforward and common approach to the encoding is to use a binary encoded ROM. The 1024 bit thermometer code is encoded to a 10 bit binary value. The encoding is executed in the subsequent clock cycle after cross-correlation, in parallel to computation of the next cross-correlation result. This relaxes the timing properties of the encoder, but involves another latch stage to hold the thermometer code. The thermometer code consists of a sequential number of 1′s corresponding to which number it represents, hence the input to the encoder is an array consisting of one unique transition from low to high. This transition is identified by a row decoder, comparing two consecutive values from the thermometer coded array. The selected row pulls down the correct precharged output lines which constitutes the correct binary representation. The output lines are sampled after a predefined time delay, τ_2 in Fig. 5.1. This type of row decoder selects multiple rows if a bubble error occurs, i.e. if the thermometer coded array is inconsistent. Several bubble error correction schemes are applicable at the cost of increased silicon area, propagation delay and power consumption [Sail 07]. A bubble error in the bubble sorted array means that the sorting is incomplete and it would be meaningless to implement bubble error correction.

Implementing the system with SKILL was especially beneficial regarding the thermometer to binary encoder. The correct pulldown transistors were automatically placed according to encoding conversion. Doing this manually would have been time consuming and error-prone.

5.1.4 Clock buffers

Large clock buffers were used instead of a clock distribution network. Construction of a clock tree without clock skew is challenging and clock edge time differences in the flip flop register and the bubble register will result in corruption of data.

The clock buffers drive large loads and should have the right fan-out. Fan-out is a measure of the ability of a logic gate output to drive a number of inputs of other logic gates. A gate driving h identical copies of itself is said to have a fan-out or electrical effort of h. The driving gates in the clock buffers are inverters and the load is not an identical copy of the gate. The electrical effort is then computed by $h = \frac{C_{out}}{C_{in}}$, where C_{out} is the capacitance of the external load and C_{in} is the input capacitance of the gate. Assuming that polarity does not matter, the in- and output capacitances are proportional to the gate sizes. With this assumption, estimates to the fan-out of the different clock buffers are found. The buffers used in the implementation had an electrical effort between five and seven, which is higher than the well known Fanout-Of-4 (FO4). A FO4 would have given the shortest time delay, but smaller buffers were prioritized.

There will be a significant lag from when the first gate receives a clock transition to when it is received at the last gate. The clock buffer drives many gates and long lines, close to 5mm. A clock tree could prevent the delay, but clock skew could cause different parts to receive clock transitions at different times. With knowledge of which gate receives the clock transition first, the registers are implemented to function independently of the clock transition delay. The registers will operate as intended by letting the clock transition flow in the opposite direction of the data path. The wanted behavior in a register is that a flip flop samples its input before the predecessor changes its output. The template and incoming shift registers are designed with the clock transition delay flowing in the opposite direction of the data. In the bubble register, the data and clock transition delay flows in the same direction because of a design flaw. This can potentially corrupt data.

Current drawn and conductor widths.

The described clock buffers drive large loads, many gates and long conductor lines. The current drawn at the clock edges was a concern at design time and thorough simulations were performed to examine the performance. A simplified model, consisting of a clock buffer driving an equivalent load, was created to execute the simulation. Simulation of the current flow from the clock buffer shows values around 55 mA on clock edges.

A typical sheet resistance for the 90 nm process is $R_{\square} = 70\ \text{m}\Omega/\square$. The conductor is close to $l = 5$ mm long. Conductors wider than 4 µm are in this design impractical so the widest conductors were chosen to $w = 4$ µm. Total resistance of the conductor is then given by $R = R_{\square}\frac{l}{w} = 70\ \text{m}\Omega/\square\frac{5\ \text{mm}}{4\ \mu\text{m}} = 87.5\ \Omega$.

The conductor resistance should be modeled in order to get a more accurate simulation. Resistances are placed in series in between the gates and the simulation is rerun. This indicates a current spike of 33 mA.

The theoretical voltage drop over the conductor is then given by Ohm's law $U = RI = 87.5\ \Omega \cdot 33\ \text{mA} \approx 2.9$ V. Capacitances and a voltage drop on the conducting node will in practice restrict and spread the current drawn and the lines are designed with a width of 4 µm.

5.2 SKILL implementation

The use of SKILL for implementing the schematics and layout made it effortless to generate designs with different register lengths and outer dimensions. This made it easy to explore different solutions and to make smaller designs which made simulations possible.

SKILL is based on the artificial intelligence language Lisp and two different programming notations are allowed:

- Algebraic notation used by most programming languages,
  e.g. `func(arg1 arg2 ...)`
- Prefix notation used by Lisp, e.g. `(func arg1 arg2 ...)`

The algebraic notation style, or C-like syntax, is said to allow a novice user to quickly learn to use the language, while the Lisp syntax allows expert programmers to access the full power of the Lisp language. To demonstrate the difference, a short code snippet, calculating a Fibonacci number, follows:

Calculating a fibonacci number using algebraic notation.

```
procedure( fibonacci(n)
  if( (n == 1 || n == 2) then
    1
  else fibonacci(n−1) + fibonacci(n−2)
  )
)
```

Calculating a fibonacci number using prefix notation.

```
(defun fibonacci (n)
  (cond
    ((or (equal n 1) (equal n 2)) 1)
    (t (plus (fibonacci (difference n 2))))
  )
)
```

All knowledge of the Lisp syntax was suppressed before coding of the SKILL scripts commenced.

5.2.1 Schematics

The different building blocks of the schematic, except the thermometer to binary encoder, were drawn manually and consist of three main parts: (1) Prelogic, e.g. control signals, buffers, delays. (2) A one bit slice, containing the bitstream multiplication and bubble sort. (3) Postlogic, e.g. outputs, delays. A SKILL script assembles the building blocks with the register length, *n*, as a parameter; the postlogic is placed, before *n* slices, each slice with its unique pulldown logic, are generated and placed. The pulldown logic corresponds to the binary representation of the placement of each slice. Finally, the placing of the postlogic constitutes completion of the circuit. The SKILL script can be summarized in pseudocode:

```
def AssembleSchematics(length)
  CreateCellView('crosscorrelator\_' + length + 'bits')
  PlaceCellView(PreLogic)
  for x in range(len(length)):
    PlaceCellView(Slice, x)
    PlaceCellView(PulldownLogic(x), x)
  PlaceCellView(PostLogic, length)
```

The script places each slice in a consecutive line and the resulting schematic is quite impractical, at least for long register lengths. The script is kept as basic as possible, avoiding complicated array and bus structures, minimizing the risk of errors. The unhandy format of the schematic is rather an advantage. Correct behavior is confirmed by generating a design with a short register length. Changes are not done in the generated schematic, but in one of the three basic building blocks.

5.2.2 Layout

The layout was generated in the same way as the schematic, but differs in the way the slices were placed to fit within the given dimensions. The slices were placed in a serpentine pattern and every other row rotated 180° to ease the routing of supply voltages as seen in Fig. 5.9. The script to generate the layout has the same structure

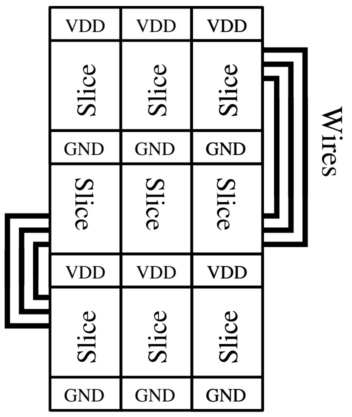


Figure 5.9: Slices placed in a serpentine pattern.

as the schematic script, the same three building blocks are placed and connected to each other. The essential difference is how the slices are placed because of the physical boundaries an actual implementation involves. Notice the different length of the wires in Fig 5.9. The largest challenge was to dynamically calculate the overall length as well as the length before a turn in each wire. Additionally, the vertical distance between the rows depends on the register length because the output lines of the thermometer to binary encoder are placed at the bottom of each slice. A small offset between two wires would most likely be intercepted by Design Rule Check (DRC) and a large offset would most likely be noticed by the layout versus schematic check. Generation of a circuit with a small register length and manual inspection of each connection was again an important debugging method and at the same time a significant source to layout optimization.

An example SKILL script can be found in Appendix B, the script generates l bits 4 to 1 MUXs from a slightly modified 1 bit 4 to 1 MUX, where l is the function parameter. The result of running the script with parameter $l = 10$ is also shown in the appendix.

5.3 Clock cycle time properties

The cross-correlator is a hybrid of synchronous and asynchronous design. This complicates a speed and power performance analysis and some concepts are established before the bitstream cross-correlator performance is evaluated in section 6.4.

5.3.1 Maximum clock frequency

Bitstream multiplication, presetting of the bubble register and the sorting will be done in one clock cycle and are the critical and most time consuming operations. The time delays bound to these operations will restrict the maximum clock frequency. The mean time properties of interest are:

$$\begin{aligned} \tau_1 &= 344 \text{ ps} \\ t_{bubble} &= 144 \text{ ps} \\ n &= 1024 \\ O_{worst\_case} &= n - 1 = 1023 \end{aligned}$$

τ_1 is the time disposed for the bitstream multiplication and time for presetting the bubble register. t_{bubble} is the time delay for one bubble operation. The register has a length n and the worst case number of sorting operations is given by O_{worst_case}.

The maximum clock frequency is determined by the time delay of the bitstream multiplication, presetting of the bubble register and the worst case sorting time:

$$\begin{aligned} f_{max} &= \frac{1}{t_{preset} + t_{bubble} \cdot O_{worst\_case}} \\ &= \frac{1}{344 \text{ ps} + 144 \text{ ps} \cdot 1023} \\ &\approx 6.8 \text{ MHz} \end{aligned}$$

This result gives an upper limit for the clock frequency and should not be confused with speed performance, although a performance analysis can indirectly be concluded from this number. One of the challenges in asynchronous design was identified in section 2.3 as performance analysis and the comparing of performance against traditional designs. The performance of the MiniMIPS, discussed in section 2.4, was measured in MIPS and the comparison with its synchronous counterpart was fairly trivial.

5.3.2 Computational performance

All available references to multiplication and summing operations in hardware is between PCM coded numbers and the bitstream cross-correlator performance should be converted to operations between Nyquist rate signals. Each clock tick initiates bitstream multiplication of 1024 bit and summing of this result. The Nyquist rate is given by f_{clk_hi}/OSR and the Nyquist template rate by n/OSR.The Nyquist rate multiplications per second is then given by

$$\text{Nyquist rate multiplications per second} = \frac{f_{clk\_hi}}{OSR} \cdot \frac{n}{OSR} = \frac{f_{clk\_hi} \cdot n}{OSR^2} \tag{5.1}$$

where f_{clk_hi} is the cross-correlator clock rate and n is the length of the bubble register. Each cross-correlation result is the sum of n multiplication results and the Nyquist rate cross-correlation is given by

$$\text{Nyquist rate cross-correlations per second} = \frac{f_{clk\_hi}}{OSR} \quad , \tag{5.2}$$

which is independent of the bubble register length since the summing is done asynchronously, at the speed of the Nyquist rate. Substituting the system parameters, $f_{max} =$ 6.8 MHz, $OSR = 8$ and $n = 1024$, gives

$$\begin{aligned} \text{Nyquist rate multiplications per second} &\approx 109\ \text{M} \\ \text{Nyquist rate cross-correlations per second} &= 850\ \text{k} \end{aligned}$$

The operating frequency used is the theoretical absolute maximum speed assuming that the time delays are equal to the mean simulation results. This result will not be tested as the measurement setup does not allow for such high frequencies.

6 Measurements

There were two alternatives to measure on and operate the circuit, and the use of a multifunction Data Acquisition module (DAQm) from National Instruments was initially chosen. The high-level programming language Python was used to implement the accompanying C library. This solution was at first successful, but rather complex. After execution of the main measurement tests, it was clear that the circuit did not work as intended. New tests were accomplished to understand when and what went wrong, but a lot of effort was used to operate and to understand the acquisition module. The second measurement setup was chosen, the use of a microcontroller (μc) module connected to a computer. The μc 's IO pins and SPI interface offered a good way to communicate with the circuit. Firmware to the μc was customized, a converter Printed Circuit Board (PCB) was milled and computer software was written.

6.1 Measurement setup

6.1.1 Printed Circuit Board

The CMOS die is packed in a 48 pin Thin Quad Flat Pack (TQFP). The TQFP48 package and all other components used are Surface Mounted Devices (SMDs). In addition to the obvious communication between the chip and the measurement tool of choice, the PCB provides level shifting of voltages and distribution of power supply lines.

The binary logic levels are represented by different voltage levels by the chip, acquisition module and the μc and are summarized in table 6.1. Level shifters are used

Table 6.1: Typical binary logic levels

	Logic level	Low voltage	High Voltage	Supply voltage (V_{DD})
Chip	CMOS	0 V – 0.5 V	0.5 V – 1 V	1 V
DAQm	TTL	0 V – 0.8 V	2 V – 5 V	5 V
μc	TTL	0 V – 0.8 V	2 V – 5.5 V	3.3 V

to translate the binary values to the correct voltage levels, depending on the supply voltage provided. This ensures right voltage levels independent of the measurement module.

Three voltage regulators are needed to drive the chip and the level shifters on the PCB. The chip requires two different voltage supplies, both 1V. The on-chip cross-correlator is given its own power supply to secure operation and to make current measurements possible. Control logic and the other project on the chip is powered by the second 1 V power net. The third power net is reserved for the chip output level shifters, shifting the logic high voltage to 3.3 V. All input and outputs are assembled in a 40 pin flat cable connector. Fig. 6.1 shows the fully assembled PCB and the different components are identified.

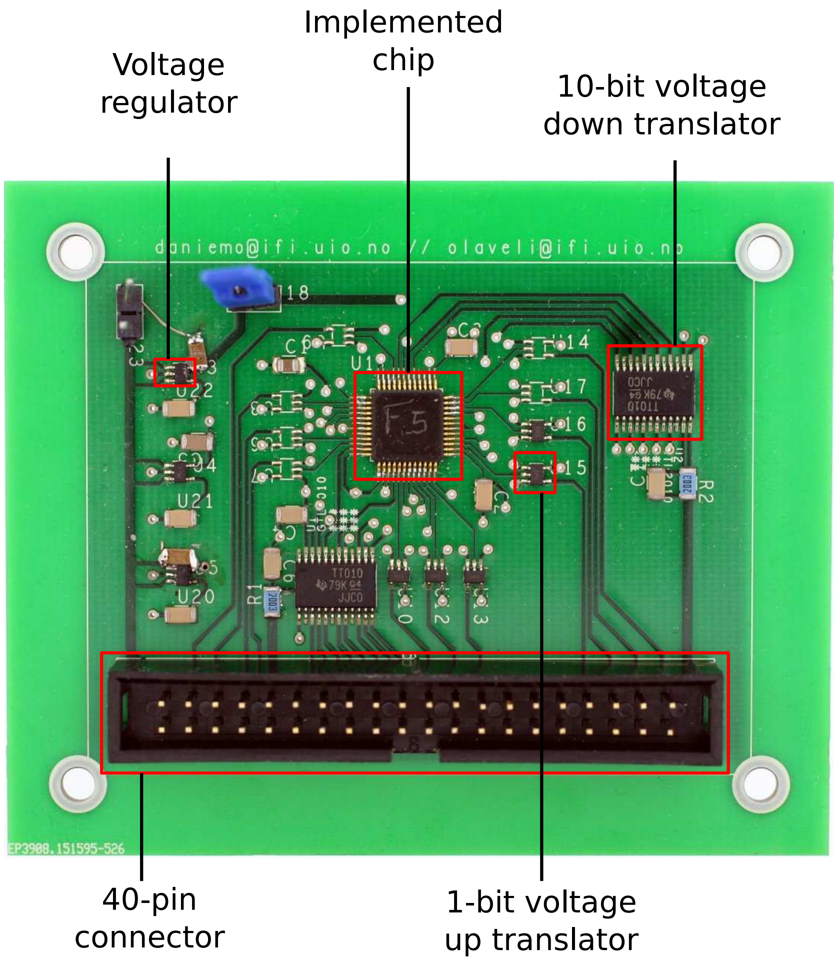


Figure 6.1: The fabricated PCB.

6.1.2 Measurement setup 1: Multifunction DAQm

The PCB was designed to function with the DAQm and a flat cable easily connects the two systems. Power supply, clock and input/outputs (I/Os) were controlled by the DAQm. The accompanying software allows graphical programming for measurement and automation, but was quickly rejected. A generic platform, not bound to the restrictions of a graphical interface, was preferred. We chose to implement the DAQm C library. A python back-end module was implemented, handling low level operations between the computer and the DAQm. The python front-end ensured a simpler user interface.

This solution worked well when outputs were the expected results from the inputs, but debugging of unexpected behavior was difficult and time consuming.

6.1.3 Measurement setup 2: Microcontroller development board

Other projects with similar I/O schemes have successfully accomplished measurements with use of *Olimex Header Board Atmel SAM7-256*. The module is powered and programmed through the USB port and I/Os are easily connected to 20 pin connectors. Generic firmware framework for the μc was adapted and a 40 pin to 2*20 pin PCB adaptor was cut. An excerpt of the firmware can be found in Appendix C. The SPI on the μc made communication to the chip intuitive and nearly trivial.

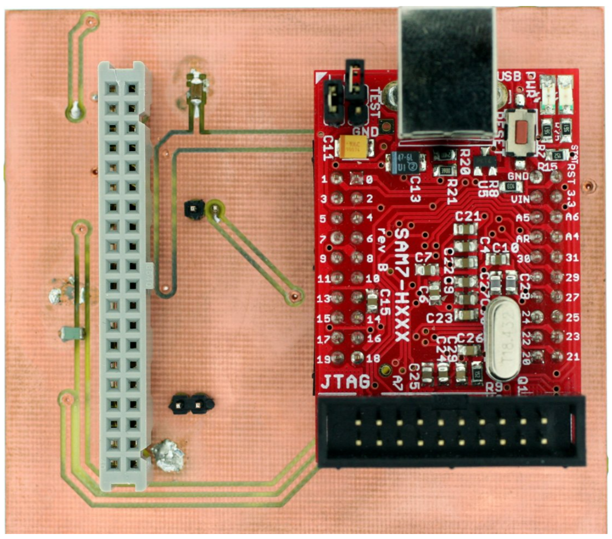
The μc is accessed from the computer through a virtual serial port link, where raw text commands are issued. An excerpt of the Python chip library can be found in Appendix D. The commands are interpreted on the μc and the requested action is fulfilled.

This solution is highly recommended for similar projects and measurement schemes. Fig. 6.2 shows the adaptor, μc and fully assembled measurement setup.

6.2 Initial measurement tests

6.2.1 D flip-flop registers

The last bit of the template and incoming register can be read. By clocking in random bit sequences, both registers were confirmed functional. This does not only confirm the registers, several other elements in the circuit function. The chip input clock, *SPI_clk* ticks ten times faster than the internal divided clock, therefore each bit in the registers are read on the output ten times. This confirms the functionality of the clock divider.


(a) PCB to μc adaptor and μc .

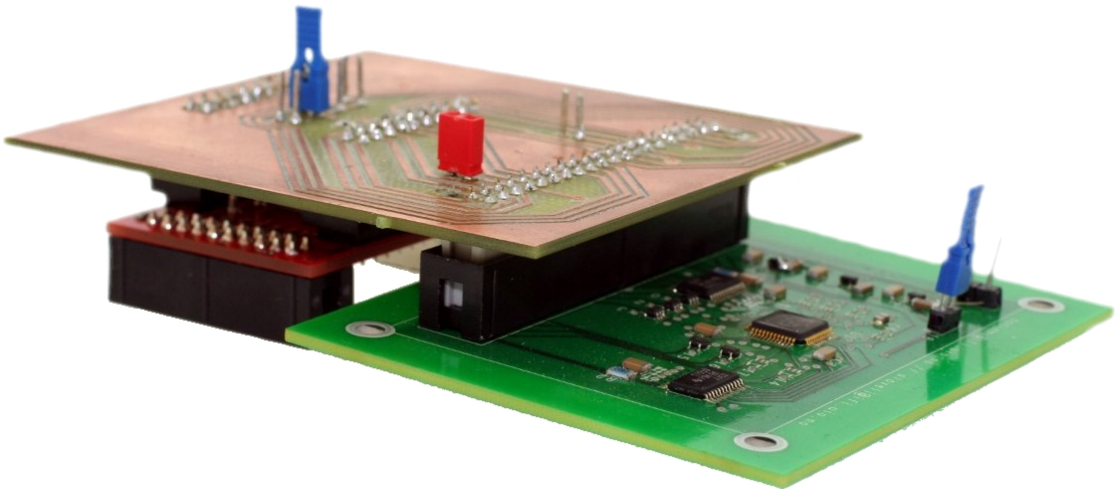

(b) The preferred measurement setup, PCB, adaptor and μc

Figure 6.2: Measurement setup 2.

Clock buffer sizes and conductor widths were discussed in section 5.1.4. The registers were read at the highest speed supported by the μc and this confirms the buffer and conductor dimensions.

The PCB layout, including different voltage nets and voltage up/down translations, is confirmed and the measurement tool functions as expected.

The initial test confirms the operation of a basic flip-flop register, but more importantly, the test verifies the operation of large parts of the system as a whole.

6.2.2 Cross-correlation

The result from the cross-correlation is a ten bit number from 0 to 1023 depending on how alike the template and the incoming signal are. The first test revealed that the cross-correlator did not operate as intended. Different measurements were run to figure out what went wrong. These are presented chronologically in the following sections to show the chain of thoughts in the debugging process.

From maximum cross-correlation to zero to maximum again

By storing zeros in the template and incoming register, the output from the circuit is expected to be 1023. The value should decrease to zero by clocking ones in to the incoming register. At this point, 1024 zeros are again clocked in to the incoming register and the output should increase to 1023 again. The result is plotted in Fig. 6.3 together with the expected output. This is an important result and several conclusions about the circuit can be made. First of all, almost all of the first 1024 cross-correlations are correct. All the equal bits are rightmost in the the bubble register, keeping in mind that the bubble register sorts ones to the right. Therefore, no sorting has to be done. The thermometer coded output is encoded to the correct binary value and the value is clocked out through the SPI register.

In the next phase, the equal bits are leftmost in the register and have to be sorted. This fails in most cases, but results in some correct single values, a couple of threshold levels and a row of correct values. This may just be enough to get some useful results from the chip and will be tested later.

This test reveals that the encoder seems to work as expected and that something went wrong with the bubble sort.

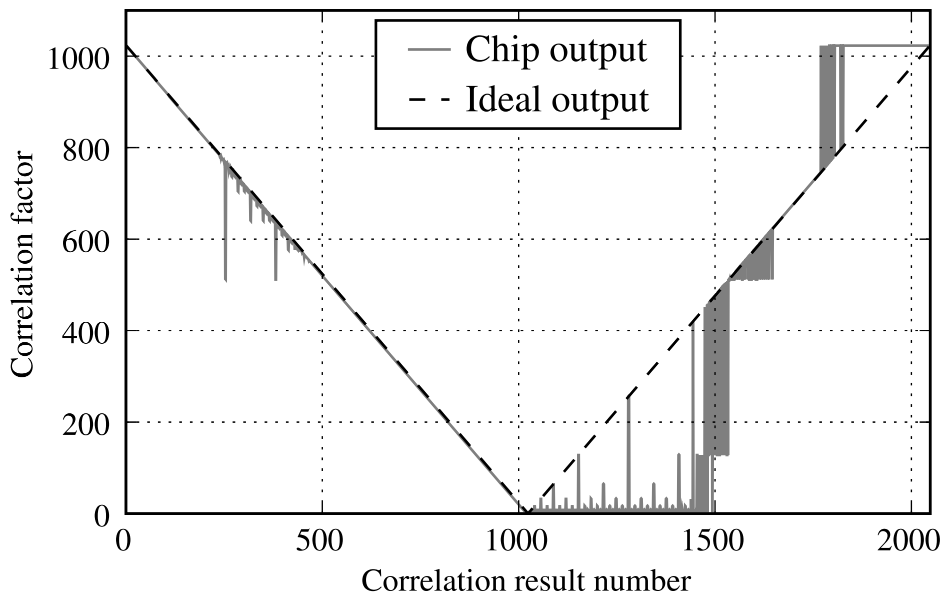


Figure 6.3: The faulty cross-correlation of 1024 bit.

Bubble out enable

A pin on the chip makes it possible to bubble out the content of the bubble register. Debugging asynchronous circuits was previously identified as a challenge, because it always runs at full speed. The minimal theoretical time between two bubbles was simulated to be 144 ps $\approx$ 7 GHz and the available measurement instruments are not adequate for such high frequencies. Ignoring these facts, measurements show a trace of varying widths as the input varies. Unfortunately, the resolution is far from high enough to conclude if the data is distorted or not at this point.

Shifting a constant number of bits through the bubble register

If a constant number of bits in the template and the incoming register are equal, the output should stay constant. By clocking data into the incoming register, satisfying constant equality in the registers, the number of times a bit has to be bubbled will decrease. Fig. 6.4 shows the result from shifting two bits through the register. Clearly something goes wrong when one or several bits have to be bubbled through the entire bubble register. The figure verifies that the output is correct when the number of bub-

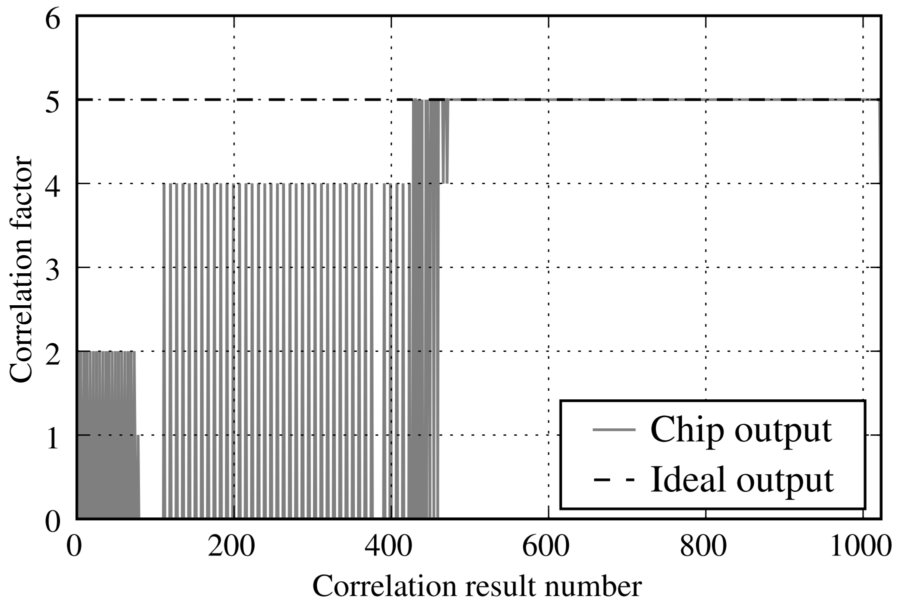


Figure 6.4: Faulty shifting of two bits through the bubble register.

bles are low enough, and in this case less than 552 times of bubbling. Similar results are obtained when sorting a higher number of bits. Remark that the correlation factor seems to be offset by three.

There could be several sources of errors, and can potentially be differences in the pull up/down transistors or in the power supply. If the pull down transistors are stronger than the pull up transistors, the width of the spike being sorted will become narrower for each bubble, eventually disappearing completely. If the pull up is stronger, it could result in spikes being merged together. Both cases will end in missing data.
Large current draw at the clock edge was a concern at design time. The bubble register was initially 512 bits long, but unrealized space on the die and the fact that a new circuit layout could be automatically generated in seconds, resulted in a 1024 bit bubble register. Adjustments, such as wider conductors and dedicated power supplies, were implemented to provide the extra current needed.

In elucidation of the design flaw revealed in section 5.1.4, the most likely reason for the unexpected behavior is a slow falling transition in the bubble sort start tick. The first latches in the bubble register start to bubble before the next latch is ready. The result in Fig. 6.4 indicates that the bubble enable signal transition is to slow in almost half the bubble register. The transition seems to get faster than the bubble operations after this point, which gives hope that the last part of the bubble register functions as intended.

There is no straight forward hardware solution to only utilize the last part of the bubble register, but the μc can contribute to successful on-chip cross-correlation at the cost of speed.

6.2.3 Exploiting the functioning part of the bubble register

After numerous lost battles, a solution was found. The working part of the circuit can be used with help from the μc . The bubble register is now constrained to 512 bits, even though the working part of the bubble register is somewhat higher. This gives a PCM output of 9 bits. To utilize the second half of the bubble register, the first half can not contribute. Different values in the template and the incoming register provides that the bubble register is preset to zero, meaning no contribution. It is not possible to set any

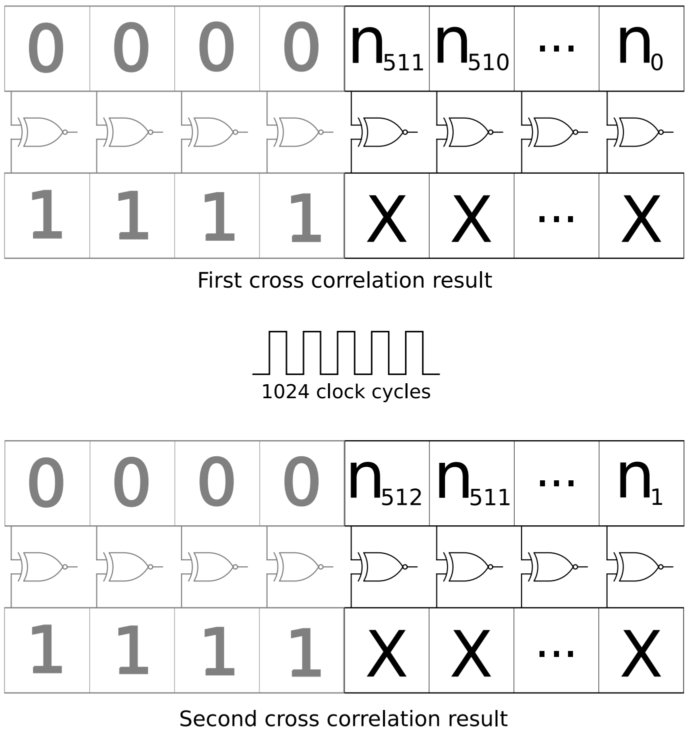

0 0 0 0 n_{511} n_{510} ··· n_0
1 1 1 1 X X ··· X
First cross correlation result
1024 clock cycles
0 0 0 0 n_{512} n_{511} ··· n_1
1 1 1 1 X X ··· X
Second cross correlation result

Figure 6.5: The functioning part of the bubble register can be used by storing different values in the first part of the template and incoming register, showed in gray.

other than the first bit in the template and the incoming register. This means that a 1024 bit sequence has to be shifted through the incoming register for every new bit and that the bitstream in the incoming register has to be stored on the μc . The chip template now has to consist of the actual 512 bit long template and 512 ones, requiring the incoming bitstream to be zero padded. The gray part in Fig. 6.5 illustrates the non-contributing part of the register, while the rest of the figure represents the functioning part of the cross-correlator. The lower part of the figure shows the bit pattern to be shifted into the incoming register to obtain the next result.

The solution is coded in the μc in such a way that the computer operations are the same as before. The only differences are reduced speed and template length. The two previously presented tests are rerun to confirm the bug fix.

Bugfixed; from maximum cross-correlation to zero to maximum again

The script on the computer sets the template length to 512 on the μc . This enables the μc to shift in the data to the on-chip incoming register as described in Fig, 6.5. The expected chip output is a cross-correlation result of 512 decreasing to 0, before increasing to 512 again. The Fig. 6.6 shows the chip output, which this time corresponds to the expected ideal output.

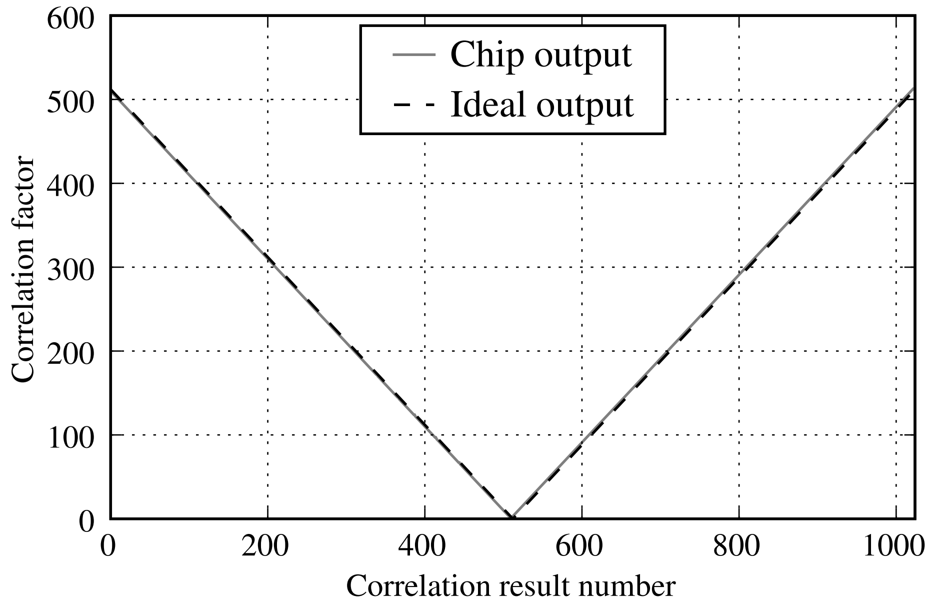


Figure 6.6: The correct cross-correlation of 512 bit.

Bugfixed; shifting a constant number of bits through the bubble register

Fig. 6.7 confirms the shifting of two bits through the register, this time with an offset of two. This offset seems to be constant when the input is constant, but lsb errors occur in other cases and will not be corrected. This test reports the cross-correlator fit for duty

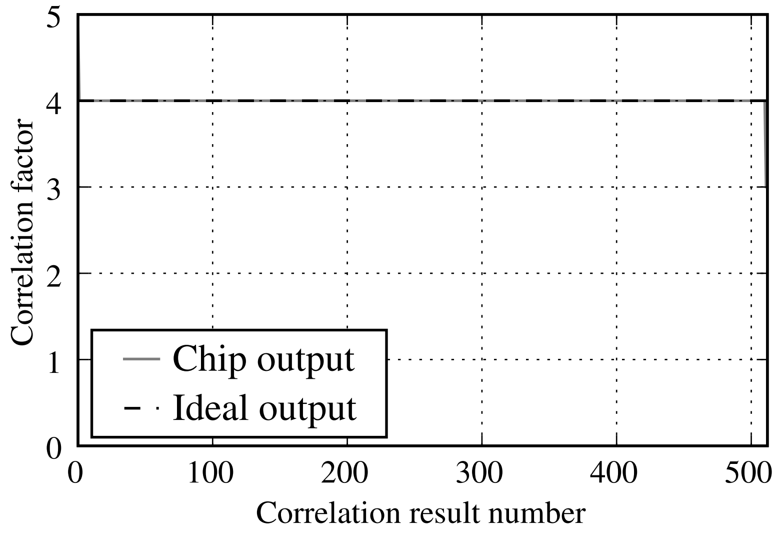


Figure 6.7: Correctly shifting two bits through the bubble register.

and the originally planned tests can take place.

6.3 Bitstream cross-correlation

The cross-correlation of two sinusoids will be the first chip benchmark. The chip bitstream cross-correlation result is compared to the software cross-correlation on the original PCM coded sinusoid, which gives a good approximation of the bitstream cross-correlation SNR.

The next chip test shows how a pattern is recognized in a noisy and distorted signal.

6.3.1 Two sinusoids

A sinusoid with a frequency of 1 Hz is converted to a bitstream in software. The template register is preset with one period of the sinusoid and the cross-correlator is fed with the data. The raw chip output is plotted in Fig. 6.8. The signal is yet to be

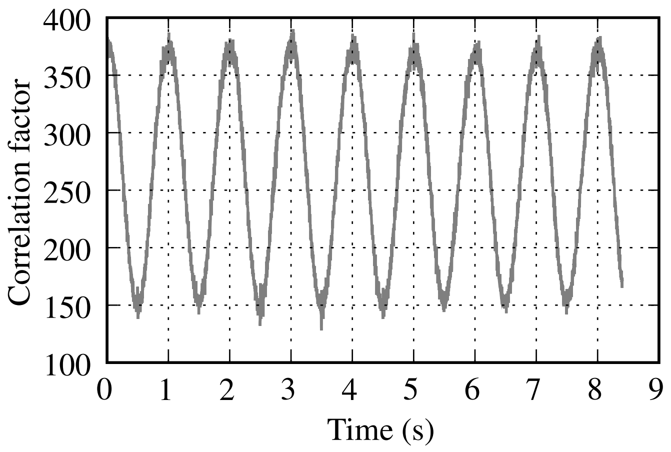


Figure 6.8: Raw chip output.

decimated, high frequency noise has to be attenuated before comparing it to the ideal cross-correlation result. The y axis is the actual equal number of bits in the signal and the template, where the value 512 indicates two completely equal signals.

The bitstream and ideal cross-correlation results of two sinusoids are plotted in Fig. 6.9. The "Ideal output" is the mathematically correct cross-correlation between the two original high resolution PCM coded signals. Both cross-correlation results are normalized. The 4th peak is enlarged to show the cross correlation error when the input signal is changing rapidly.

The error e of the bitstream cross-correlation is found by subtracting it from the ideal version of the result. Fig. 6.10 shows that the worst case error is approximately -23 dB. The SNR is found to be

$$SNR = 20 \log 10 \left(\frac{A_{ideal}}{A_e} \right) \approx 25.8 \text{ dB} \tag{6.1}$$

where A is the Root Mean Square (RMS) value of the signal x of length n given by

$$A = \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} \tag{6.2}$$

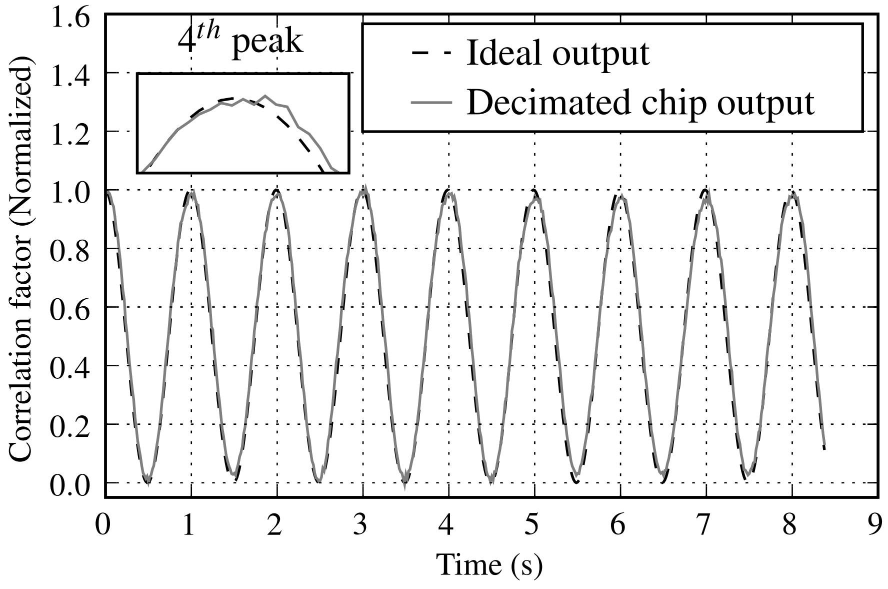


Figure 6.9: Cross-correlation results of two sinusoids.

The SQNR of a second-order $\Delta\Sigma$ modulator was calculated in subsection 3.1.2 to be 34 dB and the SNR of the cross-correlation seems poor. The template used was one period of a sinusoid, created to fill the template register. This means that the input signal has a small change in the amplitude between each sample and more noise than expected is introduced. The SNR of the modulated signal is 20 dB, the cross-correlation result has less noise than the bitstream itself.

To confirm that the cross-correlation does not contribute any noise, a new sinusoid was created with the correct OSR and bitstream cross-correlated with a longer version of itself. The Fast Fourier Transform (FFT) of the decimated result, plotted in Fig. 6.11, shows that the noise components are attenuated with 33 dB, which corresponds to the expected noise in a second-order $\Delta\Sigma$ modulator.

The results confirm the theoretical bitstream cross-correlation calculations derived in section 3.3; the bitstream cross-correlation does not contribute significant noise.

A lesson learned is the importance of using the correct OSR when modulating the bitstreams, especially when the input is a noiseless sinusoid. The sinus with the higher sampling rate leads to in-band idle tones and a lower SNR was achieved. The length of the template register is fixed and the OSR of eight can only be complied for a lim-

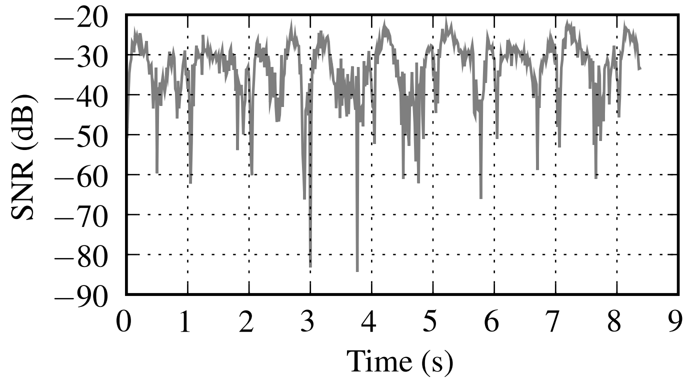


Figure 6.10: The error between bitstream and ideal cross-correlation.

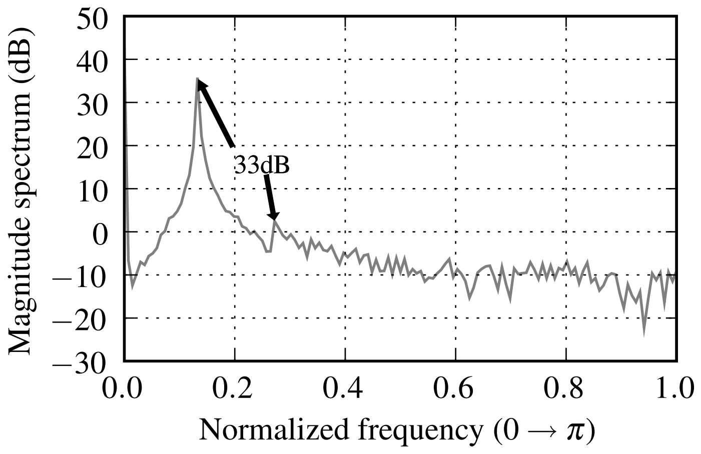


Figure 6.11: FFT of the decimated bitstream cross-correlation result.

ited set of templates. Templates used in further tests of the chip will be modulated at the sampling rate that leads to a template length of 512 bits. A possible lower SNR is accepted in exchange for real world examples.

Decimation

A first order decimation filter written in software is used for the low pass filtering of the signal. OSR samples are simply summed and averaged to form the Nyquist rate result. Section 3.1.3, "Decimation", stated that the bitstream from a modulator of order L should be decimated with a filter of order $(L + 1)$. First order filtering of the output from the second order modulator would have given a coarse result. First order filtering of the chip output gives a good result because of the cross-correlation operation. A higher order decimator gives a slightly smoother result. The cross-correlation operation seems to attenuate the high frequency noise, which is evident on the signals FFT. The frequency response of a bitstream and a bitstream cross-correlation result is plotted in Fig. 6.12.

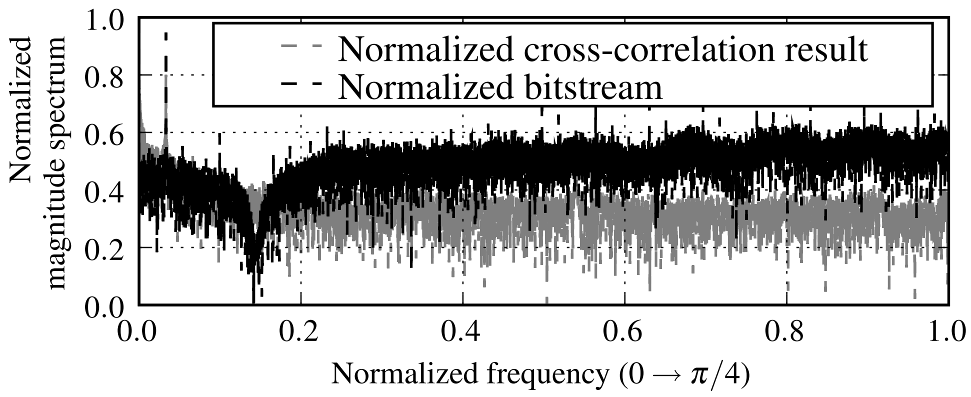


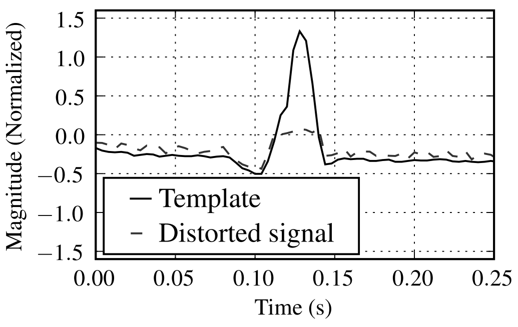
Figure 6.12: The FFT of a bitstream and a bitstream cross-correlation result.

6.3.2 Pattern recognition

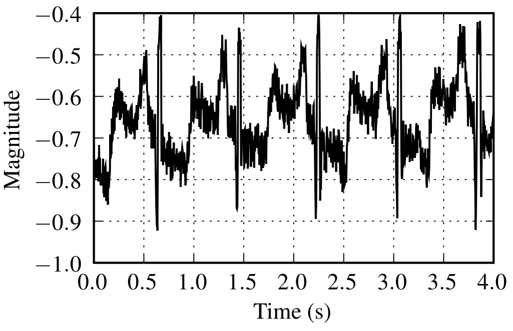
One application of cross-correlation is pattern recognition, e.g. to identify a specific pattern received over a communication channel. The specific pattern is stored in a template and cross-correlated with the received signal. The template can be identified in the signal even if the signal is, to a certain extent, distorted and noisy. The amplitude of each sample in the cross-correlation signal is a measure of how much the received signal resembles the template signal at that location.

Consider the template and signal plotted in Fig. 6.13. The template is derived from the signal before distortion and is the spike, without noise, around 0.6 s in Fig. 6.13(b). Comparing the template and the signal reveals that the signal amplitude has been cut and white noise is added. This template and signal will be the first real world test for the bitstream running cross-correlator.

Both signal and template are modulated to bitstreams in software and fed to the cross-correlator, the raw chip output is plotted in Fig. 6.14. The raw chip output is very noisy and it is hard to imagine that the cross-correlation has identified the searched pattern. The first order decimated result is plotted in Fig. 6.15. The PCM time domain



(a) A generic template.



(b) A signal received over a noisy channel.

Figure 6.13: The pattern (a) to be recognized in the signal (b).

representation of the two signals were cross-correlated in software, named *Ideal output* in the figure, for comparison.

The pattern searched for is located as evident spikes and the chip result is close to the ideal cross-correlation. Cross-correlation of the none noise version of the input signal is performed in chapter 7.

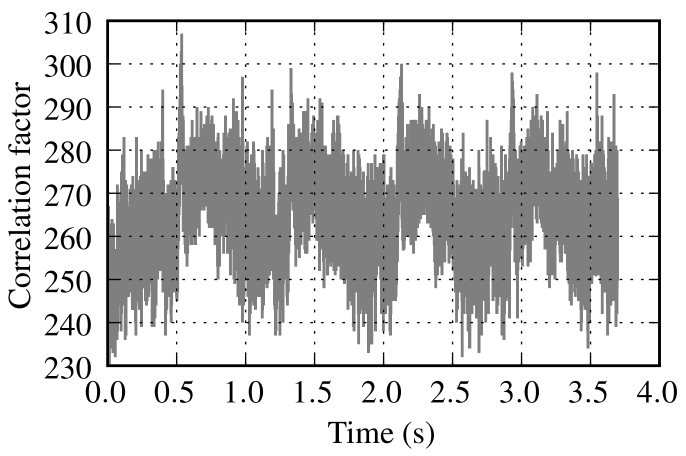


Figure 6.14: Raw chip output.

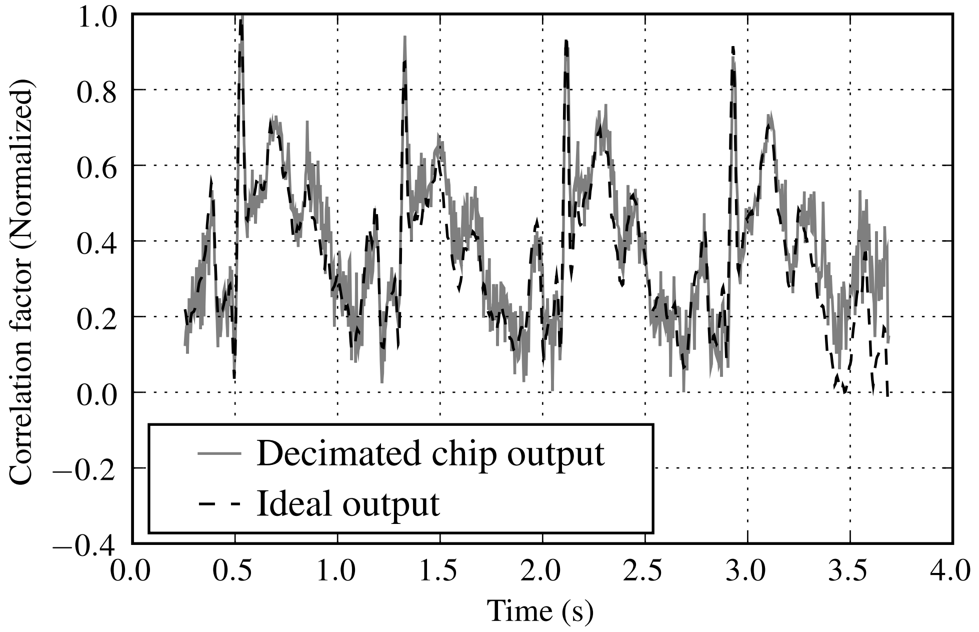


Figure 6.15: Cross-correlation result of the template and signal in Fig. 6.13.

6.4 Power consumption

Measuring pins on the PCB were intended to provide an easy method to measure the current drawn by the on-chip cross-correlator. This measuring scheme turned out to introduce enough noise on the supply net to corrupt the cross-correlation result. An improvised solution which included scraping off conducting lines on the PCB, soldering and even gluing made it possible to measure the current before the voltage regulator. Subtracting the current used by the voltage regulator gives the current drawn at the 1 V power supply.

The cross-correlation power dissipation depends on the clock speed, which is controlled by the μc SPI clock. Each package of data is sent over the SPI at this speed, but is reduced if the μc has other tasks to accomplish, i.e. write results to the Universal Serial Bus (USB) buffer. A special μc method was written to ensure the correct cross-correlation clock frequency, the actual clock frequency was confirmed with an oscilloscope.

The largest measuring uncertainty, after securing a stable clock frequency, is how the current drawn is affected by the non functioning part of the bubble register. A rerun of the test where the cross-correlation output was expected to decrease from 1023 to 0 and then increase to 1023 again, while measuring the current, reveals that the dissipated power is almost proportional to the expected cross-correlation result. This indicates that measured power dissipation is a good approximation to the power dissipated by the entire 1024 bit cross-correlator.

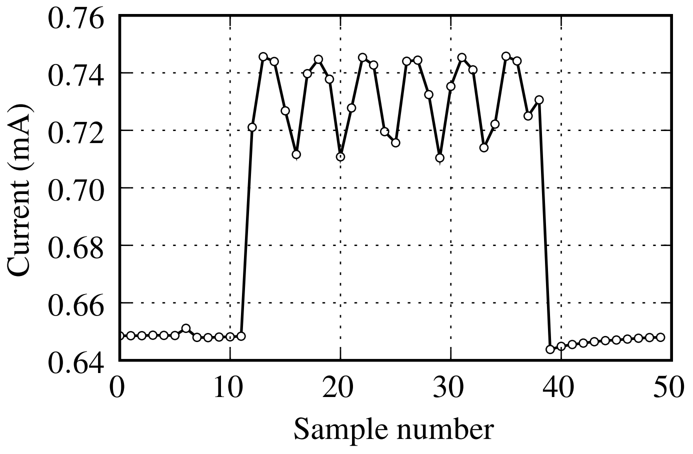


Figure 6.16: Current consumption at *clk_hi* = 18.8 kHz.

The lowest possible SPI clock frequency is 188 kHz, resulting in the internal clock frequency, *clk_hi* = 18.8 kHz. A digital multimeter is connected to the computer and measures the current drawn at constant intervals. A large capacitance is connected between the input to the regulator and GND to even out the current drawn from the power supply over time. The script measuring current is started before the cross-correlator is fed with a sequence resulting in a output from 1023–0–1023 over and over at a constant frequency. The voltage supply is 1 V. The measured current is plotted in Fig. 6.16. The small increase in power consumption at the sixth sample is caused by setting the two bitstream registers. The consumed power is proportional to number of equal bits in the two registers, which confirms that the power consumption is signal dependent. The static power consumption is high compared to the dynamic at this operation speed, $P_{static} = 0.65$ mW. The mean power consumption while operating the cross-correlator is $P = 0.73$ mW. This gives $P_{dynamic} = P - P_{static} = 0.08$ mW when *clk_hi* = 18.8 kHz. The overall power consumption was found to be the sum of the constant static power and the dynamic power which was dependent on the number of gates actively switching.

The same measuring scheme is repeated, doubling the clock frequency each time. The results are summarized in Fig. 6.17.

Frequency (kHz)	P (mW)
18.8	0.73
37.5	0.85
75.0	1.1
150.0	1.5
300.0	2.3
600.0	3.4
960.0	5.1

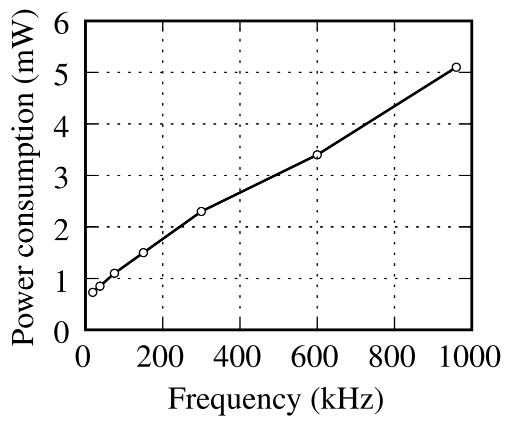


Figure 6.17: Measured mean power consumption at different frequencies.

The power consumption is as expected linear to the clock frequency. Comparing the power consumption presented in the paper in Appendix A with Fig. 6.17 illustrates an important advantage of asynchronous systems, but an issue regarding performance analysis. The power dissipation presented in the paper is measured when processing real data and is somewhat lower than the power dissipated during this measuring scheme. The measurements from this section attempts to cover the average power dissipation at an average computational load which gives a good basis for comparison.

6.5 Performance

Different properties of the bitstream cross-correlator are discussed and compared to similar available hardware solutions. The bitstream cross-correlator efficiency depends on an available bitstream with a low OSR and a decent resolution. This can be the case in an application where a sensed analog signal is modulated and the bitstream is extracted before decimation. The bitstream cross-correlator is compared to an 8-bit μc . A bitstream with a resolution of 8 bits is hard to create with a second-order modulator with such low OSR. All input and outputs of the bitstream cross-correlator are assumed to be oversampled in the following comparisons.

6.5.1 Power performance

The implemented circuit is intended to be a proof-of-concept and no power considerations were taken at design time. The efficiency of the bitstream cross-correlator is compared to a high performance, low power, 8-bit μc , namely the Atmel ATmega48PA. The data of interest are extracted from the ATmega48PA data sheet and summarized in table 6.2. The power consumption numbers for the ATmega48PA are measured with all peripherals turned off, even the clock lines to those units are disabled, and is the typical power consumption at the given speed and supply voltage.

Table 6.2: ATmega48PA properties of interest.

Power consumption	20 MHz, 5.5 V	57 mW
Power consumption	15 MHz, 4 V	20 mW
Throughput		Up to 16 MIPS at 16 MHz
Load indirect		2 clock cycles
Multiply unsigned		2 clock cycles
Add two registers		1 clock cycle
Branch if not equal		1 / 2 clock cycles (2 if true)
Relative jump		2 clock cycles
Increment		1 clock cycle

The bitstream cross-correlator Nyquist rate multiplications per second were in Eq. 5.1 found to be $f_{clk\_hi} \cdot n/OSR^2 = 16 f_{clk\_hi}$. A clock rate of 960 kHz yields 15.36 M multiplications each second with a power consumption of 5.1 mW. μc instructions for multiplying in a loop could be:

`L1:`

Multiply data	2 clock cycles
Relative jump `L1`	2 clock cycles
One multiplication loop	4 clock cycles

This requires that the data is written and read to the correct registers in between μc operations. The μc power consumption is not linear to the clock frequency, a frequency of 15 MHz is chosen for this comparison, revealing multiplication results four times slower than the bitstream multiplications. ATmega48PA performs 3.75 M multiplications each second with a power consumption of 20 mW. The Figure Of Merit for this comparison is given by

$$FOM = \text{(number of operations)}/\text{power dissipated.} \tag{6.3}$$

This FOM is not accurately when comparing calculations at different clock frequencies. Table 6.2 shows that an increase in operation speed requires an increase in the power supply voltage, thus dissipating more power per operation. The chosen FOM is easily extractable from the μc 's data sheet and gives an rough underestimate at different operation speeds. The power dissipation profit using bitstream multiplications is then given by:

$$FOM_B/FOM_{\mu c} = \frac{15.36\text{ M}}{5.1\text{ mW}} / \frac{3.75\text{ M}}{20\text{ mW}} \approx 16 \tag{6.4}$$

The bitstream cross-correlator seems in this case to use almost 16 times less power than a low power, multiplication effective μc .

To complete the cross-correlation, the multiplication results have to be summed. The bitstream cross-correlator sums the $1024/8 = 128$ multiplication results in the cross-correlation window asynchronously. The Nyquist rate cross-correlations per second were in Eq. 5.2 found to be $f_{clk\_hi}/OSR = f_{clk\_hi}/8$. A clock rate of 960 kHz yields 120 k cross-correlations each second with a power consumption of 5.1 mW. μc instructions for cross-correlating in a loop could be:

`L1:`	
Reset result register	$2 \cdot 1$ clock cycles
Reset inner counter	1 clock cycle
`L2:`	
Load template	2 clock cycles
Increment counter	1 clock cycle
Multiply data	2 clock cycles
Accumulate 16-bit result	$2 \cdot 1$ clock cycles
Compare counter	1 clock cycle
Branch not equal `L2`	2 clock cycles
Relative jump `L1`	2 clock cycles
Inner loop	10 clock cycles
Outer loop	5 clock cycles

To complete one cross-correlation result, the inner loop, `L2`, has to execute 128 times and the outer loop,`L1`, one time, resulting in 1285 clock cycles per cross correlation result. To match the bitstream cross-correlator speed, the μc has to run at 154 MHz which is not possible. The highest clock frequency is 20 MHz, dissipating 57 mW, resulting in almost 16 k cross-correlations per second. The power dissipation profit using bitstream cross-correlation is then given by:

$$FOM_B / FOM_{\mu c} = \frac{120\text{ k}}{5.1\text{ mW}} / \frac{16\text{ k}}{57\text{ mW}} \approx 84 \tag{6.5}$$

The bitstream cross-correlator is designed to perform cross-correlation, while the μc is a generic platform capable to perform any calculation. The result is promising and the implemented chip is worthy to be compared to a more specific Nyquist rate cross-correlator, which is not done in this thesis. Minimizing the power consumption was not an issue at design time. The area used for the cross-correlator could almost be halved by using standard cells provided by the factory. A halved area will possibly halve the gate capacitance which will have a great effect on both static and dynamic power dissipation.

6.5.2 Speed performance

The upper speed limit for the bitstream cross-correlator in this implementation, seems to be around 960 kHz. The previous section stated that a μc can perform the same computational load with a clock frequency of 143 MHz. The μc has a maximum clock frequency of 20 MHz and can maximum compute the cross-correlation result 7.5 times slower than the bitstream cross-correlator.

If the speed limit on the implemented chip is the bubble time, the statistical simulation of the bubbling would be 40σ from the actual delay. The most probable reason for the speed limit is the high current drawn at the clock edges. Large clock buffers cause noise at the power supply even at low frequencies. Again, the use of standard cells would have improved operation. The many long registers in the design require large clock buffers, but a more clever solution can probably spread the current drawn out in time.

The acquired clock frequency is most likely not the limitation when processing real-time data, the register length confines the template length. The maximum Nyquist rate signal is given by $960\text{ kHz}/8 = 120\text{ kHz}$, this gives a template length of 1 ms. A more applicable Nyquist rate could be 250 Hz, giving room for a template of more than 0.5 s. This sample rate is suitable when sampling heartbeats, which enables the implemented bitstream cross-correlator to be suitable for heartbeat detection. More on this topic in the next chapter.

6.5.3 Resolution efficiency

Empirical and theoretical computations indicate that the resolution is not depreciated by the bitstream cross-correlation. Noise in the cross-correlation result is mainly modulation noise. Since the cross-correlation result is the sum of equal bits from the XNOR stage, the output is restricted to 10 bits before decimation which is considerably less than the 16 bits result computed by the μc . The output resolution is higher than the resolution of the incoming bitstream.

6.6 Suggested improvements

The simplest improvement of the bitstream cross-correlator, which will involve great improvements in terms of power savings and silicon area, is customizing of basic building blocks of the circuit. All building blocks in the implemented design are created from NAND and inverters, without adapting of transistor dimensions. The main power consumers in the circuit are probably the three 1024 bit registers with the belonging clock buffers. Other solutions for the registers are tempting, maybe one of the asynchronous shift registers summarized in [Dobk 06] are employable, approaching a fully asynchronous system. The last main synchronous element in the circuit would then be the time allowed for the bubble sort. The whole operation depends on the time it takes for the bubble sort to conclude, which is somewhere between instantaneous and the worst case bubble time. A single zero-one transition in the thermometer code can only occur when the bubble register is stable, the circuit is then ready to read to start the next computation.

The Muller pipeline discussed in section 2.1.3 resembles the bubble register in behavior and appearance. A short glance at how the Muller C-element can be used in bubble sort is given.

6.6.1 Bubble sort register using Muller C-elements

The symmetric implementation of the Muller C-element in [Sham 98] was created and the transistor widths were dimensioned as a trade-off between propagation delay and power consumption. The layout area occupied by the element is equal to the latches in the actual implementation. Smaller dimensions could have resulted in a slower, but less power dissipating solution. One bit in the bubble register is realized from one Muller C-element, one AND gate and one inverter. A Monte Carlo simulation confirms the operation and the mean bubble time is estimated to 116 ps, almost 20% shorter than the simulated mean time of the original design.

The Muller C-element relaxes the previously discussed τ_3 and is a well known element in asynchronous logic. The example also touches how speed, power efficiency and layout area should be customized for the planned implementation.

7 Electrocardiography beat detection using cross-correlation

Cross-correlation is a generic pattern-matching computational element suited for several signal processing tasks. The bitstream running cross-correlation solution presented in this thesis aims at low power operation at low or moderate signal frequencies. The convolution possibilities of the chip increases its usability to include filter operations. The convolution filter is programmable and can e.g. be adapted to the specific input data.

Many of these applications are emerging with biomedical applications and one area of application is in the field of beat detection in ECG analysis.

7.1 Beat detection in Electrocardiography

Cross-correlation can be used for heartbeat detection in an ECG analysis. An introduction to beat detection and ECG is given before the implemented cross-correlator is tested using one of the most promising beat detection approaches.

7.1.1 Electrocardiography

Electricity is what makes the heart beat. ECG is the recording of the electrical activity of the heart and it is possible to diagnose many cardiac disorders through perturbations in these recordings. The standard ECG consists of 12 leads, recorded by electrodes placed on both arms and legs, and across the chest. Each lead views the heart at an unique angle, facilitating sensitivity to a particular region of the heart. Arrhythmia refers to any disturbance in the rate, regularity, site of origin, or conduction of the cardiac electrical impulse. Single abnormal beats frequently occur in the majority of healthy individuals, but many arrhythmias can be dangerous and some require immediate therapy. ECG is today the most important tool for diagnosis of arrhythmia and a long term ECG is often needed to identify such irregularities. The Holter monitor provides long tracing of one or more leads, where the leads that provide the most information are chosen. The recordings are stored and later analyzed.

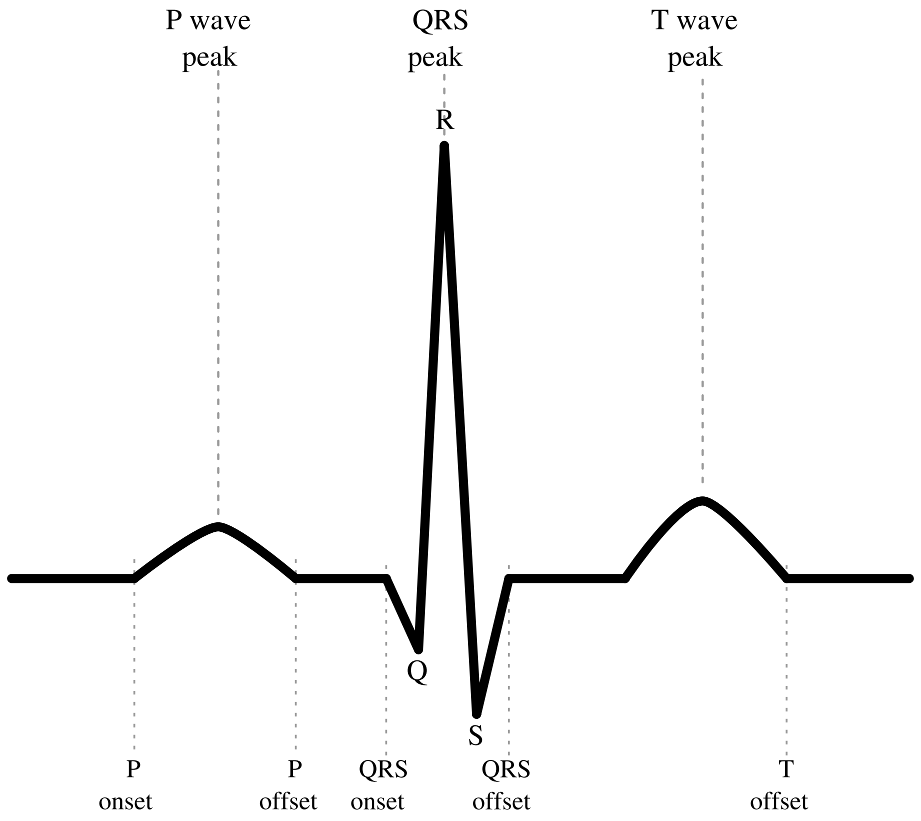


Figure 7.1: Schematic representation of a normal ECG including P wave, QRS complex and T wave beat markers.

The electrical activity from one cycle of cardiac contraction and relaxation is schematically drawn in Fig. 7.1. Not all electrical events of the heart are evident on an ECG. The P wave is caused by contraction, or depolarization, in the heart's prechambers, atrial contraction. The next electrical activity is called the QRS complex and the precise configuration of these events can vary greatly. The first upward deflection is more precisely called an R wave and is caused by the contraction of the left and right ventricles, the two main heart chambers. The T wave is caused by relaxation, or repolarization, of the ventricles.

The intervals and segments between events are given names and different irregularities can be identified by the length of these intervals. Examples are the QT interval, from QRS onset including the T wave, and ST segment, from QRS offset to the start of the T wave.

The QT database

A number of automated methods for measuring intervals and segments in the cardiac cycle have been designed. The lack of standardized databases containing a sufficient large number of manually annotated heartbeats, have prevented good evaluation of the performance of such algorithms. Tremendous effort from clinicians is required to manually annotate a statistically significant set of ECG records. The QT Database [Lagu 97] includes ECGs to challenge beat detection algorithms with real-world variability. Cardiologists have manually annotated sections from each record in the database. The collection of data contains a total of 105 fifteen-minute excerpts of two leads ECGs, more than 50 hours of recordings.

Software to extract the heartbeat records and the annotation files is available online, but a Python wrapper script was created for easier access. An excerpt of this script can be found in Appendix E. The Fig. 7.2 shows one cardiac cycle with manually annotated markers.

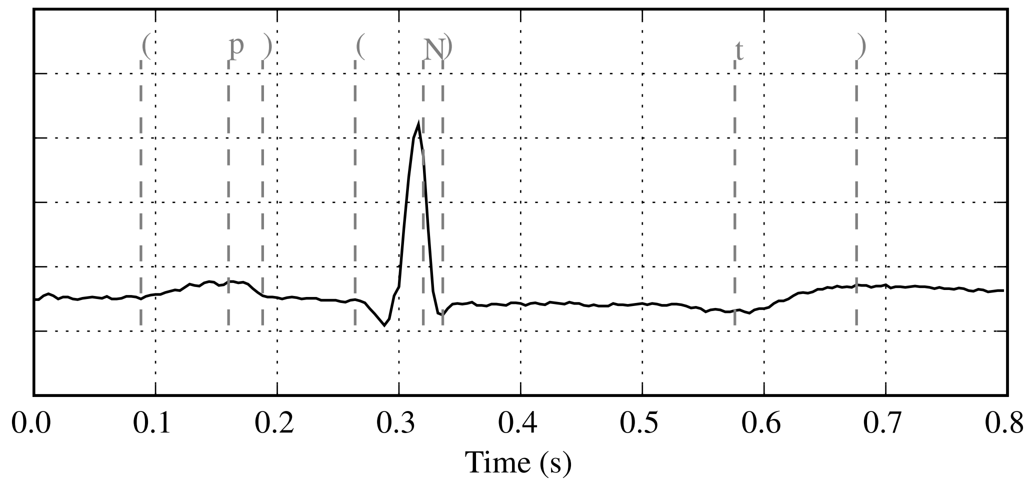


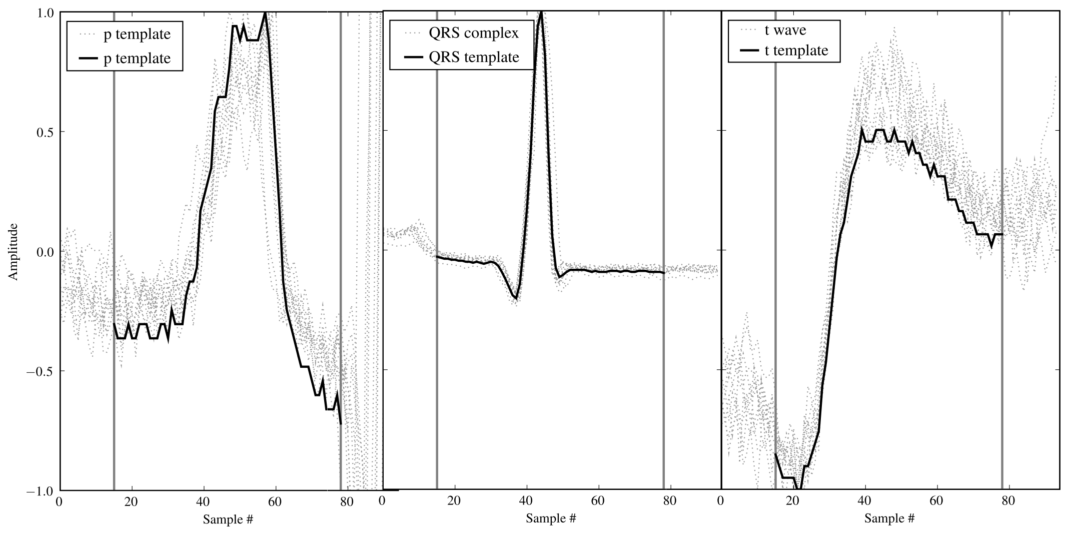
Figure 7.2: One heartbeat with with manually annotated markers.

7.1.2 Beat detection

Beat detection involves identifying all cardiac cycles in ECG recordings and locating each identifiable waveform component within a cycle, the P wave peak, the QRS peak and the T wave peak, including the belonging on- and offset values. The accuracy of the beat detection has significant impact on the overall classification performance and various computerized methods have been developed to meet the minimum error values that should be expected with any automatic algorithm. One method that does not meet the requirements involves the cross-correlation between a template representing one generic heartbeat and the heartbeats of a patient.

Multi-component based beat detection

An improvement of the traditional cross-correlation method is proposed in [Last 04], where each of the identifiable waveform components within a cycle are searched for in isolation. Three templates were used, one for the P wave, one for the QRS complex and one for the T wave. The first step is to locate the QRS complex by cross-correlating the QRS template with the ECG signal. This method is repeated with the P ant T wave templates. In each case, the highest value of the cross-correlation result within the given correlation interval is compared to a predefined threshold. The threshold value is established during a pre-learning phase and can be adjusted during operation. The on- and offset values are generated automatically based on the used templates.



(a) P wave template. (b) QRS wave template. (c) T wave template.

Figure 7.3: The three templates.

There are at least two drawbacks to the following bitstream multi-component based beat detection analysis. 1) The $OSR = 8$ which restrains the template length to 64

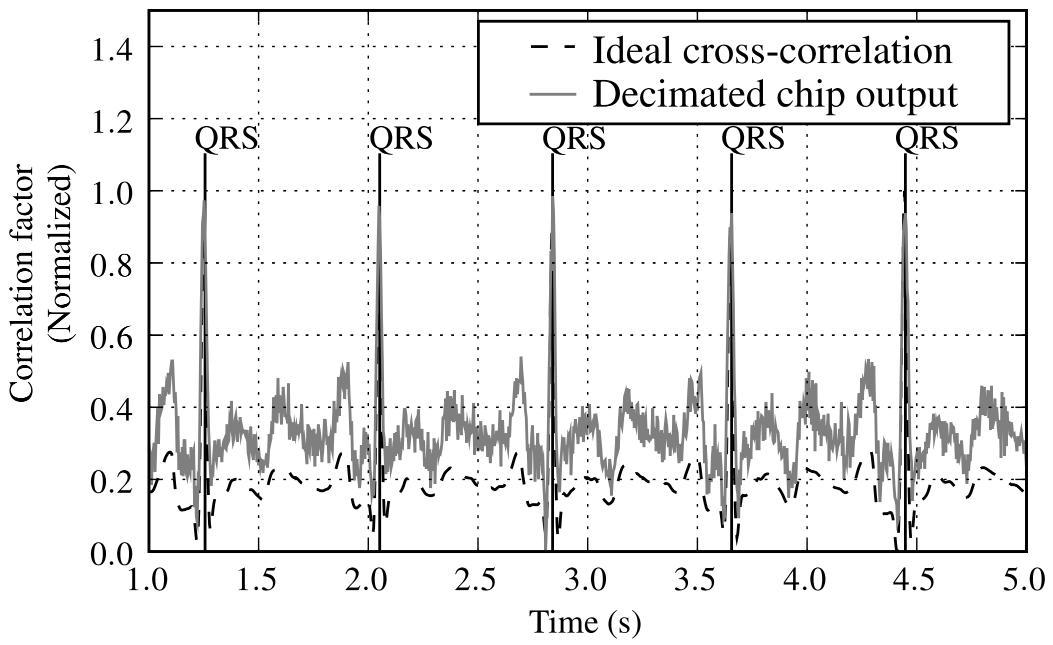


Figure 7.4: Cross-correlation results of the QRS complex.

samples. 2) The fact that finding and generating the most advantageous template is a difficult and time consuming task. In addition, the template and ECG signal have to be scaled before modulation, giving even more alternatives to create the best combination. Each template was created by summing and averaging the ten first identifiable components of each type. Fig. 7.3 shows the three templates and the data which they were generated from. The vertical lines show the boundaries of the template, 64 samples wide. The templates are scaled so that the mean of each template is zero. The ECG signal is scaled to from -1 to 1 to span the modulator dynamic range. This also gives the most similar result compared to cross-correlation of the PCM coded signals in software.

The first objective is to localize the QRS complex. The decimated chip result is plotted in Fig. 7.4 together with the ideal cross-correlation. Again, "ideal" is the mathematical correct cross-correlation between the two original high resolution PCM coded signals. The label "QRS" and the belonging line shows the markers inserted by clinical experts. Solely by inspection the bitstream cross-correlation results are promising and very close to the "ideal" result.

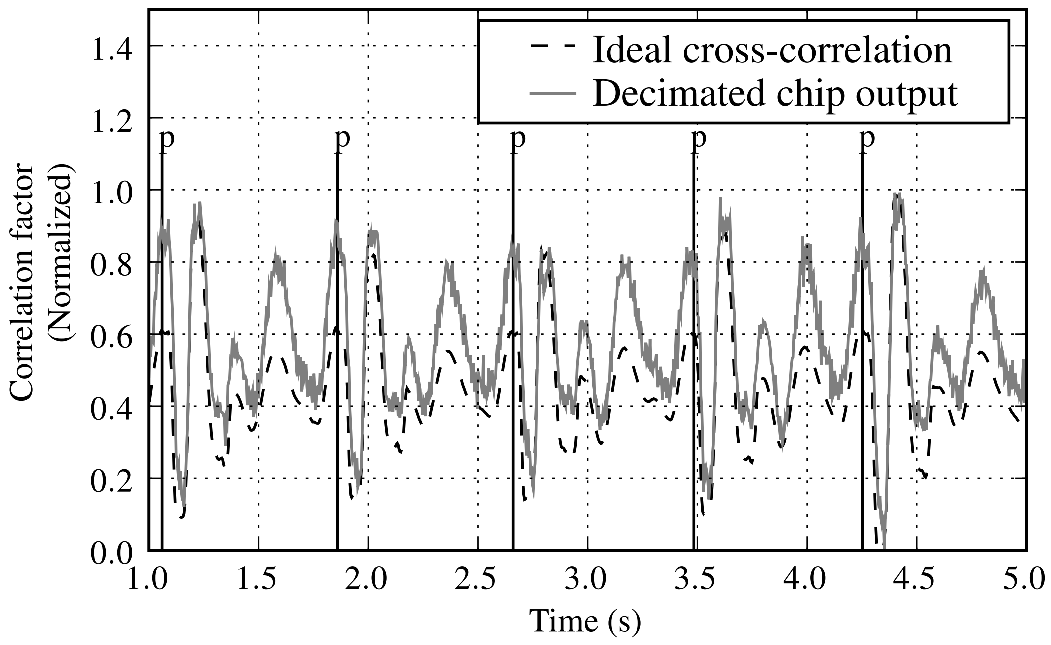


Figure 7.5: Cross-correlation results of the P wave.

The next component to detect is the P wave, which historically is the most difficult wave to detect. Fig. 7.5 shows that the P wave is not exclusively found. Both the T wave and QRS complex gives large peaks, which makes the localization of this wave a challenge. Problems detecting the P wave are also reported in [Last 04], but not to this extent. More important is that the P wave gives a higher peak in the bitstream cross-correlation result, which may be the result of scaling before modulating the signal. A equal scaling in the "ideal" case, seems to give poorer results. A better template would probably have given a better result.

The last wave to localize is the T wave. Fig. 7.6 shows again that the bitstream cross-correlation gives a better result than the "ideal" case and the T wave can easily be detected by thresholding.

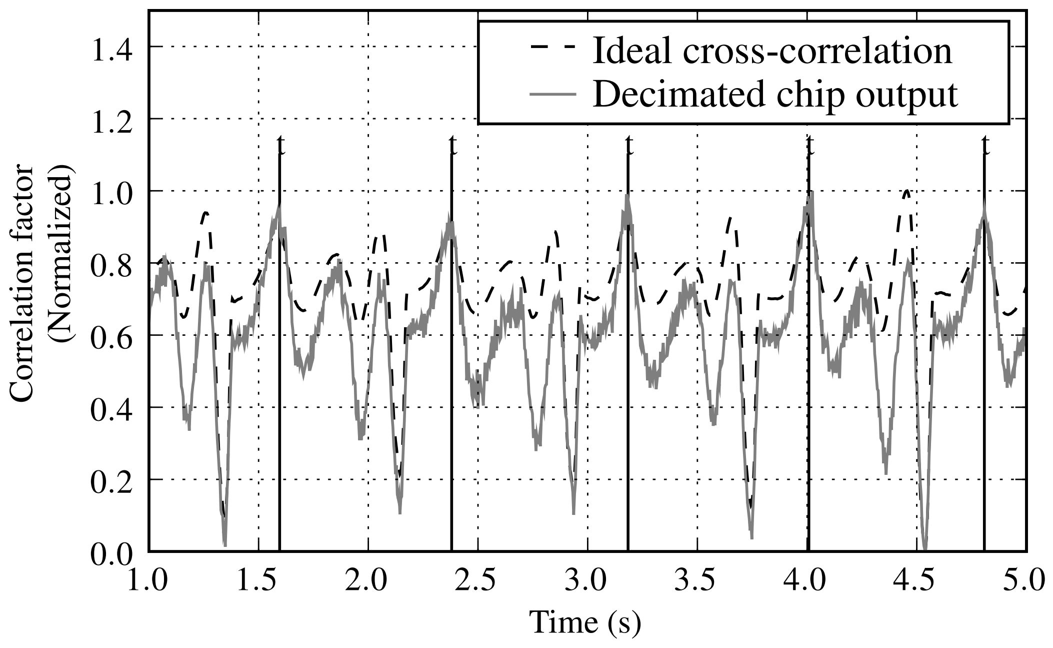


Figure 7.6: Cross-correlation results of the T wave.

7.1.3 Conclusion

This first attempt of bitstream multi-component based beat detection seems promising, but is far from the work performed in [Last 04] where results were statistically compared to other methods. A proper evaluation of the bitstream cross-correlation method for detection of components in an ECG signal demands extensive analysis, unfortunately beyond the scope of this thesis. The accomplished results seem promising. Some of the bitstream cross-correlation results were better than expected, which probably is caused by different scaling of the bitstream and PCM signal around zero.

8 Conclusion

A single-chip cross-correlator has been presented in this master thesis. The work includes processing signals in their oversampled $\Delta\Sigma$ domain and asynchronous techniques which together result in a novel signal processing solution. Bitstream operations were presented in [Malo 91] and asynchronous principles were derived as early as in [Mull 59]. The ideas are old, but not practiced to a great extent. Combining these two nontraditional techniques lead to a unique proposal to the computation demanding cross-correlation operation.

Bitstream processing profits from simple multiplication operations, complex digital hardware like a multiplier can be substituted by a single gate. A novel asynchronous bubble register, which is found to be surprisingly similar to the asynchronous Muller pipeline, avoids increased clock frequency when summing the multiplication result, and makes the power dissipation proportional to the actual amount of processing done.

The implemented bitstream cross-correlator has proven its usability on low frequency signals and has been demonstrated on signals such as simple sinusoids to more complex real-world examples. The chip has been tested to perform heartbeat detection, a simple comparison between the bitstream and PCM cross-correlation seems promising. The possible utilization as a convolver is exciting. The cross-correlator is measured to mean power consumption of 5.1 mW when performing 120 k cross-correlations each second at Nyquist rate. This is 84 times better than the low power μc ATmega48PA. The chip was a proof-of-concept with a far from optimal layout. The μc is a generic tool and more power efficient cross-correlator architectures exist. A speed and energy performance between peers remain.

The bitstream cross-correlation earnings depend on an already available bitstream coded input signal, modulated at a low Oversampling Ratio. The achieved resolution is mainly dependent on the $\Delta\Sigma$ modulator.

The SKILL implementation of the bitstream cross-correlator/convolver has made the system easily adaptable to a large variety of possible utilizations. Different measurement schemes were set up, and the most advantageous alternative was developed further to perform simple initial tests and heartbeat detection. A design flaw seemed to sabotage the implementation, but a solution was found. The design bug did not compromise testing nor proof-of-concept.

The ideas and concepts behind the implemented chip can hopefully be used in low power, low frequency applications and one possible utilization can be heartbeat detection. Further development of the concept can result in a fully asynchronous system. Extrapolation of the concepts behind bitstream processing and asynchronous operations can potentially make a foundation for a clockless, power efficient and error robust processing unit.

A Paper

The following paper has been submitted to the 35th European Solid-State Circuits Conference (ESSCIRC). Acceptance status is as of this writing not yet known.

Power efficient Cross-correlation using Bitstreams

Olav E. Liseth, Daniel Mo, Håkon A. Hjortland, Tor Sverre “Bassen” Lande and Dag T. Wisland
Dept. of Informatics, University of Oslo, Norway
Email: olaveli@ifi.uio.no, daniemo@student.matnat.uio.no, haakoh@ifi.uio.no, bassen@ifi.uio.no, dagwis@ifi.uio.no

***Abstract*—The fundamental operation of cross-correlating signals is viable in a number of signal processing applications. In typical pattern-matching applications, cross-correlation is desirable. In this paper we present a power efficient implementation of a time-domain cross-correlator suitable for integration in CMOS. Bitstream coding of both data and template simplify multiplication operations. Measured performance of a CMOS implementation in 90 nm technology is reported.**

I. Introduction

A major trend in microelectronics is integration of specialized solutions in an increasing number of applications. The idea of ubiquitous computing is certainly exciting, but at the same time demanding. Both sensing and controlling real world processes demand mixed-mode solutions combined with challenging signal conditioning and processing. The notion of small, portable, battery-operated systems often organized in a wireless sensor network (WSN) has initiated significant research activity. In these applications size is limited and low-power operation is mandatory for battery operation. The benefit of adopting specialized silicon systems is evident in applications like WSN motes [1].

An overall characteristic of these ubiquitous computational devices is mixed-mode operation. Sensing of external states is accomplished with analog-to-digital converters (ADCs) and controlling of external processes requires DACs. A popular and power-efficient converter architecture is sigma-delta, or delta-sigma, converters well suited for integration in digital technology.

A recurring signal processing task in real-world signal analysis is pattern-matching recovering special features of some sensed signal. As an example in this work we will use signal processing for Electrocardiogram (ECG) classification. Although filter-based solutions have been developed with great sophistication over the last decades [2], recent publications [3] indicate that cross-correlation based methods are preferable. Furthermore, the miniaturization of ECG-monitoring devices are pursued both in research and in industry [4]. The notion of heartbeat detectors embedded in the ECG electrode and configured in a wireless sensor network is tempting and would enable long-term ECG analysis. Substitution of the quite clumsy Holter monitors used today would make life easier.

In this work we will show how power-efficient cross correlators are implementable in standard CMOS technology, exploring bitstream-coded signals. The internal signal representation called *bitstream* is found in sigma-delta converters, but is usually decimated to Nyquist rate binary coded numbers. However, some signal processing like filtering using bitstreams are reported [5], [6], [7], [8]. Another application of bitstreams is found in the audio coding format named Direct-Stream-Digital used in SACD, developed by Sony and Philips [9].

The idea of cross-correlation using bitstreams was proposed in [10] and evaluated for heart rate variability study in [11]. In this paper we present an implementation of a complete cross-correlation chip with a binary coded interface.

II. Bitstream Cross-correlator

A discrete estimate of the cross-correlation of two sequences is found by the equation:

$$r(t) = \sum_{k=0}^{n-1} y(k)x(t+k)$$

The finite, time-variant sequence $x(t)$ of length n is cross-correlated with a template sequence, $y(t)$, by multiplying each element of the two sequences over a window of length n and summing the result. The result, $r(t)$, is a good estimate of the cross-correlation between the two finite sequences. For every new sample of the incoming signal another $r(t)$ may be computed creating another element of a cross-correlation sequence between the incoming signal and the template.

From a computational perspective we need to do n multiplications and sum the results for each sample of the incoming signal. We either need n multipliers running in parallel or to speed up the clock with a factor of n. Then we need to figure out an efficient summing operation in the simplest form requiring another n iterations following the multiplications. No wonder alternative pattern-matching measures are sought when power is precious.

In this paper we explore the idea proposed in [10] of implementing cross-correlation by processing bitstreams. By doing so, quite power efficient single chip heartbeat detectors embedded in the ECG electrode are feasible.

III. Single-chip Cross-correlator

To allow for a simple interface to the chip, both input and output signals are assumed to be Nyquist rate binary encoded. Conversion to and from oversampled bitstream representation is done on-chip. Fig. 1 shows how the bitstream is only necessary internal to the system. However, a Serial Peripheral Interface Bus (SPI) is used for interfacing, and on-chip multiplexers facilitate testing of individual blocks.

A. Bitstream Conversion

The binary-to-bitstream signal conversion is done using a two step interpolation filter and a sigma-delta modulator. The interpolation filter upsamples an input signal at the Nyquist rate by the oversampling ratio required by the modulator. Interpolation in several steps is commonly used. This allows for a combination of an anti-alias filter, with a narrow transition band, and a more hardware efficient Cascaded Integrator Comb (CIC) filter [12]. This is a compromise between filter quality and silicon area, which in turn affects power consumption.

Similarly, a decimation function is required for removing the high-frequency quantization noise present in bitstream coded signals, shown as the CIC decimator in Fig. 1.

Using a low oversampling ratio (OSR) when modulating the bitstream obtains the largest power savings [10]. As a consequence, the possible signal quality of the bitstream is limited. Compromising between these two considerations, the system OSR is set to 8. Still the bit sequence may be of significant length depending on the time-span of the correlation window. In our test-chip we use a correlation length of 1024 bits.

B. Bitstream Operations

Multiplication between bitstreams has a significant advantage compared to its multibit counterpart and can be carried out using basic logic gates. The dynamic range of the implemented modulator is normalized to $-1 \leq x \leq 1$, where x is the input. The probability that the output of the modulator, X, is 1 or 0 is then given by $P(X = 1) = (1 + x)/2$ or $P(X = 0) = 1 - P(X = 1) = (1 - x)/2$. The modulation of two input signals x and y is regarded as uncorrelated and results in two bitstreams, X and Y. The XNOR of the two bitstreams results in:

$$\begin{aligned} P(X \overline{\oplus} Y = 1) =& P(X = 0) \cdot P(Y = 0) \\ & + P(X = 1) \cdot P(Y = 1) \\ =& \frac{1}{2}(1 + xy) \end{aligned}$$

which indicates that operations usually requiring complex digital circuitry can be done with a simple logic gate when processing bitstreams. The results from bitstream multiplications are just a single bit from each multiplier. A summing operation still remains.

For power efficiency we try to avoid clocks exceeding the oversampled clock frequency. Knowing the multiplier results are all single bits, the summing is reduced to counting the number of '1' after multiplications. A novel asynchronous solution is explored to compute the sum during the same clock cycle as the multiplication. The counting operation is split in two operations:

1) Sorting bit sequence from multipliers. The sorting method used is inspired by the software bubble sort algorithm, but is implemented completely asynchronously.
2) Encoding sorted result as binary number. By interpreting the sorted bit sequence as a thermometer coded result, a binary number is encoded.

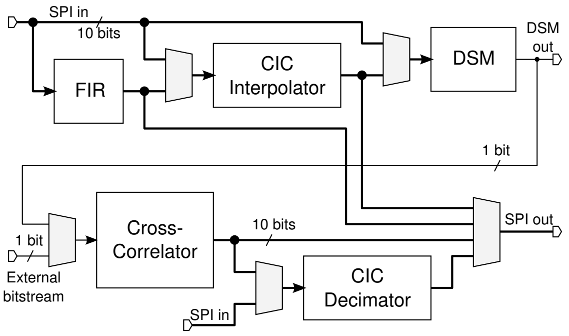


Fig. 1. Block schematic showing main signal paths. The multiplexers allow each block to be tested individually. Only the SPI in and out pins are used for the full signal path.

The asynchronous operation is achieved using inherent gate delays explained below.

C. Bitstream Cross-correlation

During setup phase, the binary-to-bitstream converter may be used or the template may be shifted in directly as a bitstream coded sequence of up to 1024 bits. Then the incoming signal is shifted into the correlation register coded as a bitstream or converted by the binary-to-bitstream converter. All bits in the two registers are "multiplied" by XNOR gates at the start of every clock cycle. The bubble register is loaded with the results from the XNOR operation after an adequate delay. Then the "bubbling" is started and the rest of the clock cycle is reserved for the asynchronous sorting operation followed by latching of results. During the next clock cycle the thermometer coded result of the bubbling is converted to a binary representation in parallel with computation of the following correlation result.

IV. Implementation

In these dedicated systems, register lengths are hard-coded by design. Depending on application, correlation window lengths must be adapted for minimal power consumption. For easy generation of different cross-correlators we have used SKILL, a LISP-like CAD system extension language. The produced SKILL scripts facilitate fast and easy generation of both schematics and layout of cross-correlators of different sizes.

A. Bitstream Cross-correlator

The bubble register used for sorting is shown in Fig. 2 and each element consists of one ordinary RS-latch, one AND gate and inverters used as delay-elements. The latches are loaded with the result from the bitstream multiplication in the beginning of the clock cycle. The bubble sorting is initiated after a predefined delay for proper settling of the latches.

It is important to notice that the exchange of bits in the bubble register is local, enabling parallel operation. Basically,

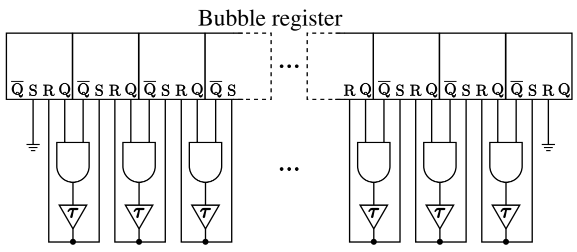


Fig. 2. Asynchronous bubble sorter.

the operation $(1, 0) \rightarrow (0, 1)$ can be carried out, provided data is stable. To ensure failsafe operation the actual exchange operation is delayed using some inverters. In this way all '1' are "bubbled" to the right while the '0' is "bubbled" to the left.

The final and stable condition of the bubble register has all '1's stacked to the right and all '0's to the left. This code is known as a thermometer code. In fact there is one and only one $(0, 1)$ sequence in the sorted result, which may be used for unique binary encoding.

The thermometer code is converted to its binary representation in the following clock cycle, while the next cross-correlation result is computed. The thermometer encoded result is fed to the binary encoder assuming a single $(0, 1)$ transition. This transition is identified by a row decoder, comparing two consecutive values from the thermometer coded array using an XOR-gate. The selected row pulls down the correct precharged output lines which encodes the desired binary result. The output lines are sampled after a predefined delay and clocked out through the SPI-interface.

B. Data Conditioning

Hiding of bitstream coding is an integral part of the cross-correlator chip. The following modules are included:

1) Interpolation filter: The anti-aliasing step of the interpolation filter is a 20-tap halfband FIR filter following an upsampling by two. This filter type has every other coefficient set to zero and is an efficient way to achieve a narrow transition band around 0.5π. Each tap in the filter is in turn simplified. Gain steps are quantized such that each are realizable as a sum of maximum two hardwired bit-shift operations. This removes the requirements for full multiplications in the filter.

The CIC filter step is of third order and performs an upsampling by four, resulting in the desired oversampling rate.

2) Sigma-Delta modulator: The modulator has a second order error feedback structure. This structure uses multibit-bit feedback and is well suited for digital input. The most important design constraint for the modulator is the requirement that the output is a bitstream. This rules out good modulator types such as cascaded or MASH architectures giving word streams. Quality of the modulator is also limited by the low OSR in the system. For an OSR of 8, modulator orders of two and more give a maximum expected signal to quantization noise ratio (SQNR) of 35–40 dB.

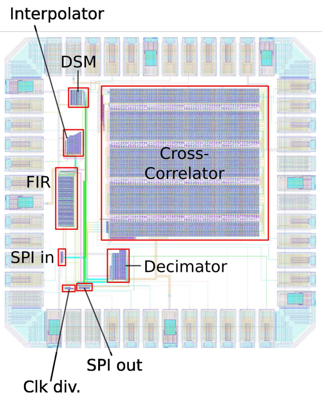


Fig. 3. Chip layout designed in STMicroelectronics 90 nm technology, delta-sigma converter and 1024-bits cross-correlator. Chip size included pads is 1×1mm

3) Decimation filter: The filter used for decimation is an ordinary CIC decimation filter of third order with a downsampling ratio of 8. This converts the oversampled output from the cross-correlation block back to a Nyquist rate signal, while reducing high frequency noise inherited from the bitstream representation.

V. Measurement Results

The chip is measured using a simple microcontroller enabling measurements of the different chip modules.

1) Interpolation filter: The frequency response of the cascaded interpolation filter is shown in Fig. 4. The response of the FIR filter step was confirmed by chip measurements, using both white noise input signals and sinusoidal inputs. Total stop band attenuation is 35–40 dB, close to expected performance.

2) Sigma-Delta modulator.: The chip modulator was tested using a full-scale sinusoidal input. Output spectrum is shown in Fig. 5 and shows the expected SQNR level.

A. System Performance

Here we briefly show the system performance by component-based ECG-analysis [3].

In Fig. 6 the cross-correlation from the test-chip is shown together with cross-correlation using the $xcorr()$-function in MATLAB. The data used is taken from the QT database [13]. The first QRS-wave from the annotated database is selected as template and loaded into the template-register as a bitstream. Then a sequence of three heartbeats are fed to the chip and the cross-correlated result is decimated and plotted. Just by inspection the cross-correlation results are promising and very close to results obtained using "ideal" cross-correlation with MATLAB. A proper evaluation of the cross-correlation chip for QRS-detection demands extensive analysis, far beyond the scope of this paper.

The main computational block doing cross-correlation is measured to 2.1 mW power consumption when active. With a clock frequency of 480 kHz and an OSR of 8 this is equivalent

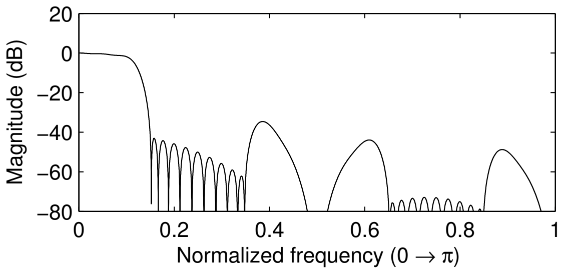


Fig. 4. Frequency response of cascaded interpolation filter. Normalized to half the oversampling frequency.

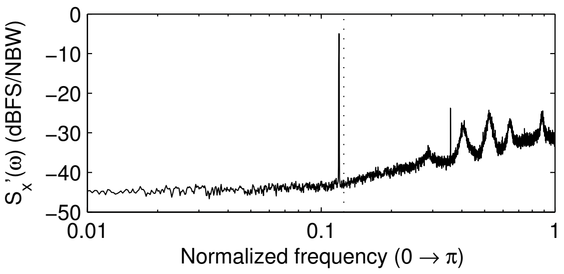


Fig. 5. Power spectral density of the sigma-delta modulator. Nyquist frequency indicated by dotted line.

to $\approx$ 7.7 M multiplications each second at Nyquist rate. In addition this first chip was a proof-of-concept with far from optimal layout. We consider these results to be very promising for low-power, single-chip pattern-recognition.

As indicated in the introduction, cross-correlation is a generic pattern-matching computational element suited for several signal processing tasks. The bitstream processing solution presented in this paper aims at low-power operation at low or moderate signal frequencies. There is a growing demand for low frequency sensor interfacing where filtering is required. The proposed cross-correlator chip may also be used as a programmable filter by time-warping the template for convolution. The cross-correlator chip is suitable both for low-frequency operation and as a programmable filter. Many of these applications are emerging with biomedical applications.

VI. Conclusion

In this paper we have presented a novel single-chip cross-correlator suitable for power-efficient pattern matching. Bitstream encoded signals found in sigma-delta converters are used for efficient multiply-and-sum operations basically substituting multipliers by simple gates.A novel asynchronous bubble-register is utilized avoiding increased clock frequency. The running cross-correlator is implemented in 90 nm technology and measured results are provided.We expect these kind of cross-correlators to be viable in power-limited, low signal frequency applications like QRS-detection of ECG-signals.

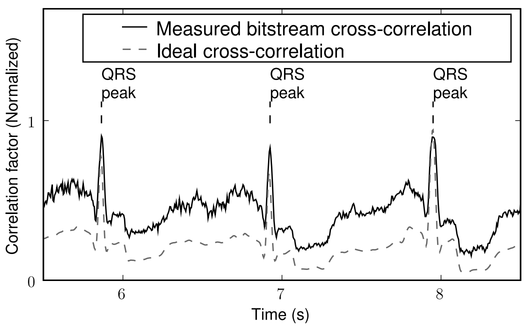


Fig. 6. Bitstream cross-correlation results.

Acknowledgment

The authors would like to thank Dept. of Informatics, University of Oslo for providing chip fabrication and lab facilities for this project.

References

[1] B. Warneke, M. Last, B. Liebowitz, and K. Pister, “Smart dust: communicating with a cubic-millimeter computer,” *Computer*, vol. 34, no. 1, pp. 44–51, jan 2001.
[2] [Online]. Available: http://www.openecg.org/
[3] T. Last, C. Nugent, and F. Owens, “Multi-component based cross correlation beat detection in electrocardiogram analysis,” *BioMedical Engineering OnLine*, vol. 3, no. 1, p. 26, 2004. [Online]. Available: http://www.biomedical-engineering-online.com/content/3/1/26
[4] [Online]. Available: http://www.toumaz.com/
[5] P. O’Leary and F. Maloberti, “Bit stream adder for oversampling coded data,” *Electronics Letters*, vol. 26, pp. 1708–1709, Sept 1990.
[6] F. Maloberti and P. O’Leary, “Processing of signals in their oversampled delta-sigma domain,” in *Circuits and Systems, 1991. Conference Proceedings, China., 1991 International Conference on*, Jun 1991, pp. 438–441 vol.1.
[7] M. F., “Non conventional signal processing by the use of sigma delta technique: a tutorial introduction,” in *Circuits and System, ISCAS ’92. Proceedings, 1992 IEEE International Symposium on*, vol. 6, May 1992, pp. 2645–2648.
[8] S. Summerfield, S. Kershaw, and M. Sandler, “Sigma-delta bitstream filtering in vlsi,” in *Circuits and Systems, 1994., Proceedings of the 37th Midwest Symposium on*, vol. 2, Aug 1994, pp. 1200–1203 vol.2.
[9] E. Janssen and D. Reefman, “Super-audio cd: an introduction,” *Signal Processing Magazine, IEEE*, vol. 20, no. 4, pp. 83–90, July 2003.
[10] T. Lande, T. Constandinou, A. Burdett, and C. Toumazou, “Running cross-correlation using bitstream processing,” *Electronics Letters*, vol. 43, no. 22, pp. –, 25 2007.
[11] M. Ho, T. Lande, and C. Toumazou, “Efficient computation of the lf/hf ratio in heart rate variability analysis based on bitstream filtering,” in *Biomedical Circuits and Systems Conference*, Nov 2007, pp. 17–20.
[12] E. B. Hogenauer, “An economical class of digital filters for decimation and interpolation,” *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on*, vol. 29, no. 2, pp. 155–162, Apr 1981.
[13] P. Laguna, R. Mark, A. Goldberger, and G. Moody, “A database for the evaluation of algorithms for measurement of qt and other waveform intervals in the ecg,” *Computers in Cardiology*, vol. 24, pp. 673–676, 1997.

B SKILL code example

The following SKILL script, `xbitmux41.il`, produces a multi bits 4 to 1 mux. A section of the produced layout running the script with parameter $l = 10$ is shown in Fig. B.1.

```
procedure( mux41( l )
  prog( ( dbcv dbmux tmp_lib_name cell_name height width d Z x y i j
        pwidth mlayer pstart ydist ddist)

    tmp_lib_name="correlation"
    cell_name=strcat("mux" sprintf(nil "%d" l) "bit41")
    dbcv=dbOpenCellViewByType( tmp_lib_name cell_name "layout" "maskLayout" "
        w")
    dbmux=dbOpenCellViewByType("master" "muxXbit41" "layout")

    declare(d[4])
    d[3]=0.150
    d[2]=0.980
    d[0]=3.205
    d[1]=3.780
    Zy=4.61
    width=3.92
    height=4.76
    pwidth=0.14
    pdist=0.2
    pdist=pdist+pwidth
    pstart=1
    x=0
    ddist=floor(l*height/4)

    declare(mlayer[4])
    mlayer[0]="M2"
    mlayer[1]="M3"
    mlayer[2]="M4"
    mlayer[3]="M5"

    for(i 0 (l-1)
      y=height*i
      x=0
      dbCreateInst( dbcv dbmux nil 0:y "R0")

      for( j 0 3

        if( j != 0
          let((contact layers )
            layers  = strcat(mlayer[j] "_M2")
                    contact = dbOpenCellViewByType("cmos090" layers "symbolic")
                      dbCreateInst(dbcv contact nil 0:y+d[j] "R90")
          )
        )
        if( (y+d[j]) < ddist*j+pdist*i then
          x=9
          dbCreatePath(dbcv list(mlayer[j] "drawing") list(0:y+d[j]
                                  -pstart-pdist*(x-i):y+d[j]
                                  -pstart-pdist*(x-i):ddist*j+pdist*i
                                  -pstart-pdist*l:ddist*j+pdist*i
                                   ) pwidth )
        else
          dbCreatePath(dbcv list(mlayer[j] "drawing") list(0:y+d[j]
                                  -pstart-pdist*i:y+d[j]
                                  -pstart-pdist*i:ddist*j+pdist*i
                                  -pstart-pdist*l:ddist*j+pdist*i
                                   ) pwidth )
        )

        let((net_name net pin term text)
                  ;input
                  sprintf(net_name "D%d<%d>" j i)
                  net = dbCreateNet(dbcv net_name)
                  term = dbCreateTerm(net nil "input")
                  pin=dbCreatePin(net dbCreateRect(dbcv list(mlayer[j] "pin")
                        list(-pstart-pdist*l:ddist*j+pdist*i-pwidth/2
                             -pstart-pdist*l+pwidth:ddist*j+pdist*i+pwidth/2 )
                            )
                  )
                  text = dbCreateTextDisplay(term term list(mlayer[j] "pin") t
                             -pstart-pdist*l+pwidth/2:ddist*j+pdist*i
                        "centerRight" "R0" "swedish" 0.16 t nil t nil t "name")
                  text~>parent = pin~>fig
            );let


        );for j
        let((net_name net pin term text)
                  ;input
                  sprintf(net_name "out<%d>" i)
                  net = dbCreateNet(dbcv net_name)
                  term = dbCreateTerm(net nil "output")
                  pin=dbCreatePin(net dbCreateRect(dbcv list("M2" "pin")
                        list(width-0.14:y+Zy-0.07 width:y+Zy+0.07)
                            )
                  )
                  text = dbCreateTextDisplay(term term list("M2" "pin") t
                            width −0.07:y+Zy
                        "centerLeft" "R0" "swedish" 0.16 t nil t nil t "name")
                  text~>parent = pin~>fig
              );let
    );for i

  let((net_name net pin term text AB layer)
    declare(AB[4])
    AB[0]=2.72
    AB[1]=2.395
    AB[2]=0+0.07
    AB[3]=width −0.07
    declare(net_name[4])
    net_name[0]="A"
    net_name[1]="B"
    net_name[2]="vdd"
    net_name[3]="gnd"
    declare(layer[4])
    layer[0]="M2"
    layer[1]="M3"
    layer[2]="M1"
    layer[3]="M1"
        for( i 0 3
            net = dbCreateNet(dbcv net_name[i])
            term = dbCreateTerm(net nil "input")
            pin=dbCreatePin(net dbCreateRect(dbcv list(layer[i] "pin")
                        list(AB[i]−0.07:0 AB[i]+0.07:0+0.14)
                          )
                )
            text = dbCreateTextDisplay(term term list(layer[i] "pin") t
                      AB[i]:0+0.07
                        "centerTop" "R0" "swedish" 0.16 t nil t nil t "name")
            text~>parent = pin~>fig
    )
  )

    dbSave(dbcv)
  ; dbClose(dbcv)
  ; dbPurge(dbmux)

  );prog
);procedure
```

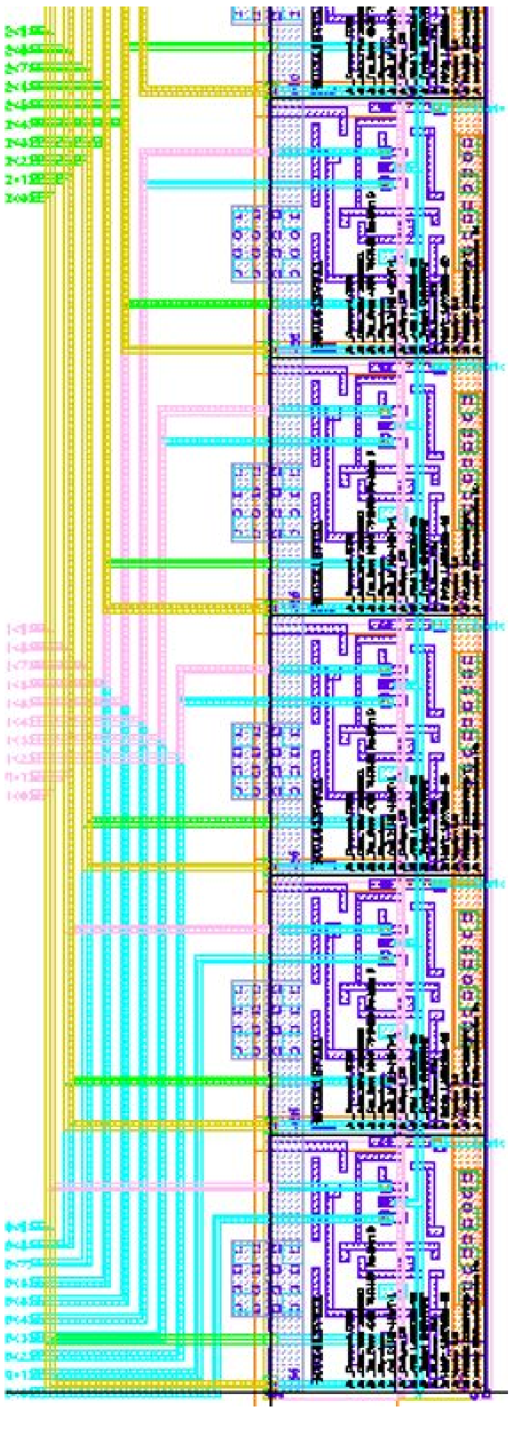


Figure B.1: Section of the layout produced by the script `xbitmux41.il`.

C Microcontroller firmware

The following code excerpt is the most important methods in the μc firmware, written in ANSI C.

```c
#include "xcorr.h"
#include "sam7.h"
#include "Board.h"
#include "peripherals.h"
#include "usb.h"

#define UINT_MAX 4294967295

unsigned int incoming[16];
int reg_length=512;
int templen=0;
int dreglen=0;

// Clear signal reseting internal clock dividers
void xcorr_clear()
{
  int i = 0;

  AT91F_PIO_ClearOutput(pPIO, PIO_CLEAR);
  for(i=0;i<10000;i++)
    ;

  AT91F_PIO_SetOutput(pPIO, PIO_CLEAR);
}

// Sets one of the registers to 1's or 0's
void xcorr_set_reg(int reg, int value)
{
  int i;

  if(reg)
    AT91F_PIO_SetOutput(pPIO, PIO_TEMPLATE_ENABLE);
  else
    for(i=0;i<32;i++)
      if(i<16)
        incoming[i]=value*UINT_MAX;
      else
        incoming[i]=0;

  AT91F_PIO_SetOutput(pPIO, PIO_EXTERNAL_DS_ENABLE);
  xcorr_clear();

  for(i=0; i<1024; i++)
  {
    spi_send(1, value*1023);
    if(i==reg_length)
      value=reg;
  }

  if(reg)
    AT91F_PIO_ClearOutput(pPIO, PIO_TEMPLATE_ENABLE);

  usb_print("\nreg ");
  usb_print_int(reg);
    usb_print("=");
  usb_print_int(value);
  usb_println("set reg confirmed");
}

// Sets bit offset
void xcorr_spi_offset(int offset)
{
  int i;
  spi_cfg_cs(1, 0, 0, 1, 1, 1);
  for(i=0;i<offset;i++)
    spi_send(1, 1);
  spi_cfg_cs(1, 0, 0, 1, 10, 10);
  usb_print("offset=");
  usb_print_int(offset);
  usb_println("");

}
// Sets the necessary chip input pins to before running CC
void xcorr_run_setup()
{
  AT91F_PIO_ClearOutput(pPIO, PIO_TEMPLATE_ENABLE);
  AT91F_PIO_SetOutput(pPIO, PIO_EXTERNAL_DS_ENABLE);
  AT91F_PIO_SetOutput(pPIO, PIO_CONFIG_OUT_A);
  AT91F_PIO_SetOutput(pPIO, PIO_CONFIG_OUT_B);
  xcorr_clear();
  usb_println("Waiting for incoming data");
}

// Sets the necessary chip input pins and clocks in register values
void xcorr_temp_setup(int value)
{
  if(value<0)
  {
    templen=0;
    AT91F_PIO_SetOutput(pPIO, PIO_TEMPLATE_ENABLE);
    AT91F_PIO_SetOutput(pPIO, PIO_EXTERNAL_DS_ENABLE);
    xcorr_clear();
    usb_println("Waiting for template data");
    return;
  }
  if(templen++<reg_length)
    spi_send(1,value*1023);

  if(templen==reg_length)
  {
    for(templen=0; templen<(1024-reg_length); templen++)
      spi_send(1, 1023);
    templen=reg_length+1;
  }
}

// Sets the necessary chip input pins and clocks in register values
void xcorr_dreg_setup(int value)
{
  int i;

  if(value<0)
  {
    dreglen=0;
    AT91F_PIO_ClearOutput(pPIO, PIO_TEMPLATE_ENABLE);
    AT91F_PIO_SetOutput(pPIO, PIO_EXTERNAL_DS_ENABLE);
    xcorr_clear();
    usb_println("Waiting for dreg data");
    return;
  }
  if(dreglen<reg_length)
  {
    i=dreglen/32;
    spi_send(1,value*1023);

    incoming[i] |= (value<<(dreglen%32));

  }
  dreglen++;

  if(dreglen==reg_length)
  {
    for(dreglen=0; dreglen<(1024-reg_length); dreglen++)
      spi_send(1, 0);

    dreglen=reg_length+1;
  }
}
```

```

// The bugfixed version is ran if reg_length=512
int xcorr_spi_send_recv_bugfix(int value)
{
  int i, j;
  int temp;

  if(reg_length==1024)
    return spi_send_recv(1,value*1023);

  for(i=0;i<16;i++)
  {
    if(i<15)
      incoming[i] = (incoming[i] >> 1) | ( (incoming[i+1]&1)<<31);
    if(i==15)
      incoming[i]= (incoming[i] >> 1) | ( value<<31);
    for(j=0;j<32;j++)
    {
      temp=(incoming[i]>>j)&1;
      spi_send_recv(1, 1023*temp);
    }
  }
  for(i=0;i<511;i++)
    spi_send_recv(1, 0);

  return spi_send_recv(1, 0);
}
// Set the length of the bubble register
void xcorr_set_reg_length(int len)
{
  reg_length=len;
}

// Method used while measuring current
int xcorr_currmeas(int f)
{
  int i;
  int j;

  spi_cfg_cs(1, 0, 0, 1, 10, 10);


  for(j=0;j<1000;j++){
    for(i=0;i<1024;i++)
      spi_send_recv(1, 1023);

    for(i=0;i<1024;i++)
      spi_send_recv(1, 0);
  return 1;
}
```

D Python chip library

The following code excerpt is the most important methods for reading and writing data to the μc from a computer.

```
#!/usr/bin/env python
import serial
from pylab import *

com = None
_run = False
TEMPLATE=1
DREG=0
usblen=32
reg_length=512
temp=''

def readbs(file):
  f=open(file,'r')
  s=f.read()
  f.close()
  return [int(b) for b in s.strip()]

def decimator( bs ):
  OSR=8
  return [sum(bs[i:i+OSR]) for i in range(0,len(bs)-OSR, OSR)]

def init_com(device = '/dev/ttyACM0'):
  global com
  com = serial.Serial( port=device, timeout=0.1)
  com.open()
  if com.isOpen(): print "Connected to", device
  usb_send('reg_length '+ str(reg_length))

def usb_send(command='',response='> '):
  from time import sleep
  global temp
  sleep(0.1)
  com.write(command+'\n')
  out=''
  out += com.read(1)

  while com.inWaiting():
    out += com.read(1)

  if not out.endswith(response):
    print "Unexpected response from uc:"
    print out
    print "Retrying once"
    while com.inWaiting() > 0:
      out += com.read(1)
    print "----"

  return out

def usb_clear():
  from time import sleep
  out=''
  out += com.read(1)
  sleep(0.1)
  while com.inWaiting():
    out += com.read(1)

def set_template(data):
  global _run
  _run = False
  r= usb_send('tempsetup')
  if r.find('Waiting for template data')<0:
    print 'Error setting template register: ', r
    #return 0
  for i in range(0, len(data), usblen):
    usb_send('tempsetup '+ str(data[i:i+min(usblen,(len(data)-i))])[1:-1]).
        split()[:-1]

def set_dreg(data):
  global _run
  _run = False
  if usb_send('dregsetup').find('Waiting for dreg data')<0:
    print 'Error setting dreg register'
    #return 0
  for i in range(0, len(data), usblen):
    usb_send('dregsetup '+ str(data[i:i+min(usblen,(len(data)-i))])[1:-1]).
        split()[:-1]

def run(data):
  global _run
  if not _run:
    _run=True
    r=usb_send('runsetup')
    if r.find('Waiting for incoming data')<0:
      print 'Error setting run mode: ', r
      #return 0
    if usb_send('spi_offset 1').find('offset=')<0:
      print 'Error setting offset'
  r=[]
  resp=''

  for i in range(0, len(data), usblen):
    resp+=usb_send('xcorr_spi_send '+ str(data[i:i+min(usblen,(len(data)-i))
        ])[1:-1])
    print "\t\t\t%i perc" %(100*i/len(data))
  resp=resp.replace('>', ' ')
  resp=resp.split()
  r.extend([int(x,16) for x in resp])

  return r

def cc(x,t,reglength=512):
  global reg_length,zeros, ones
  zeros=[0]*reg_length
  ones=[1]*reg_length
  reg_length=reglength

  init_com("/dev/ttyACM0")
  usb_send('reg_length '+ str(reg_length))
  if type(x) is str:
    incoming=readbs(x)[:15000]
  else:
    incoming=x
  if type(t) is str:
    template=readbs(t)[:reglength]
  else:
    template=t[:reglength]

  set_template(template)
  set_dreg(incoming[0:reglength])
  h=run(incoming[reglength:], liveplot=0, decimate=0)
  return h



if __name__ == '__main__':
  try:
    # CC between record and QRS complex template
    r=cc('heartdata/sel100.bs','heartdata/N.bs')
    raw_input('enter to exit...')
  except e:
    print "ERROR" + e.value
  except KeyboardInterrupt, e:
    ioff()
    print e
    raw_input()
```

E QT database wrapper script

The following script extracts ECG recordings from the QT database. The `main()` function shows an example on how to use the script. The script relies on the two programs, *rdsamp* and *rdann* provided by the creators of the database.

```
#!/usr/bin/env python

import commands, os
from pylab import *

qtdb='/ifi/midgard/p15/bitstream/olav/qt/qtdb/'

def getRecord(record='sel100', update=False):
  """
  Convert record to a list
  Saves the converted record as an ascii file
  Returns record as a list
  """
  p=os.path.realpath('.')
  os.chdir(qtdb)
  ascii=os.path.join('ascii',record+'.dat.txt')
  if update or not os.path.isfile(ascii):
    command="rdsamp -r " + record + ">" + ascii
    failure, output = commands.getstatusoutput(command)
    if failure:
      print output
      os.chdir(p)
      return -1
  lines = open(ascii).readlines()
  data=[[],[],[]]
  for line in lines:
    s=line.split()
    data[0].append(int(s[0]))
    data[1].append(int(s[1]))
    data[2].append(int(s[2]))
  os.chdir(p)
  return data

def getAnnotation(record='sel100', annotator='q1c', update=False):
  """
  Extract annotations to the belonging record
  Saves the converted annotation as an ascii file
  Returns list of annotations
  """
  p=os.path.realpath('.')
  os.chdir(qtdb)
  ascii=os.path.join('ascii', record + '.' + annotator + '.txt')
  if update or not os.path.isfile(ascii):
    command="rdann -r " + record +" -a " + annotator + ">" + ascii
    failure, output = commands.getstatusoutput(command)
    if failure:
      print output
      os.chdir(p)
      return -1
  lines = open(ascii).readlines()
  data=[[],[],[]]
  for line in lines:
    s=line.split()
    data[0].append(int(s[1]))
    data[1].append(str(s[2]))
    data[2].append(int(s[4]))
  os.chdir(p)
  return data

def plotRecord(record, annotators=None, als='--'):
  """
  Plot the record and the given anotations
  """
  dmax=max(record[1])
  dmin=min(record[1])
  toffset=20

  acolor=['k','.5','c','r','y','g','k','m','0.3','0.6','0.9']

  plot(array(record[0])/250.,record[1],acolor[0])
  x1=record[0][0]
  x2=record[0][-1]
  if annotators:
    for j in range(len(annotators)):
      for i in range(len(annotators[j][0])):
        if annotators[j][0][i]>=x1 and annotators[j][0][i] <= x2:
          x=array(annotators[j][0][i])/250.
          plot([x,x],[dmin,dmax+(len(annotators)-j)*toffset],acolor[j+1],ls=
              als)
          text(x,dmax+(len(annotators)-j)*toffset, annotators[j][1][i],color=
              acolor[j+1])
  xlabel('Time (s)')

def main():
  record='sel100'
  data=getRecord(record,update=0)
```

```
    an='q1c'

    a=getAnnotation(record, an)

    plotRecord(data, [a])
    show()

if __name__ == '__main__':
    main()
```

Bibliography

[Alpe 86] N. Alperin and D. Sadeh. "An improved method for on-line averaging and detection of ECG waveforms". *Comput. Biomed. Res.*, Vol. 19, No. 3, pp. 193–202, 1986.

[Davi 97] A. Davis and S. M. Nowick. "An Introduction to Asynchronous Circuit Design". Tech. Rep. UUCS-97-013, September 1997.

[Dobk 06] R. Dobkin, R. Ginosar, and A. Kolodny. "Fast asynchronous shift register for bit-serial communication". *Asynchronous Circuits and Systems, 2006. 12th IEEE International Symposium on*, pp. 10 pp.–127, March 2006.

[Feng 03] W.-C. Feng. "Making a Case for Efficient Supercomputing". *Queue*, Vol. 1, No. 7, pp. 54–64, 2003.

[Geer 05] D. Geer. "Is it time for clockless chips? [Asynchronous processor chips]". *Computer*, Vol. 38, No. 3, pp. 18–21, March 2005.

[Inos 62] H. Inose, Y. Yasuda, and J. Murakami. "A telemetering system by code modulation, delta sigma modulation". *IRE Transactions on Space Electronics and Telemetry*, Vol. 8, pp. 204–209, 1962.

[Jans 03] E. Janssen and D. Reefman. "Super-audio CD: an introduction". *Signal Processing Magazine, IEEE*, Vol. 20, No. 4, pp. 83–90, July 2003.

[Kim 03] N. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, and V. Narayanan. "Leakage current: Moore's law meets static power". *Computer*, Vol. 36, No. 12, pp. 68–75, December 2003.

[Lagu 97] P. Laguna, R. Mark, A. Goldberger, and G. Moody. "A database for the evaluation of algorithms for measurement of QT and other waveform intervals in the ecg". *Computers in Cardiology*, Vol. 24, pp. 673–676, 1997.

[Land 07] T. Lande, T. Constandinou, A. Burdett, and C. Toumazou. "Running cross-correlation using bitstream processing". *Electronics Letters*, Vol. 43, No. 22, 2007.

[Last 04] T. Last, C. Nugent, and F. Owens. "Multi-component based cross correlation beat detection in electrocardiogram analysis". *BioMedical Engineering OnLine*, Vol. 3, No. 1, p. 26, 2004.

[Lind 88] K. Lindecrantz and H. Lilja. "New software QRS detector algorithm suitable for real time applications with low signal to noise ratio". *Journal of Biomedical Engineering*, Vol. 10, pp. 280–284, 1988.

[Malo 91] F. Maloberti and P. O'Leary. "Processing of signals in their oversampled delta-sigma domain". *International Conference on Circuits and Systems. Conference Proceedings, China.*, Vol. , No. , pp. 438–441 vol.1, June 1991.

[Mart 01] A. J. Martin, M. Nystroem, P. Penzes, and C. Wong. "Speed and Energy Performance of an Asynchronous MIPS R3000 Microprocessor". In: *Caltech Computer Science Technical Report*, June 2001.

[Mart 97] A. J. Martin, A. Lines, R. Manohar, M. Nystroem, P. Penzes, R. Southworth, U. Cummings, and T. K. Lee. "The Design of an Asynchronous MIPS R3000 Microprocessor". In: *Proceedings of the 17th Conference on Advanced Research in VLSI (ARVLSI '97)*, p. 164, IEEE Computer Society, Washington, DC, USA, 1997.

[McCl 03] J. McClellan, M. A. Yoder, and R. Schafer. *Signal Processing First*. Prentice Hall, 2003.

[Mo 09] D. Mo. *To be published*. Master's thesis, University of Oslo, 2009.

[Mull 59] D. E. Muller and W. S. Bartky. "A Theory of Asynchronous Circuits". In: *Proceedings of an International Symposium on the Theory of Switching*, pp. 204–243, Harvard University Press, April 1959.

[Nors 97] S. R. Norsworthy, R. Schreier, and G. C. Temes. *Delta-Sigma Data Converters : Theory, Design, and Simulation*. John Wiley & Sons, 1997.

[Sail 07] E. Sail and M. Vesterbacka. "Thermometer-to-binary decoders for flash analog-to-digital converters". *Circuit Theory and Design, 2007. ECCTD 2007. 18th European Conference on*, pp. 240–243, August 2007.

[Schr 05] R. Schreier and G. C. Temes. *Understanding Delta-Sigma Data Converters*. John Wiley & Sons, 2005.

[Sham 98] M. Shams, J. Ebergen, and M. Elmasry. "Modeling and comparing CMOS implementations of the C-element". *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, Vol. 6, No. 4, pp. 563–567, Dec 1998.

[SKIL 03] *SKILL Language Reference*. Cadence Design Systems, 06.30 Ed., 9 2003.

[Smit 97] S. W. Smith. *The scientist and engineer's guide to digital signal processing*. California Technical Publishing, San Diego, CA, USA, 1997.

[Spar 01] J. Sparsø and S. Furber. *Principles of Asynchronous Circuit Design - A Systems Perspective*. Kluwer Academic Publishers, December 2001.

[Weis 91] M. Weiser. "The Computer for the Twenty-First Century". *Scientific American*, Vol. 265, No. 3, pp. 94–104, 1991.

[West 05] N. H. Weste and D. Harris. *CMOS VLSI Design. A circuit and Systems Perspective*. Pearson, 2005.