NTNU – Trondheim
Norwegian University of
Science and Technology

Efficient Implementation of Cross-Correlation in Hardware

David Eirik Taylor

Electronics System Design and Innovation
Submission date: May 2014
Supervisor: Kjetil Svarstad, IET
Co-supervisor: Lloyd Clark, Atmel Norway AS

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Abstract

Low-area matched filter and correlator designs are explored in this thesis, for ADC resolutions of 1- and 2-bits. Correlators are used extensively in spread-spectrum communication technologies, where they serve as a means of detecting a known pseudo-random sequence (PN code). The correlator designs presented here are intended for direct-sequence spread spectrum (DSSS) radio, where the data to be sent is expanded using either the PN code, or the inverse of the PN code. The correlator or matched filter will then respond with a positive or negative peak when a data bit is detected.

To test various correlator designs a testbench is developed in MATLAB, where a DSSS data sequence can be created and corrupted with an adjustable level of white Gaussian noise. The data stream with noise is filtered with an automatic gain stage, and sampled using an ADC of variable resolution and sampling rate. The sampled signal is then fed to a mathematical model of the given correlator design to see how it behaves. For an objective measure of performance in the presence of noise, a novel noise immunity test bench was developed, which subjects the correlator models to a signal with increasing levels of noise. The SNR where the correlator is no longer able to extract the correct data bits from the signal is considered the noise immunity level.

Several HDL matched filter designs are presented for both 1- and 2-bits of ADC resolution. The 1-bit matched filters are tested using the Barker-11 PN code, whereas the 2-bit correlators are tested using a 36 chip long chirp sequence. For both the 1- and 2-bit correlators, a specific design type using a multiplexed parallel counter was the most area efficient. A novel grouping correlator design is also presented for 2-bit operation, however the area required by this design is larger than that of the other designs. The results from the grouping design indicate that a significant reduction in dynamic power is present. In terms of power efficiency, the dual correlator designs showed promising results of half the power consumption of the other designs. The design of parallel bit counters used in the matched filters are also presented, along with the area per bits required for each design.

Verification of the designs is performed using mathematical correlator models, which are subjected to the same input as the Verilog modules. The results from these two tests are compared, and any discrepancies are reported to the user of the testbench. The mathematical and Verilog correlator models are fed with a simulated real-world input signal, which is essentially random noise for purposes of testing functionality.

Sammendrag

Temaet for denne masteroppgaven er implementasjon av matchetfiltre og korrelasjonsenheter med minimalt arealforbruk, og for ADC-oppløsninger på henholdsvis 1 og 2 bit. Korrelasjonsenheter brukes utbredt innen spread-spectrum kommunikasjonsteknologi, der de har som oppgave å oppdage en på forhånd kjent pseudotilfeldig sekvens (PN kode). Enhetene presentert her er utviklet for "direct-sequence spread spectrum" (DSSS) radio, der hvert enkelt databit representeres enten med PN koden, eller den inverse av PN koden, avhengig av verdien til databiten. Korrelasjonsenheten vil gi maksimalt eller minimalt utslag hver gang PN koden mottas, og disse utslagene kan leses av som de opprinnelige databitene.

For å teste støypåvirkning og andre faktorer på korrelasjonen ble en testbenk utviklet i MATLAB, der en DSSS datasekvens kan lages og ispes med et valgt nivå av Gaussisk hvit støy. Datastrømmen med støy blir så normalisert, og deretter samplet ved hjelp av en analog til digital konverter med justerbar oppløsning og samplingsrate. Det samplet signalet settes så inn i en matematisk modell av korrelasjonsenheten som testes for å se hvordan den oppfører seg. For å få et objektivt mål på evnen til å motstå støy ble det utviklet en støyimmunitetstestbenk, som tilfører et signal med stadig høyere støynivå til en korrelasjonsmodell. SNR-nivået der dataen i signalet ikke lenger tolkes riktig blir satt som støyimmuniteten til korrelasjonsmodellen.

Flere HDL matchetfilterdesign presenteres for både 1 og 2 bit med oppløsning. 1 bit matchetfilterene bruker Barker-11 sekvensen som PN kode, mens 2 bit enhetene bruker en 36 chip lang chirpsekvens. For både 1 og 2 bit enhetene var det et bestemt design som brukte en multiplexed parallellteller som var mest arealeffektiv. En original korrelasjonsenhet som grupperer ADC samples blir også presentert for 2 bit ADC oppløsning, og som viser en betydelig reduksjon i dynamisk strømforbruk. Strømforbruket til dual correlator designene viste en halvering i strømforbruket sammenlignet med de andre enhetene. Design av parallellbittellerne som brukes i korrelasjonsenheten blir også presentert, sammen med arealet per telte bit.

Designverifisering er gjort ved bruk av matematiske modeller, som utsettes for de samme inngangssignaler som Verilog modulene. Resultatene fra disse to testene sammenlignes, og forskjeller rapporteres til brukeren av testbenken. Både de matematiske og Verilogmodulene mates med en simulert DSSS signal, som for testformål er tilfeldig støy.

Thesis Details

Candidate Name: David Eirik Taylor

Assignment Title: Efficient Implementation of Cross-correlation in Hardware

Supervisor: Kjetil Svarstad

External Supervisor: Lloyd Clark, Atmel Norway AS

Problem description

Cross-correlation is a signal processing technique that is extremely useful in wireless receivers, radar, image recognition, and many other scenarios in which a pattern must be reliably detected in noise. This thesis would explore how to implement cross-correlation in hardware to minimize gate count and/or power consumption. Of special interest for this project are implementations in which the signal to be correlated is obtained from a low-resolution analog-to-digital converter with only one or two bits of output.

Preface

This Master's thesis was written at the Norwegian University of Science and Technology (NTNU), under the department of Electronics and Telecommunications. It signifies the academic pinnacle of my time at University, and the final barrier before earning a Master of Science degree in Electrical Engineering.

I would like to thank my supervisor at NTNU, Professor Kjetil Svarstad, and my external supervisor at Atmel, Lloyd Clark, for always being available when required, and for providing valuable feedback during the course of this thesis. I would also like to thank my friends here in Trondheim for making these years the most fun in my life, and my parents for encouraging my pursuit of higher education.

David Eirik Taylor
Trondheim, May 2014

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

This report focuses on the design of a small footprint correlator, for use in direct-sequence spread spectrum communication technologies. With recent advances in low power electronics combined with increasingly smaller integrated circuits, small, battery-driven smart devices are becoming ever more prevalent. Cross-correlation is an integral part of different wireless systems such as satellite navigation (GPS, Galileo and GLONASS), WLAN, RFID, IS-95, WCDMA, CDMA 2000, etc. In this report the design of correlators connected to a two level (1-bit) and three level (2-bit) ADC are explored.

1.1 Contribution

Correlators for digital cross-correlation are mature technology, with most earlier work aimed at reducing power consumption, or on filters with special PN codes and/or units using several bits of ADC resolution. Area minimization of a single low ADC resolution matched filter has however not been mentioned often in literature, and is the objective of the design effort in this thesis.

During the course of this thesis, a number of matched filter and correlator designs were created. Of these, some proved particularly area efficient, while others showed promising results in terms of power consumption. As minimal area consumption was an expressed goal for these designs, this was the center of the design focus. The 1-bit correlator designs were designed first in an exploratory manner, with varying methods which sought to minimize the area. Synthesis of these modules was not performed until they had all been designed, and as the main purpose of the 1-bit correlators was to determine which trade-offs were present, no further optimization was performed on these designs. The 2-bit designs were based around the most area-efficient 2-bit correlators, along with a few novel designs only possible with increased ADC resolution.

1.2 Report Layout

The first chapters (2 through chapter 3) of this report are aimed at exploring how matched filters operate and behave, and deal with the basics of direct-sequence spread spectrum radio, some of the previous work in this field, and the MATLAB simulation of a radio link with correlator. A novel noise immunity testbench is also presented. Once the theoretical background has been covered, the architectural design process is presented in chapter 4, where various HDL modules are described for 1-bit and 2-bit resolution, followed by the sub-modules used in the correlator designs. A description of how their correct functioning has been verified is given in chapter 5. After these chapters the results are presented in terms of area consumption, computation speed, noise immunity and power dissipation in chapter 6. The results are then discussed, and a conclusion presented finally in chapters 7 and 8.

Chapter 2

Background Material

Some material is presented here to give an insight into the applications of digital cross-correlation and how this estimate compares to an analogue correlation. The theoretical basis for cross-correlation is briefly mentioned, before the fundamentals behind direct-sequence spread spectrum radio are presented.

2.1 Digital cross Correlation

Cross correlation provides a measure of similarity between two waveforms, as a function of a time lag between them. This operation is commonly used to find one signal within another, as not only is the degree of similarity determined, but also the phase difference or delay between the two signals. The mathematical definition of the discrete time cross correlation is:

$$f \star g[n] = \sum_{m=-\infty}^{\infty} f^*[m]g[m+n] \tag{2.1}$$

For all samples m, and at time lag n. In a digital system the signals will be quantized, leading to quantization noise in the correlation signal. The performance of a discrete correlator relative to an analog correlator is known as the degradation factor, which is a measure of how well a discrete correlator can estimate the correlation of two signals compared to the estimation made by an analog correlator. It has been shown that this factor can be calculated analytically, and the results for some bit sizes are presented in table 2.1.[3][6]

Bits (n)	Sampling rate	Degradation factor
1	2B	1.57
1	4B	1.35
2	2B	1.14
2	4B	1.06
3	2B	1.05
Infinite	-	1.00

Table 2.1: Degradation factor for various sampling rates at 2^n quantization levels.

2.2 Direct-Sequence Spread Spectrum Radio

In spread spectrum technologies a signal is purposely spread in the frequency domain, in order to increase its bandwidth beyond that required just to send the data. The benefits of spread spectrum communications are antijamming, antiinterference, low probability of intercept, multiple user random access communications with selective addressing capability, high resolution ranging and accurate universal timing.[10] The act of spreading the signal is done by injecting a higher frequency spread spectrum code into the transmitting chain. Despreading the signal again is done at the receiver side, and is only possible with the same spread spectrum code used to spread the orignal signal. The code used to spread the signal needs to be as noise-like as possible, but also reproducible by both the transmitter and receiver. For this, pseudo noise codes, or PN codes, are used. A valid PN code must fulfill a number of criteria pertaining to its statistical properties, and the most common codes are named. Often used PN codes are maximal length sequences, Gold codes, Hadamard-Walsh codes, Kasami codes, and Barker codes.[1][8]

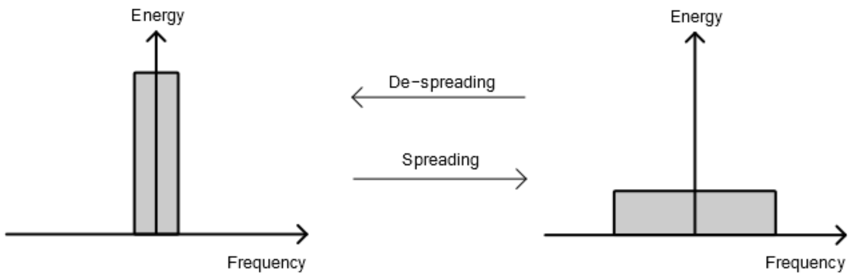

Figure 2.1: Power spectral density during spread/de-spreading.

In direct-sequence spread spectrum radio, or DSSS, the PN code is mixed with the data bit prior to transmission. E.g. a 1 may be represented by a given PN code, whereas a zero by the inverse of the same PN code. This is the case in the original IEEE 802.11 standard.[2] DSSS is also used in satellite navigation systems (GPS, Galileo an GLONASS) and CDMA cellular phones. The bits in a PN code are referred to as chips, to differentiate them from the data bits. A single data bit after spreading will contain as many chips as the PN code length when using a so-called ”short code”. When using a ”long code” the data bits might be spread with different portions of the original PN code, meaning that it will take a number of data bits before the same chip sequence is transmitted again.[17] In this thesis only short codes are considered. To despread a received signal coded in this way, the cross correlation between the incoming signal, and the selected PN code is calculated with each incoming sample of radio data. A data bit 1 corresponding to the PN code will cause a positive spike in the correlation waveform, whereas a data bit 0 will cause a negative spike. Due to the auto-correlation properties of the PN codes, noise gives a correlation score near 0 for a normalized output between [1,-1]. Reading data from a the cross correlation waveform is a simple matter of detecting spikes, and their polarity, which can be performed with two threshold detectors.

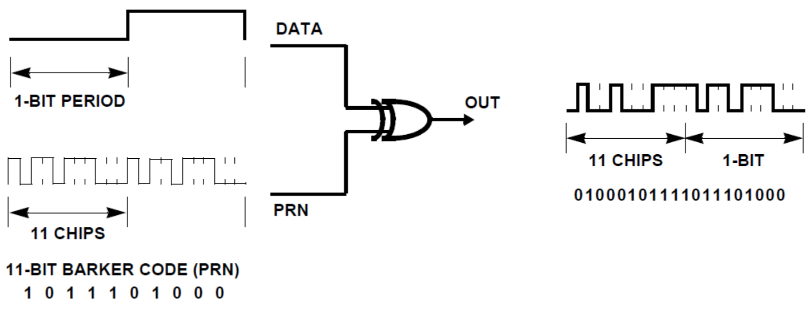

Figure 2.2: Spreading of data using the Barker-11 PN code.

The cross correlation unit used is typically called a correlator, or matched filter. In many implementations the PN code is constant, which reduces the complexity of the cross correlation down to a matched filter operation, which is merely a finite impulse response filter, with coefficients equal to the time-reversed PN code. For an input sequence exactly equal to or exactly inverse of the filter coefficients, the output of a matched filter will be maximal or minimal respectively. It is the implementation of such a matched filter which is explored in this thesis.

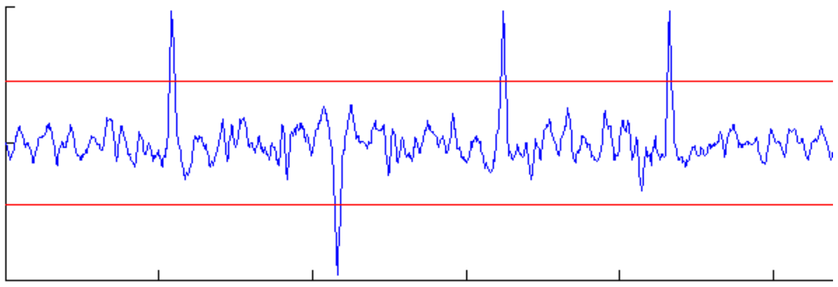

Figure 2.3: Example of DSSS correlation waveform, for the data sequence 1011.

Multiple access can be achieved with DSSS through CDMA (code division multiple access). Here each transmitter and receiver pair are given a unique PN code to communicate with. The data from other transmitters is encoded with a different PN code than that the receiver holds, and will only be interpreted as random noise, resulting in low correlation scores. One of the properties of a good PN code is that it has poor correlation with other PN codes. Multiple transmitter/receiver groups using CDMA on the same frequency band is in some ways analogous to several people in a room speaking different languages.

2.3 Previous Work

Matched filters for digital cross-correlation have been in use for many years, but the bulk of previous work done on them has been focused on low power solutions, filters with special PN codes, area reduction of the search algorithms needed in the PN code synchronization phase[4] [14] and/or units using several bits of ADC resolution. Area minimization of a single low ADC resolution matched filter seems to be a somewhat overlooked topic.

For reducing power consumption, a number of clever methods have been utilized. In [5] the authors shift the PN code rather than the ADC register to reduce the switching activity. The ADC samples are stored in a register file instead, meaning the oldest sample is replaced with the newest sample on every ADC clock cycle. Power savings are also made in the calculation of the correlation, by using a threshold detector to monitor the result from an initial summation block, and if it is below some threshold the second summation block is halted. Further savings are made by switching from signed 2's complement notation, to an offset binary notation, which simplifies the numerous summation operations. Various other approaches can be found in [12] [7].

In the case of a specially chosen PN code, [15] has shown that the correlator can be organized in a manner where a repeating structure is exploited to reduce the computational complexity. The PN code is chosen such that it consists of one inner PN code of length L expanded by another outer PN code of length M. The correlator can then be built with an inner matched filter using the coefficients of the inner code, and with a delay-line of length L. The results from this filter are fed into another matched filter using the coefficients of the outer code, with delay-line length M. The output from the outer matched filter is the total correlation. The first matched filter uses a unit delay, whereas the outer matched filter uses a delay of L.

A method of grouping and segmenting the filter structure is presented in [16], in which the length of the tapped delay line is reduced by grouping filter coefficients. The grouped outputs are summed using a Finite-Delay Accumulator, and some clever multiplexing to reduce the resource requirements in a FPGA. A straightforward approach is given in [13] where both the ADC- and PN code register can be switched to circular buffers. By doing so, the numerous multiplication and summation steps can be reduced to a single multiplier and counter unit.

Chapter 3

Specification and System Analysis

Prior to implementing any correlator designs in Verilog, simulations were performed in MATLAB. To get a feel for how a correlator unit behaves in a radio application, a simulated radio link was used to transmit data to a receiver. The receiver quantizes the data from the radio link, and applies it to a correlator unit. The quantization steps can be altered, as can the signal-to-noise ratio and per-chip sampling rate.

To get an objective measure of how well a given correlator topology (sampling rate and quantization level) performs at various levels of noise, a testbench was created which would successively apply more noise to the radio signal, while trying to read data bits from the correlator output. Once a single data bit is read incorrectly, the current noise level is recorded and the test is run again. After a number of iterations, the average noise level is presented as the noise immunity level.

3.1 Radio Link Simulator

This MATLAB script is intended to simulate the spread spectrum data sequence using any PN code, and mixed with additive Gaussian white noise. The signal to noise ratio can be adjusted using the desired_snr parameter. The sampling rate is used at this stage in simulation to ensure each chip with added noise is divisible by the sampling rate, which is a requirement when the radio data is sampled and quantized later on. To simulate the behavior of a real radio link, an automatic gain control unit has also been simulated on the receiving end. The unit works to reduce the input signal amplitude such that peaks remain close to $+/-1$. All figures presented later in the report showing a simulated radio link have been generated using this script.

```
% Enter various constants for the simulation
sample_rate = 5;                % Multiple of the chip rate, depends on
    correlator design
desired_snr = -4;                % In decibels
data = [1 1 -1 -1 -1 1 1 -1 -1 -1 -1 1 1 1];   % Data sequence to
    transmitt
noise_resolution = 100;          % How many samples of noise for every
    sample of data

%% Generate sequences
PN_Code = [ 1 zeros(1,7) -1 zeros(1,6) 1 zeros(1,5) -1 zeros(1,4) 1 zeros
    (1,3) -1 zeros(1,2) 1 0 -1 ];

% Create the reference code used in the correlator
% Expand the data and PN code to match the sampling rate
pn_code = reshape((PN_Code' * ones(1, sample_rate))', 1, length(PN_Code)*
    sample_rate);
tx_data = reshape((data' * pn_code)', 1, length(pn_code) * length(data));
% Expand the tx_data so additional noise samples can be added
tx_temp = reshape((tx_data' * ones(1, noise_resolution))', 1, length(
    tx_data)*noise_resolution);

% Create noise and scale to match desired SNR
noise = randn(1, length(tx_data)*noise_resolution);
noise = noise / norm(noise) * norm(tx_temp) / 10.0^(0.05*desired_snr);
% Noise is assumed to be added to the signal. The receiver will likely
    have
% an Automatic-Gain-Control stage, normalizing the signal amplitude
radio_data_noise = tx_temp + noise;
radio_data = radio_data_noise * (1- var(abs(radio_data_noise)));
% Verify that the SNR is as desired
radio_snr = 10 * log10(mean(tx_temp.^2) / mean(noise.^2));

% rx_data is radio_data sampled at every x sample.
rx_data = radio_data(1:noise_resolution:length(radio_data));
```

3.2 Correlator Simulator

The received radio data is sampled by a simulated ADC, of selectable resolution and sampling rate. This is done by defining the ADC intervals first, such that the space $[-1, 1]$ is split into $(2^{\text{ADC bits}})$ discrete values. Next a new vector is defined in such a way that a series of comparisons can be done to check which discrete value to fixate a given sample to. The vector used for comparison is the interval $[-1, 1]$, first split into $2^{\text{ADC bits}} + 1$ values, and then having the lowest value removed from the set.

```
function [ rx_data ] = variable_bit_quantize( input_vector , ADC_bits )
%VARIABLE_BIT_QUANTIZE Quantize vector into discrete values in [-1, 1]

    % Space between each ADC level
    adc_interval = 2 / ((2^ADC_bits)-1);
    % Vector containing valid ADC values in [-1,1]
    adc_span = (0:adc_interval:2) - 1;
    % Vector used to place ADC values
    adc_interval = 2 / (2^ADC_bits);
    adc_check = (0:adc_interval:2) - 1;
    adc_check = adc_check(2:end);

    % Next the data is quantized to allowable levels
    rx_data = zeros(1, length(input_vector));
    for i = 1 : length(input_vector)
        % Quantize each sample
        for j = 1 : length(adc_check)
                if input_vector(i) < adc_check(j)
                    rx_data(i) = adc_span(j);
                    break;
                end
        end
        if j == length(adc_check)
            rx_data(i) = adc_span(end);
        end
    end
end
```

The correlation itself is calculated using the MATLAB function xcorr, taken between the sampled and quantized rx_data and the PN code.

```
% Cross correlation of radio data and PN code
r1_xy = xcorr(rx_data , pn_code);
L = length(rx_data) + length(pn_code) - 1;
start_corr = length(r1_xy) - L;
r1_xy = r1_xy(start_corr:end);
```

3.3 Noise Immunity Testbench

In an attempt at creating an objective measure of the noise immunity possessed by each correlator design, an algorithm has been developed to subject a given mathematical correlator model to an increasingly noisy signal, while trying to read data from it based on the correlation peaks. The correlator models themselves will be presented in chapter 5.2 on page 49.

Data is read using a threshold detector. If a correlation peak is detected above the upper threshold it counts as +1 (representing databit 1), and below the lower threshold as −1 (representing databit 0). Once a data bit has been detected the algorithm refuses to detect a new data bit until a certain timeout has passed. This timeout is proportional to the length of the PN code, as it is known the interval between data bits is at least the length of the PN code. Due to the effects of noise, the actual timeout is a value slightly lower than the PN code length.

```
function [ detected_data ] = detect_correlator_peaks( correlation_vector ,
      pn_code_len , upper_thres , lower_thres , sample_rate )
%DETECT_CORRELATOR_PEAKS Detects peaks corresponding to data bits.

% Find peaks in the correlator output above or below certain thresholds.
% Translate each into a data bit, before engaging a timeout.

    % The minimun expected time between peaks, given in samples
    TIME_OUT_THRES = floor( 0.7 * (sample_rate * pn_code_len));
    timeout = 0;
    % Create the data vector, will at most find a peak every time
         possible,
    % this vector can be shortened later by removing leftover zeros
    detected_data = zeros(1, ceil(length(correlation_vector) /
        TIME_OUT_THRES));
    data_index = 1;
    % Run through the correlation vector
    for j = 1:length(correlation_vector)
        % if the timeout is inactive, look for a peak
        if (timeout ~= 0)
            timeout = timeout - 1;
        else
            % Look for positive bit
            if (correlation_vector(j) >= upper_thres)
                detected_data(data_index) = 1;
                data_index = data_index + 1;
                timeout = TIME_OUT_THRES;

            % Look for negative bit
            elseif (correlation_vector(j) <= lower_thres)
                detected_data(data_index) = -1;
                data_index = data_index + 1;
                timeout = TIME_OUT_THRES;
            end
        end
    end
    % Trim zeros from the detected data vector
    zero_index = find(detected_data==0, 1, 'first');
    if (zero_index == 1)
        disp('No data detected!')
        detected_data = 0;
    else
        detected_data = detected_data(1:zero_index-1);
    end
end
```

The algorithm runs a number of passes before averaging the final results, and also calculating the standard deviation of these. The number of passes to run is user selectable, and each pass is independent of the others. Because of this, it is trivial to parallelize the computations in order to save processing time. During a pass, the noise level is first increased from zero in rough steps until errors are detected in the data bits read from the correlator output. Once this happens the noise level is reduced by two rough steps, and the increment level set to fine. The algorithm runs again closer to the solution, but at a finer resolution. Once an error is detected during this pass the signal conditions are stored, and the algorithm returns to complete a new pass. Once the desired number of passes have been reached, the algorithm calcualtes the mean and standard deviation, presenting the average noise level required before more than $\frac{1}{1000}$ data bits are erroneous.

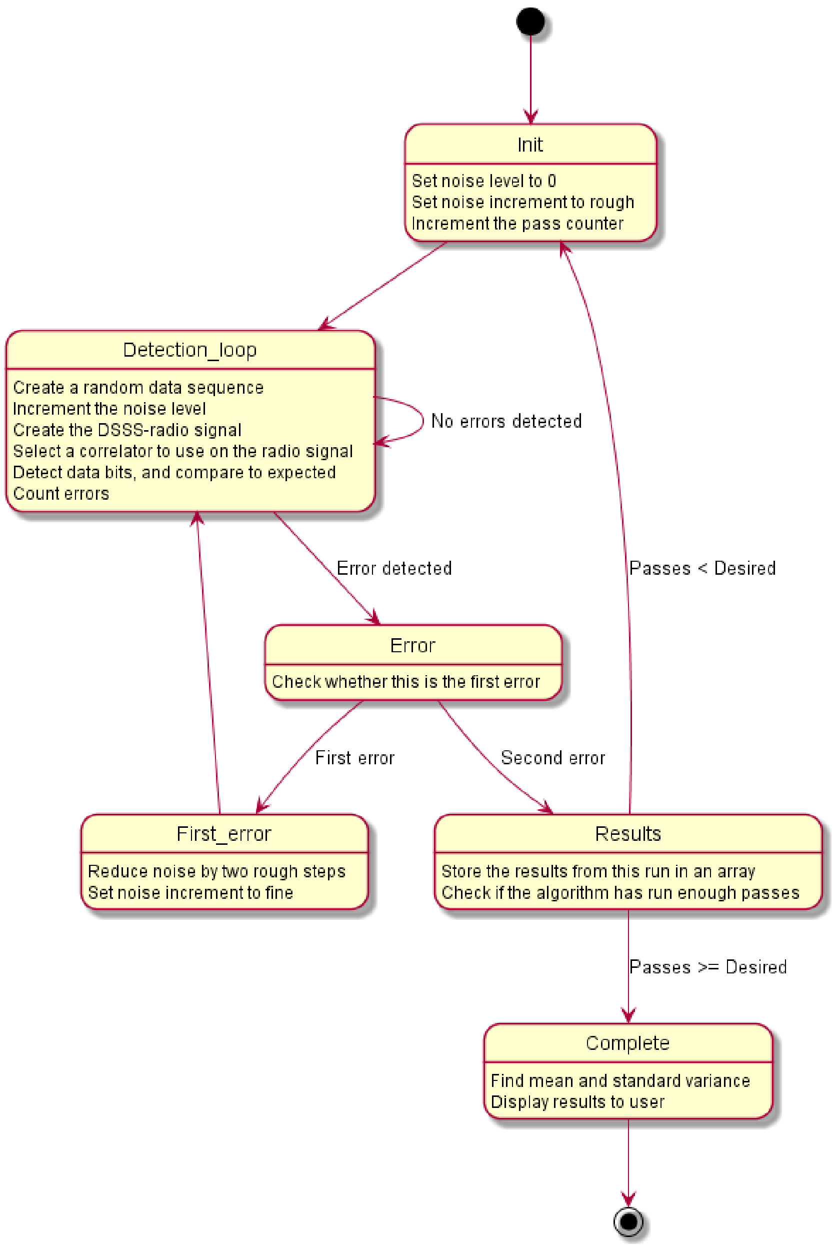
Init
Set noise level to 0
Set noise increment to rough
Increment the pass counter
Detection_loop
Create a random data sequence
Increment the noise level
Create the DSSS-radio signal
Select a correlator to use on the radio signal
Detect data bits, and compare to expected
Count errors
No errors detected
Error detected
Passes < Desired
Error
Check whether this is the first error
First error
Second error
First_error
Reduce noise by two rough steps
Set noise increment to fine
Results
Store the results from this run in an array
Check if the algorithm has run enough passes
Passes >= Desired
Complete
Find mean and standard variance
Display results to user

Figure 3.1: State diagram of the noise immunity algorithm.

Chapter 4

Architectural Design

4.1 One-bit Correlator Implementation

This section contains the implementation details of the 1-bit correlator designs. Each of these correlators is intended to be used with a simple threshold ADC, which detects whether the input signal is above or below some midpoint, represented by 1 or 0 respectively. All of the correlators presented have been designed to use the 11-chip long Barker sequence as the PN code, although the architectures presented can be used with any PN-code, with minor changes. The Barker-11 code is as follows: [1 1 1 -1 -1 -1 1 -1 -1 1 -1].

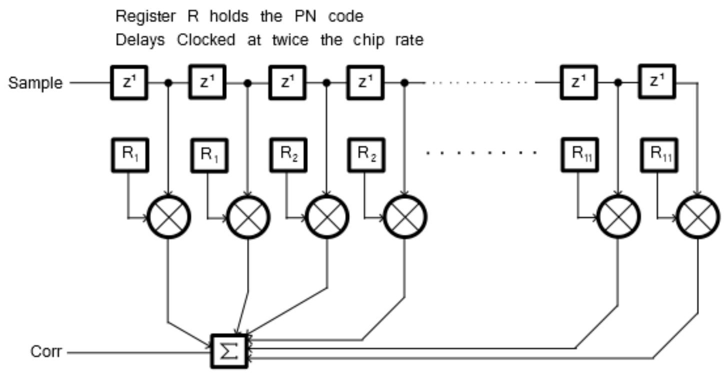
Register R holds the PN code
Delays Clocked at twice the chip rate
Sample
z^{-1}
R_1
R_2
R_{11}
Σ
Corr

Figure 4.1: Matched filter schematic, sampled at twice the chip rate.

The correlator designs are all variations of a matched filter correlator, in which the ADC samples are moved across a multiplier chain representing the flipped/mirrored version of the pseudo random sequence used. As each chip in the sequence is either +1 or -1, the multiplications only alter the sign of the ADC value. In a 1-bit system, this multiplication can be performed with a buffer for +1, and an inverter for -1. +1 is represented by 1, and -1 by 0 in the following designs.

Once each bit in the ADC register has been multiplied with the corresponding chip value, it is summed with the others. This total sum is the correlation between the value currently held in the ADC register, and the PN code. Regardless of the architecture used in the correlator designs, every design requires a memory element to hold some of the previous ADC samples, and some means of multiplying them with the correct portion of the PN code. Currently this is done using a long shift register, and a number of buffers and inverters. Summing the set bits in an efficient manner is where area savings can be made, without sacrificing functionality. This is where each design presented below differs.

To improve noise immunity, the apparent length of the PN code can be increased by oversampling a number of times, defined here as N, for each chip. This increases the PN code length by a factor of N. A value of $N = 1$ is defined here as meaning a sampling rate equal to the chip rate, not twice the chip rate which would be the Nyquist frequency. The correlator shown in figure 4.1 has $N = 2$, notice how pair of samples are multiplied with the same chip in the PN code.

4.1.1 Standard Output

The correlator designs in this section provide an output signal which is can be expressed as the sum of the Kronecker delta functions, taken on pairs of ADC samples and PN codes. The mathematical formula for the estimated correlation at a discrete time instant n is given by:

$$\delta_{i,j} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

$$Corr[n] = \sum_{i=0}^{N \cdot L} \delta_{ADC[i],\text{PN Code}[i]} \tag{4.2}$$

Where the ADC is the ADC sample register, and PN Code is the expanded

Barker-11 sequence, with $+1 = 1$ and $-1 = 0$. L is the length of the PN code used, which in this case is 11, and N is the samples per chip taken by the ADC. The number representation used is offset binary, meaning a signal with the greatest degree of negative correlation (-1.0) is represented by 0, and a signal with the highest degree of correlation $(+1.0)$ is represented by $N \cdot L$.

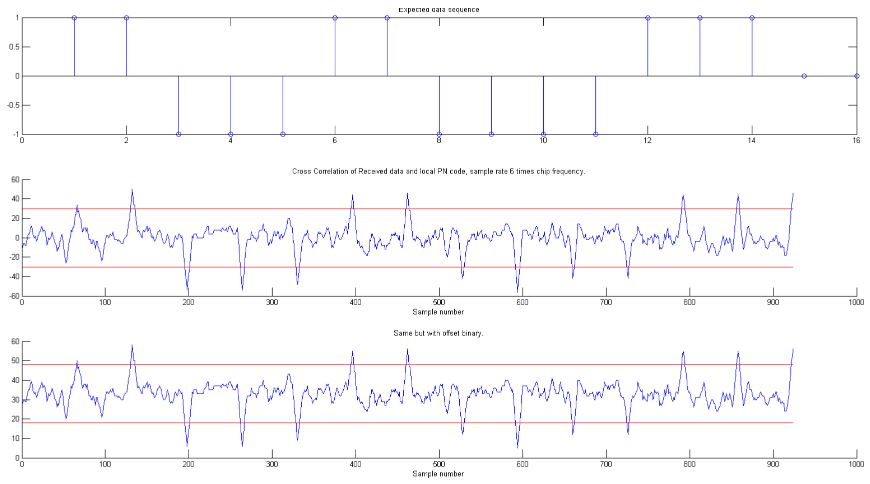

Figure 4.2: Comparison of signed and offset binary correlator output.

An example of the output signals are shown in figure 4.2. First the data sequence is provided, then the two correlation results, in either signed or offset mode. The two output signals are identical in terms of information, as the offset mode is merely the signed signal plus a constant factor of $\frac{N \cdot L}{2}$. In figure 4.2 this offset equals 33 from $N = 6$ and $L = 11$.

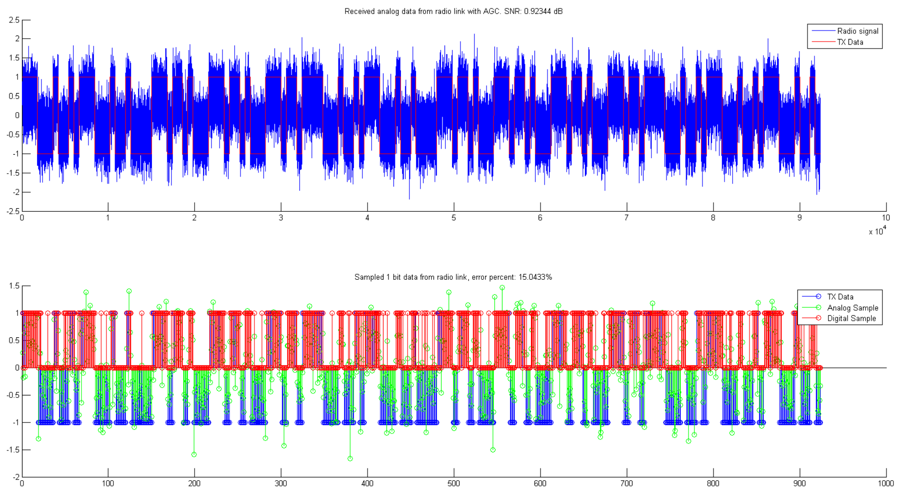

Figure 4.3: Radio signals and ADC values used to create the waveforms in figure 4.2.

The radio signal used to extract ADC samples from is shown in figure 4.3, The SNR is $0.92dB$, and the sign of the sampled ADC value differs from that of the data bit in 15.0% of the samples. From figure 4.2 it is clear that the correct data sequence can still be retrieved error free.

Parallel Correlator, reference

The first of the correlator designs, this one has been deemed the reference design for later comparisons. This is because it is easy to scale, intuitive in its architecture, and provides the result expected from the definition in formula 4.2. It also provides the correlation result after a single clock pulse, assuming the propagation delay is less than the clock period. This allows for high chip rates as the ADC can operate at the same frequency as the global clock.

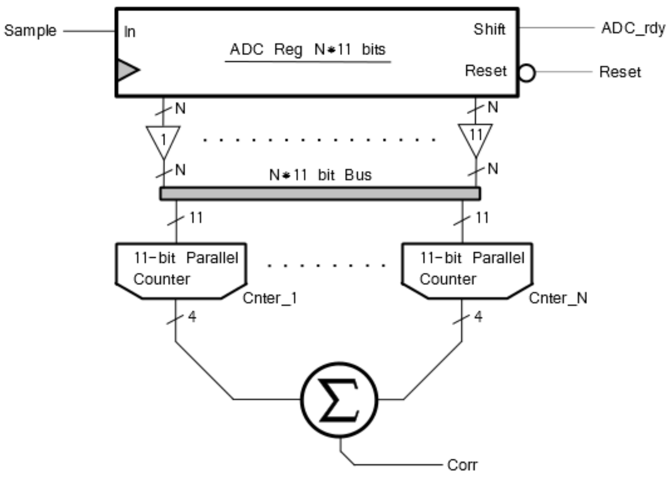

Figure 4.4: Schematic representation of the 1-bit parallel correlator.

From the schematic in figure 4.4 the basic architecture of the correlator can be seen. The ADC samples are shifted into a shift register, and either buffered or inverted, depending on the Barker-11 chip at that location. The multiplication result is summed by N 11-bit parallel counters, which simply count how many set bits there are in a vector (i.e. the hamming weight). The result from the N parallel counters is summed by a final N input adder stage, for the total correlation measure. The parallel counter design is presented in chapter 4.3.2 on page 41.

Parallel Correlator, multiplexed parallel counter

This correlator design is similar to the reference, with the exception being that the 11-bit parallel counter is multiplexed to sections of the multiplication output bus. The result from the eleven bit counter is integrated, by adding it to the output register for each clock pulse. The purpose of this is to reduce the area consumption of the circuit, at the expense of additional time (clock cycles) required to calculate the correlation estimate. The result is ready after $N + 1$ clock cycles. N clock cycles to shift between segments of the multiplication result, and one additional clock cycle for the ADC register to shift in a new sample prior to being multiplexed.

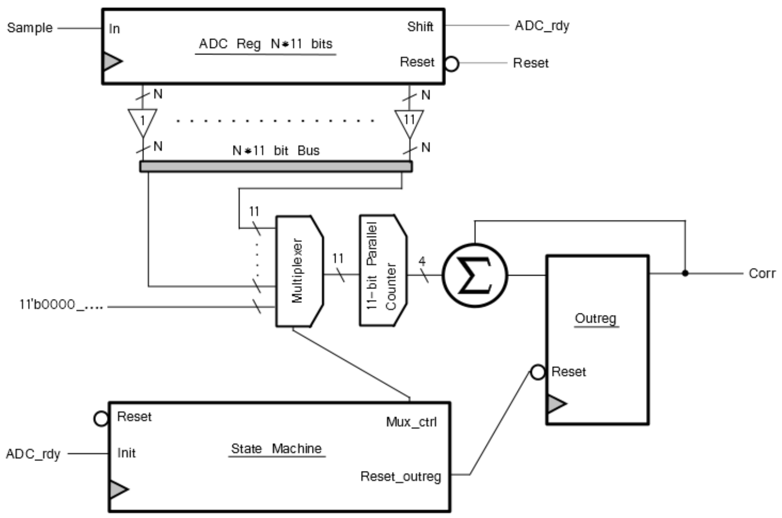

Figure 4.5: Schematic representation of the multiplexed 1-bit parallel correlator.

The assumption behind this design is that a number of 11-bit parallel adders will be larger than the required multiplexer and state machine replacing them. It is also assumed the output register needed to integrate the counter results requires less space than the N 4-bit input adder needed in the reference design.

Dual Correlators

This design takes advantage of the linear properties of the cross-correlation operation. The superposition principle states that for any linear system the net response is the sum of each partial response, had they occurred individually. This can be taken advantage of when determining the correlation estimate, by using several correlators which operate at a reduced sampling rate. If the correlators are given samples in an interleaving manner, then the total correlation estimate will be equal to the sum of all sub correlator outputs. The advantage to this is that the demands on each sub correlator are reduced, at no expense to the total resolution of the system.

Consider a system with M correlators, where the total system samples at a speed of N times the chip rate. Each of the M correlators will then only need to sample at a rate of $\frac{N}{M}$. The correlators are given samples in a way that they are $\frac{360}{M}$

degrees out of phase. For a system with $N = 4$, and $M = 2$, this implies that each sub correlator will sample at twice the chip rate, and 180 degrees out of phase with each other. The correlator system presented below used $M = 2$, i.e. two sub-correlator units.

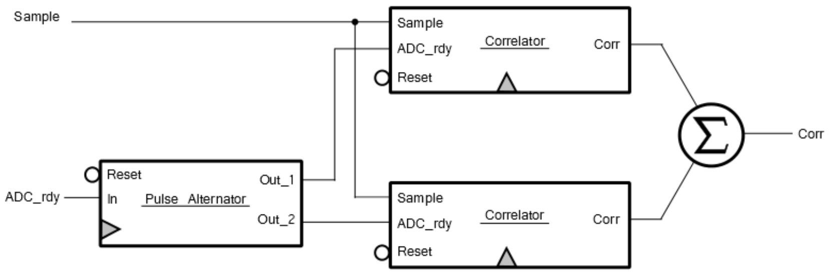

Figure 4.6: Schematic representation of the dual correlator.

Implementing a multi correlator requires very little overhead, the only demands are summing of the sub correlator outputs (and potentially holding their value if this is not done already) and a system to distribute samples to the correct correlator at each instant n. For a system with two sub correlators a single adder and a pulse alternator is all that is needed. The pulse alternator will set one of the outputs high for one clock pulse, following a high input, essentially gating the clock to each unit one at a time. This design assumes the two sub correlators can be implemented in less area than half of the reference design, and also that the overhead required for the pulse alternator is negligible, otherwise area savings will be non-existent.

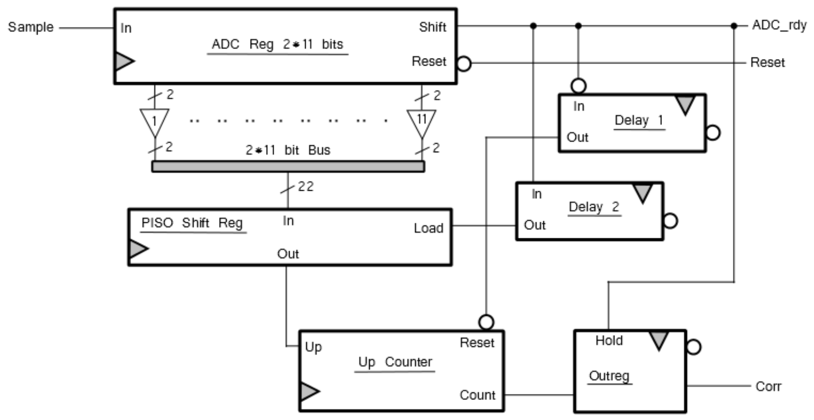

Figure 4.7: Schematic representation of the sub correlator.

The schematic in figure 4.7 shows how the sub correlators were implemented. For testing, the system sampling rate is $N = 4$, so each sub correlator will operate at $N = 2$. A parallel-in, serial-out shift register is used to control an up counter, which essentially counts how many bits are set in the shift register. It takes two clock cycles before the output is latched into the shift register. An additional 22 clock cycles are needed to shift every value out of the shift register, ending in 24 clock cycles needed for the result. The output of the counter is held by an additional register so that the adder used in the dual correlator is able to sum the result when needed. The same functionality can also be achieved using two eleven bit counters and an adder if using the reference design from chapter 4.1.1 on page 18. It was later shown through synthesis that this approach was more effective, see the table 6.2 on page 59.

4.1.2 Reduced Resolution

These correlator designs have had their output resolution reduced by voting on the oversampled values. Ultimately, the value of a chip is what is interesting, and not the value of each individual oversampled value. As such, a majority vote is taken on the oversampled values to estimate what the chip value is. In a system without noise, the vote will logically be the same as the chip value. In a system with noise however, it is assumed that a majority of the oversampled values will have the correct value, and hence the majority vote will yield the correct value in this case also. The resulting signal will swing from 0 for full negative correlation to the length of the PN code (11 in this case) for full positive correlation. In other words there are fewer quantization levels in the signal, and the resolution of the result is independent of the oversampling rate. This is not to say the oversampling rate isn't important, as the more samples there are to vote upon, the more accurate the vote will be.

Voted Correlator

This design operates at $N = X$, and places an X-bit majority voter on each group of oversampled bits, and uses a single eleven bit counter to determine the total correlation. In this architecture the oversampling factor must match the size of the majority voter uses, which limits the oversampling factor to odd numbers only. The output will have a value between 0 and 11. The correlation result is ready in a single clock pulse, just like the reference design. Design of the majority voters is covered in chapter 4.3.4 on page 44. This particular design was synthesized for an oversampling-rate/voter-size of 3 and 5.

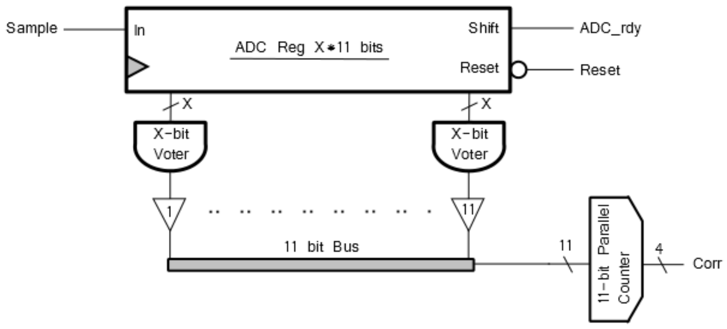

Figure 4.8: Schematic of the five bit voted correlator.

In figure 4.9 the output waveform can be seen in comparison to the standard output. Both correlators sample at $N = 5$ times the chip frequency. Notice how the voted output simply resembles a more heavily quantized version of the full resolution output.

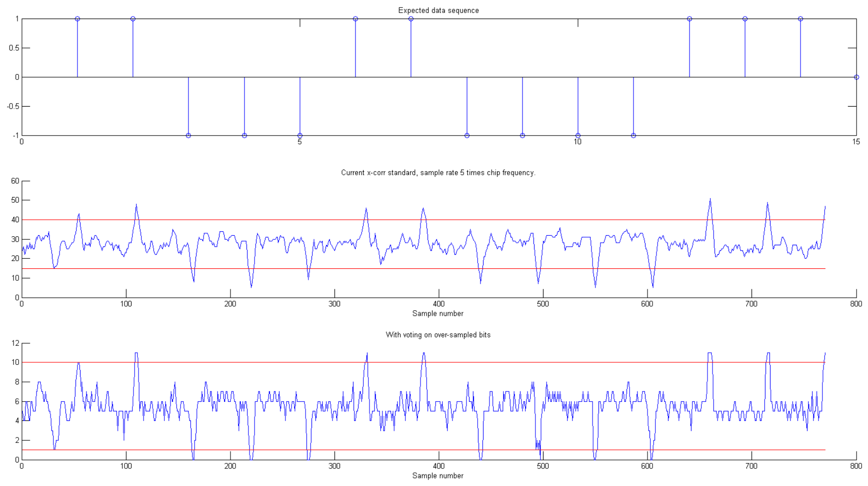

Figure 4.9: Comparison of standard resolution output to the voted output.

Multiplexed Voted Correlator

This is a variation of the voter design, in which the oversampled bits to be voted on are multiplexed to a single 3-bit majority voter. The assumption is that multiplexing the inputs to a single 3-bit voter unit requires less area than simply using the full number of bit voters. It is also assumed the upcounter requires less space than the eleven bit parallel counter. The correlation output is between 0 and 11 and similar to that of the 5-bit voted correlator, i.e. a more discretized version of the standard correlation. The correlation result is ready after 12 clock cycles. Eleven clock cycles to shift the voter between the multiplications results, and one additional clock cycle for the multiplication result to propagate prior to being muxed. This design was only attempted for $N = 3$.

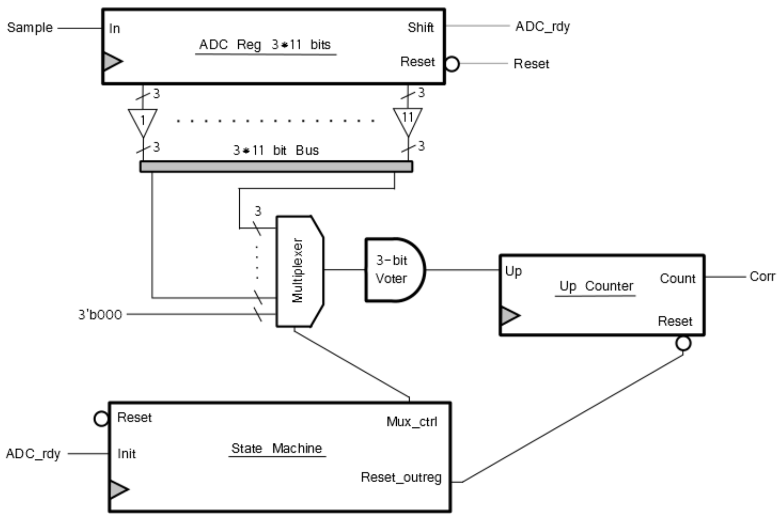

Figure 4.10: Schematic of the three bit voted correlator.

Dual Correlators

See the first two paragraphs in chapter 4.1.1 on page 20 and figure 4.6 on page 21 for the theoretical background on how this correlator works, and the block diagram. The only difference between this design and the one in chapter 4.1.1 is the use of voted sub-correlators operating at $N = 3$. They were either of the type presented in chapter 4.1.2, or 4.1.2. As the sub-correlators run at a sample rate of $N = 3$, the total sample rate of this system is 6 times the chip rate. The output resolution is also doubled, from $0 - 11$ to $0 - 22$.

The actual output from the correlator unit can be seen in figure 4.11 for a simulated radio signal with SNR $0.94dB$, and 14.0% erroneous ADC samples.

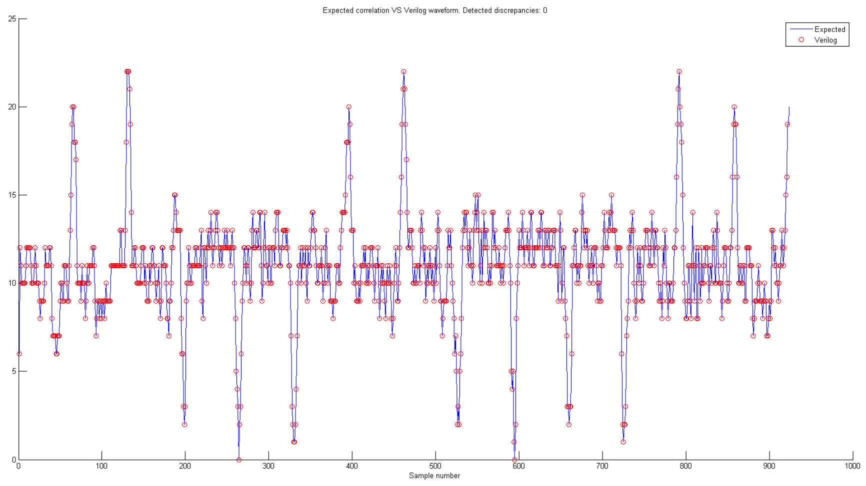

Figure 4.11: Correlation waveform of the dual voted correlator unit.

4.2 Two-bit Correlator Implementation

This section contains the implementation details of the 2-bit correlator designs. Rather than a single threshold comparator, two are now used to detect whether the signal is above a positive midpoint (+1), below a negative midpoint (−1), or in between the two (0). If the current sample is above the positive midpoint a 1 is present on Sample_pos, and a zero on Sample_neg, the inverse is true for a sample below the negative midpoint. In the case that the sample doesn't exceed either threshold, both inputs receive a zero. This is not true 2-bit quantization, as rather than four discrete signal levels, only three are used.

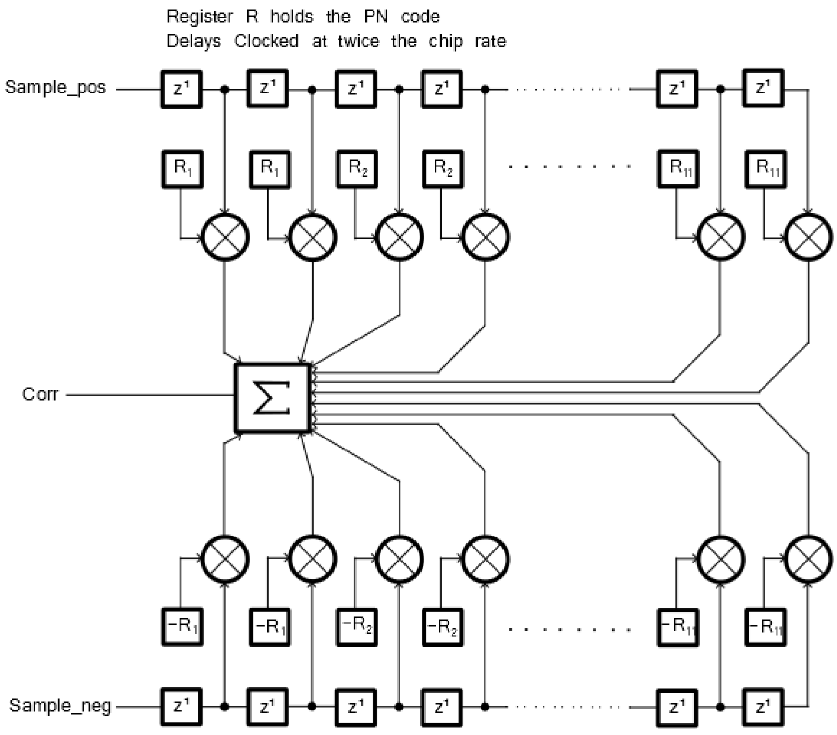

Figure 4.12: Schematic representation of a matched filter, sampled at twice the chip rate.

Doing so opens up for some simplification in implementing the matched filter, as it can be split into a positive- and negative 1-bit correlator, with the total correlation being the sum of the two. The negative ADC register is correlated against the negated PN code, while the positive ADC register is correlated against the unaltered PN-code. See figure 4.12. It follows from this that the same principles introduced in chapter 4.1 on page 15 apply here as well. The output resolution can be both full (standard as it has been called) or reduced by the use of voting. No two-bit designs were created using voting, as simulations showed the noise performance wasn't comparable to that of the full resolution correlators. See table 6.10 on page 64. The use of only three out of four quantization levels available from 2-bits of storage allows for some compression techniques to save register space. This is introduced in chapter 4.2.4 on page 32.

The 2-bit correlator designs presented here all use a 36-chip long chirp code: [1 zeros(7) -1 zeros(6) 1 zeros(5) -1 zeros(4) 1 zeros(3) -1 zeros(2) 1 0 -1]. The zeros in the PN code reduce the amount of counting required to determine the correlation, as the resulting product will simply be zero regardless of the ADC value. This leaves 8 polarity carrying chips in the PN code. Unlike in the 1-bit case where the output resolution for this PN-code would be $N \cdot 8$, it is now $N \cdot 16$. The reason for this is that the span between the number of discrete levels is now 2 (-1,0,1) rather than just 1 (0,1).

4.2.1 Reference Correlator

Like with the 1-bit correlator designs, the reference design uses no multiplexing or other time expanding techniques, and instead computes the result in a single clock cycle. This is achieved by using the full number of parallel bit counters needed, and summing the outputs from these all at once. The size of the X-bit parallel counter seen in figure 4.13 was chosen as 4, 8 and 16. The reason for this was ease of scaling the sampling factor N, and also to see if there is an optimum point for the bit counter size.

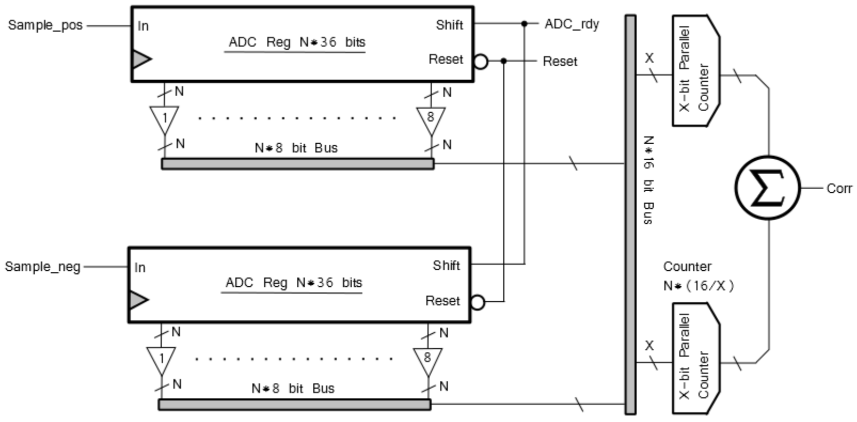

Figure 4.13: Block diagram of the 2-bit reference correlator design.

4.2.2 Multiplexed Correlator

This correlator design operates much like the multiplexed 1-bit correlator. A single X-bit parallel counter unit is used, and sections of the PN-code product bus are multiplexed to its input for counting. When a new ADC sample is ready, the state machine clears the output register, and waits one clock cycle for the ADC register to shift. Then following each clock cycle thereafter, the value in the output register is added to the result from the bit counter, and stored in the output register. Once all sections of the product bus have been counted, the state machine multiplexes the parallel counter input to all zeros before returning to the idle state. In this mode the output is held constant, as it is summed with zero for all the following clock cycles. As was the case for the reference correlator, the unit was designed to use X-bit parallel counters of 4, 8 or 16 bits. The results from the 1-bit synthesis seemed to indicate that the multiplexed design was the most area efficient, so it assumed that a similar approach for two-bit correlator would yield an area efficient design.

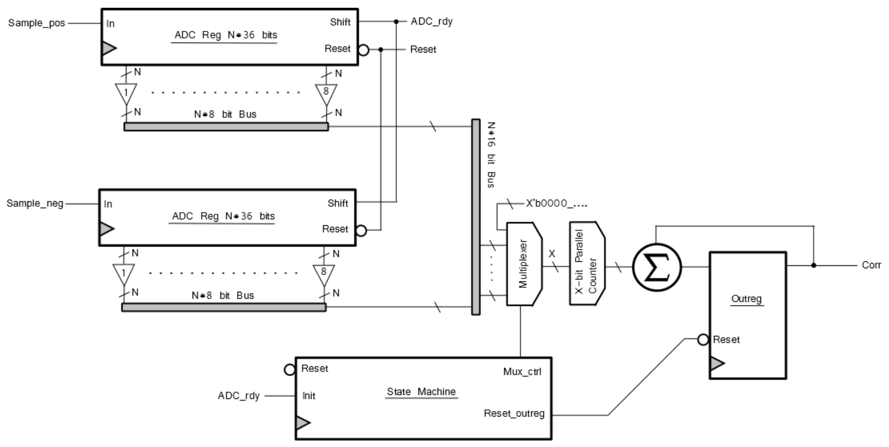

Figure 4.14: Block diagram of the 2-bit multiplexed correlator design.

4.2.3 Multiplexed Sum Correlator

This design is identical to the multiplexed correlator, with the exception being that the full number of bit counters are used instead of just one. The multiplexer selects bit counter results which are added to the output register. The reasoning for this was to reduce the overhead required to multiplex the product bus, as fewer signal lines are needed to multiplex just the result bus. The trade-off here is that the full number of parallel counters is needed, just like in the reference design. This design would show whether the overhead of multiplexing many signal lines cost less than requiring several counter units.

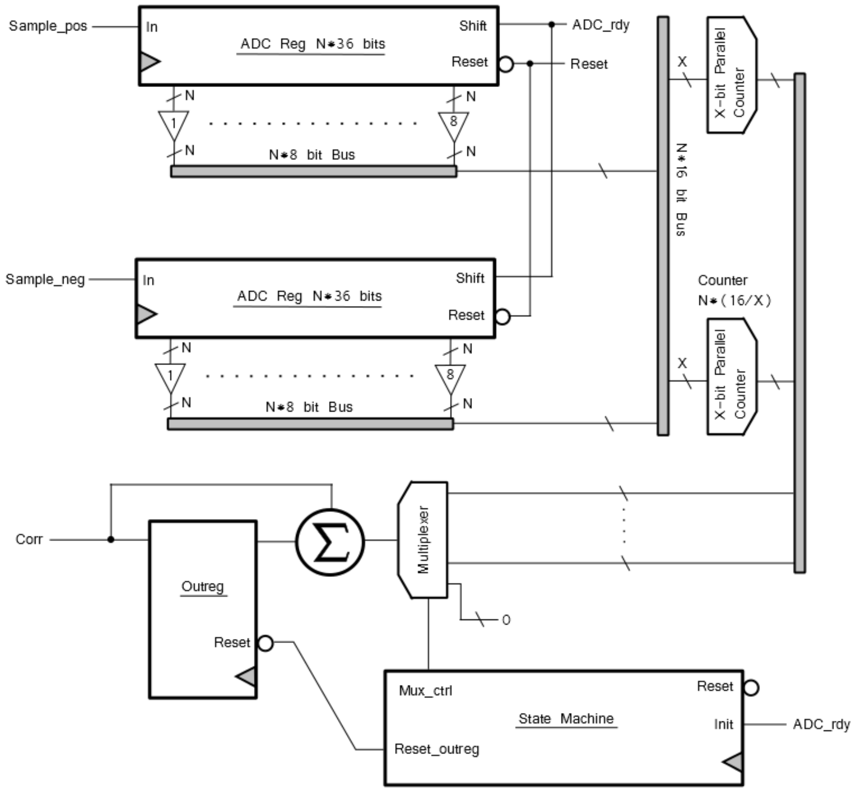

Figure 4.15: Block diagram of the 2-bit multiplexed sum correlator design.

4.2.4 5-bit Grouping Correlator

As was mentioned in the introduction, some storage space goes to waste when only using three of four possible discrete ADC levels. It can be shown that when grouping together three ADC samples, the space required is reduced from 6-bits to 5-bits. Keep in mind that ($3^3 < 2^5$), because 3 samples holding 3 values each results in 27 unique states. A MATLAB script was written to test the potential register savings at various groupings, and it was found that regardless of the group size up to groups of 36, the savings would remain within 0.83 ± 0.04. To avoid potential complexity problems from large group sizes, it was decided to use a group size of 3. The MATLAB code used is shown below.

```
for i=1:36
    states = 3^i;
    un_opt_bits = i*2;
    % Determine needed bits
    roof = 0;
    opt_bits = 0;
    while roof < states
        opt_bits = opt_bits + 1;
        roof = 2^opt_bits;
    end
    percent_saving = opt_bits/un_opt_bits;
    disp(['States: ' num2str(states) ' Unopt Bits: ' num2str(un_opt_bits)
        ' Opt bits: ' num2str(opt_bits) ' Savings: ' num2str(
        percent_saving)])
end
```

In this case, the ADC register can be reduced to $\frac{5}{6}$ of the size required by the other designs. This introduces a new problem however, and that is that samples need to be encoded to a compressed format so they fit in a 5-bit register. The encoding scheme used is presented in the chapter on the proof-of-concept design. The exact encoding scheme used isn't critical to the function of the design, but does have an impact on the logical complexity needed. It was later discovered that the size of the encoding/decoding units was negligible compared to that of the rest of the correlator design (see table 6.9 on page 63), so no further effort was put into finding an optimal encoding scheme.

A second problem is grouping samples together in triples before they can be encoded. In the first two designs presented a new encoded value is calculated for every new sample, thereby eliminating the problem of grouping samples together. This has problems of its own however, namely that each previously encoded 5-bit value has to be changed for every new ADC sample. It has been observed that the information stored in the 5-bit registers repeats itself with a delay of three clock cycles, i.e. the contents of register_1[n] are register_0[n-3], and register_2[n] is simply register_0[n-6]. So far no clever way to exploit this has been found that would not require a large number of additional delay registers. In the final design presented samples are grouped into triplets, before the correlator is allowed to perform calculations on them. By doing this samples can simply be shifted along a 5-bit wide shift register for each new sample. The disadvantage to this is having to determine the correlation at n, n-1 and n-2 each time the 5-bit encoded samples are shifted.

Proof of concept

To test the idea, a proof-of-concept correlator was designed, with no intention of either area or power efficiency. To keep the design simple, the sampling rate N was set equal to the group size of the encoder (3), and a new 5-bit encoded value would be calculated for every new ADC sample. This implied that the each 5-bit register had its own encoder, and a last sample decoder which was used to determine the last sample in time within the 5-bit encoded storage. See figure 4.16. This correlator can determine the correlation in one clock cycle. Using a sampling rate of $N = 6$ would also be possible, as it would only need a doubling of each module. However, as $N = 3$ is easier to implement this was chosen first to see whether the design was promising or not for further work.

A special count decoder block is used to count the correlation of the specific 5-bit value, depending on the polarity of the chip at that location. A count decoder block is only present for 5-bit registers overlapping a 1 in the PN code. For the 36-chip chirp code used here, [1 zeros(7) -1 zeros(6) 1 zeros(5) -1 zeros(4) 1 zeros(3) -1 zeros(2) 1 0 -1], that would imply a positive count decoder on registers; 2, 9, 20, 35, and negative count decoders on registers; 0, 5, 14, 27. (Indexing starting from the right) The remaining registers are not counted. The total correlation is the sum of the count decoder outputs.

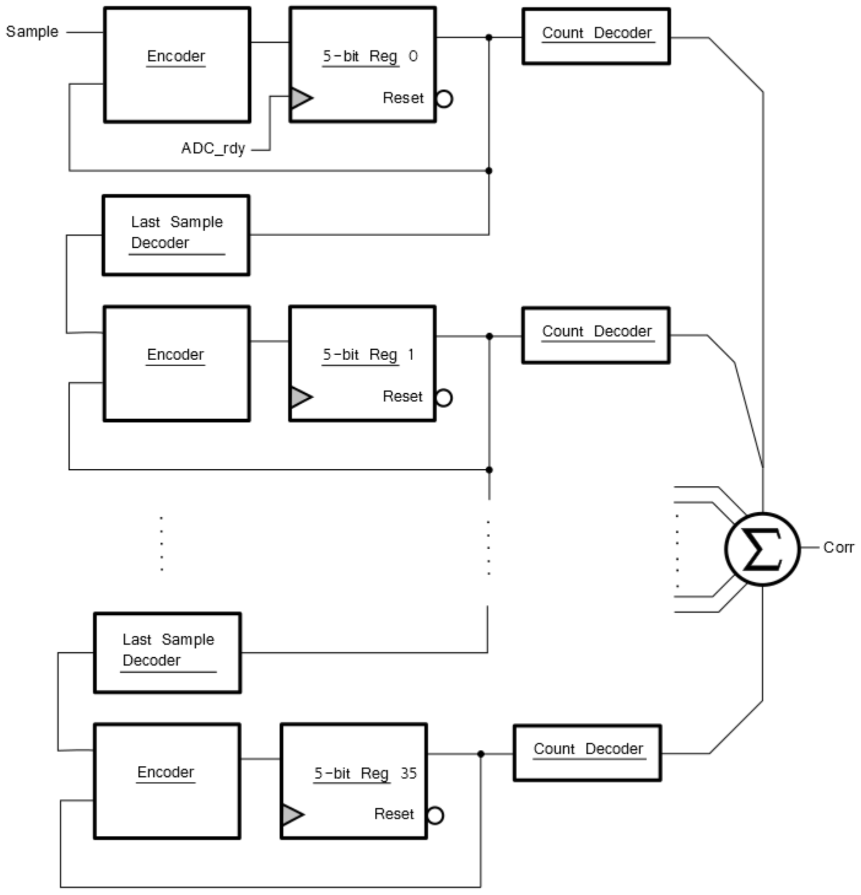

Figure 4.16: Block diagram of the grouping correlator design.

There are over $2 \cdot 10^{33}$ ways to encode 27 values using 5-bits of information, so determining the optimal encoding isn't a trivial task. However, it was observed that the samples could be grouped into a positive and negative sequence, where the MSB of the 5-bit code could be used to signify the polarity. This left 14 different sample sequences to encode into 4-bits of information, or still over $1 \cdot 10^{13}$ encoding schemes. To simplify the logic required to decode the last sample, any sample sequence ending in a 1 was given a LSB of 1, otherwise it was given a

0. The remaining bits in the code were simply counted upwards for each new sample sequence. This was possible for all but one of the sequences ending in a 1, which was assigned an otherwise unused combination of the remaining bits. See table 4.1 for the complete encoding scheme. The choice of sample sequences was somewhat arbitrary, with the only goal being to keep as few samples negative as possible, and also prevent any of the final samples from being negative. The reason for this was to make the sequences more intuitive, in the sense that they can be either positive or negative. It also simplifies the logic for the last sample decoder block.

Sample Sequence	Code	Encoded Value
000	A	0000
001	B	0001
010	C	0100
100	D	0010
110	E	0110
101	F	0011
011	G	0101
111	H	0111
-101	I	1001
0-11	J	1011
-110	K	1000
1-11	L	1101
-1-11	M	1111
-111	N	1100
Unused	-	1010
Unused	-	1110

Table 4.1: Sample sequences and 4-bit (unsigned) encoding. Additional 5th bit sets the sign.

The table above does not include the 5th sign bit. For a negative sequence, e.g. 10-1 (-I) the 5th bit is 1 and the total encoded value is 11001. The corresponding positive sequence -101 (I) is encoded as 01001. From table 4.1 one can see that the magnitude of the last sample in the sequence is easy to detect, being the LSB or the special case of code N. The sign of the last sample is given as the 5th sign bit in the full encoded value. The logical expressions used in the "last sample decoding block" for determining which ADC sample to output are:

$$
\begin{aligned}
Magnitude &= (Bit[3] \wedge Bit[2]) \vee Bit[0] \\
ADC_{pos} &= \overline{Sign} \wedge Magnitude \\
ADC_{neg} &= Sign \wedge Magnitude
\end{aligned}
$$

More generally the expression for the Magnitude can be made independent of the encoding scheme used, by stating the Magnitude is true for sequences; B, F, G, H, I, J, L and N, regardless of the polarity.

The encoding block itself consists of three sub encoders, each assuming the newest input sample is a certain value. The actual value of the newest sample is used to select which of these sub encoders to use when generating an encoded value. When a new sample is present, the current sample sequence is shifted right and the rightmost sample is lost. The newest sample is appended to the beginning of the sequence. Table 4.2 shows all possible sequences, and what the next sequence will be depending on the newest sample received. This table is used to map the encoding scheme from table 4.1 into actual logic for the encoder block.

New +1	New 0	New -1	Previous Sequence
D	A	-D	A, B, -A, -B
F	B	I	C, G, -J
E	C	K	D, F, -I
H	G	N	E, H, -M
-K	-C	-E	I, -D, -F
-I	-B	-F	J, -G, -C
-M	-J	-L	L, -K, -N
-N	-G	-H	M, -E, -H
L	J	M	N, K, -L

Table 4.2: Encoded values given a new sample and previous sequence.

The count decoding block consists of two sub decoders, one for positive polarity and one for negative polarity of the sample sequence. In the event that the chip at the location where the encoded value is being counted is -1, the selector used to choose and output simply reverses its choice. I.e. if the current chip in the PN code is +1, the positive counter is used for a positive sequence (+X) and negative counter for a negative sequence (-X). If the current chip is -1, the opposite is the case. Counting is justified as a +1 sample being worth 2, 0 is worth 1, and -1 is worth 0. From this the values in table 4.3 can easily be deduced.

Sample Sequence	Code	Positive Count	Negative Count
000	A	3	3
001	B	4	2
010	C	4	2
100	D	4	2
110	E	5	1
101	F	5	1
011	G	5	1
111	H	6	0
-101	I	3	3
0-11	J	3	3
-110	K	3	3
1-11	L	4	2
-1-11	M	2	4
-111	N	4	2

Table 4.3: Decoder truth table.

From tables 4.1, 4.2 and 4.3 it is clear that the exact 4-bit encoding scheme used is of little consequence to the function of the correlator, as long as it is mapped correctly to the sample sequences.

Multiplexed Grouping Correlator

Upon seeing that the proof-of-concept grouping correlator worked, an attempt was made to minimize its area consumption. The last sample decoder and encoding block are common components for every 5-bit storage register, and as simple memory-less combinatorial circuits, they are the perfect candidate for multiplexing. To share a single encoder between all of the 5-bit registers, the input of the encoder is connected to the 5-bit storage bus via a multiplexer, and the output connected to every one of the 5-bit registers. A multiplexer is placed on each register input, and used to select either its own contents, or that of the encoder block, when its contents are updated. Every 5-bit register is updated on a rising clock pulse. An alternative method to this would be to gate the clock to each of the 5-bit registers, thereby saving the use of a two input multiplexer. Surprisingly, when studying the schematic from the synthesis results the tool had actually implemented clock gating. However, the schematic is presented using multiplexers due to the way the code was written.

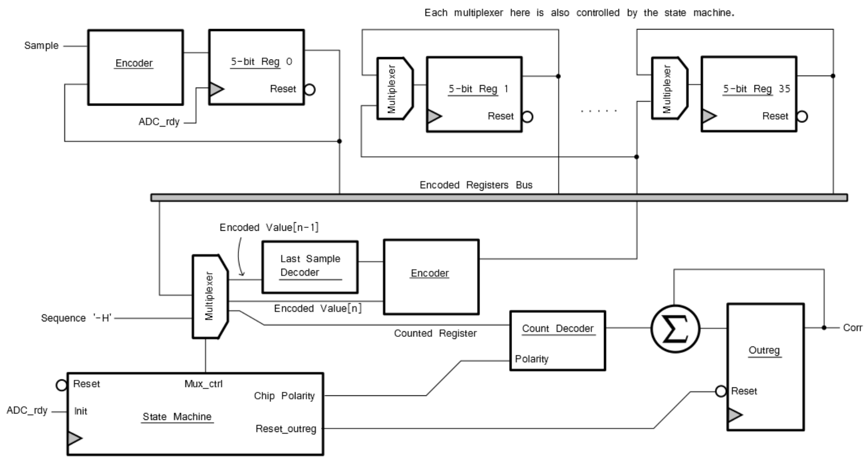

Figure 4.17: Block diagram of the multiplexed grouping correlator design.

The count decoder block is also connected to the 5-bit register bus, and can be multiplexed to the encoded values that need counting, or to the default sequence '-H' in table 4.3, which is decoded as 0 by the count decoder. An integrator is used to sum the correlation, based on the output of the count decoder. A state machine is used to control the multiplexer feeding the encoder, count decoder, and every 5-bit register. This correlator design takes 44 clock cycles to compute the correlation result. 36 clock cycles are needed to update each 5-bit register with a new encoded value, and 8 clock cycles are needed to shift the count decoder to the required locations.

Time Delayed Grouping Correlator

This way of encoding the samples would only require a single encoder unit, and a 5-bit wide shift register. A small state machine would gather three samples into a 6-bit register, before encoding them into a 5-bit value. Once this has been done the new encoded data is shifted into the 5-bit wide shift register containing all of the previously encoded samples. The correlator would then have to compute the correlation for n, n-1 and n-2 while the next three samples are collected. A another small state machine could multiplex the three correlation results to the output on each ADC cycle, meaning the correlation result is effectively delayed by three samples. The main source of complexity in this design would be decoding the correlation count for n, n-1 and n-3. In the proof of concept design, one 5-bit encoded value corresponded to one chip of the PN code, making count decoding easy. In this design the count will need to be decoded by reading the value in two 5-bit encoded values at once, see figure 4.18. This also necessitates the use of one additional 5-bit register to store some of the delayed samples. Some clever encoding could make this easier. E.g simplify the detection of the first sample in the sequence, as was done with the last sample in the sequence for the proof of concept design. Another drawback is the additional logic used to multiplex the three output results, although depending on how the correlation result is used later on, this may not be required.

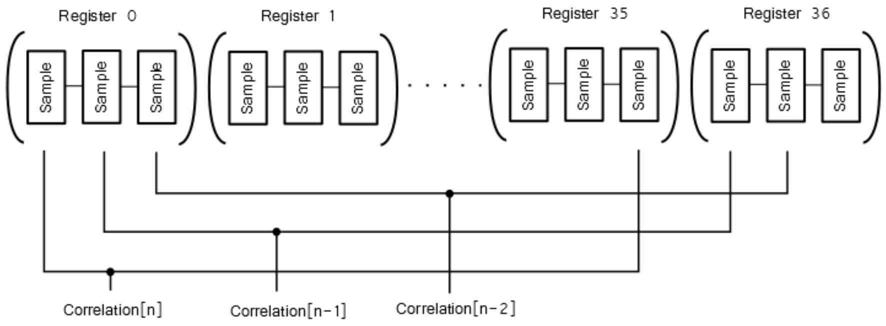

Figure 4.18: Time delayed correlation and samples.

4.3 Hardware Modules

In this section the more specialized hardware units required for the correlator designs are detailed, namely the parallel bit counters and majority voters. An X-bit parallel counter units count how many bits in an X-bit long vector are set. This is also known as the pop-count or hamming weight of the bit vector. For just 2 or 3 input bits, a half or full-adder serves the same function as parallel counter. Synthesis results from previous works seem to indicate that designs utilizing only half and full-adder cells are superior[11], however some alternatives are tried here to test this.

4.3.1 Four bit parallel counter

The four bit parallel counter was designed in three different ways, to both test how the synthesis tool behaved, and also determine which design method was the most efficient in terms of area. The first implementation used a verilog case selector for each of the sixteen input sequences, and a predetermined sum as the output. Essentially the truth table (table 4.4) was written in verilog. This rapidly becomes unpractical as the number of input bits increases.

A	B	C	D	S_2	S_1	S_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	0
0	1	1	1	0	1	1
1	0	0	0	0	0	1
1	0	0	1	0	1	0
1	0	1	0	0	1	0
1	0	1	1	0	1	1
1	1	0	0	0	1	0
1	1	0	1	0	1	1
1	1	1	0	0	1	1
1	1	1	1	1	0	0

Table 4.4: Truth table for the four-bit parallel counter.

The second 4-bit unit was designed using a Karnaugh map based on the truth table, and Reed-Müller logic simplification techniques. The logical functions corresponding to each bit are:

$$S_2 = ABCD$$
$$S_1 = (A + B + C)(A + B + D)(A + C + D)(B + C + D)\overline{ABCD}$$
$$S_0 = A \oplus B \oplus C \oplus D$$

The final implementation of a four-bit counter was made by simply adding the partial sum from two half-adders. The sums from the half-adders is added using a half-adder and full-adder configured as a ripple-carry adder, see figure 4.19.

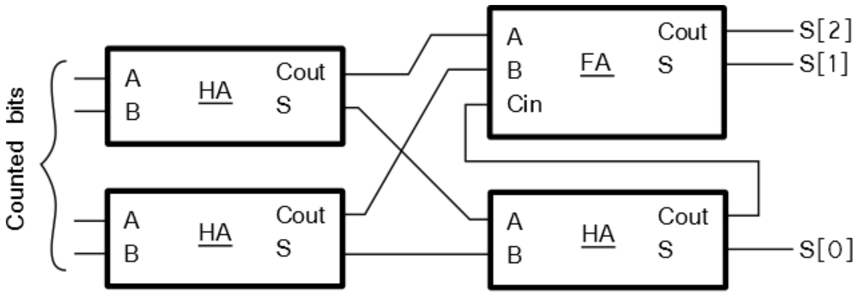

Figure 4.19: Four-bit parallel counter, using half and full-adders.

4.3.2 Eleven bit parallel counter

The reason for designing the unit to count exactly eleven bits is solely because this is the length of the Barker-11 code used as a pseudo random sequence. This implies the length of the ADC register will always be a multiple of 11, which makes scaling the correlator designs easy. To create an 11 bit parallel counter, it was decided to add the sums of a 4-bit and 7-bit parallel counter. The 7-bit unit was made using nothing more than full adder cells, the first two serve as 3-bit counter, and the remaining two as a 2-bit ripple-carry adder. The carry-in input of the adder serves as the 7th input to the parallel counter, see figure 4.20.

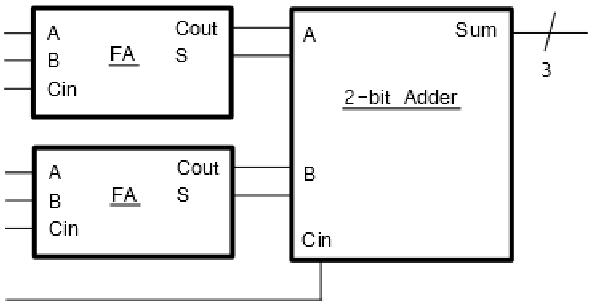

Figure 4.20: Schematic representation of the seven bit counter.

An alternative way to create the eleven bit unit is by solely using full adder cells, see figure 4.21. Like in the seven bit parallel counter one full adder cell is used to gain three more inputs, and the carry input on the 3-bit adder is used as the eleventh input. The 3-bit adder is implemented as a ripple carry adder, where the MSB of one input is always zero. This allows for the final full adder cell to be replaced with a half-adder cell. Synthesis results (see table 6.1) have shown this to be the most area efficient implementation, so it is used as the eleven bit parallel counter in every implementation where it is needed.

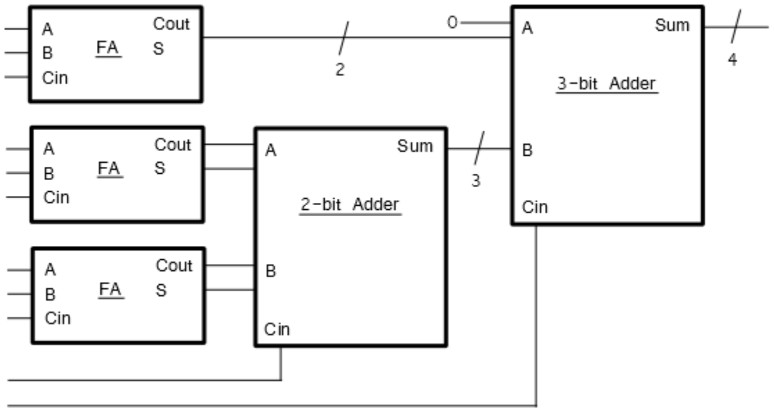

Figure 4.21: Schematic representation of the eleven bit counter.

4.3.3 Other Parallel Bit Counters

Based on the synthesis results for the eleven bit counters, the 8 and 16 bit units used for the 2-bit correlators were constructed using only full and half-adder cells. The 8-bit parallel counter first uses two full-adders and single half-adder to sum to bits. The three sums from these are summed using a carry-save adder (two more full-adder cells), and the result from the carry-save adder is summed using a final ripple-carry adder (one half-adder and one full-adder cell).

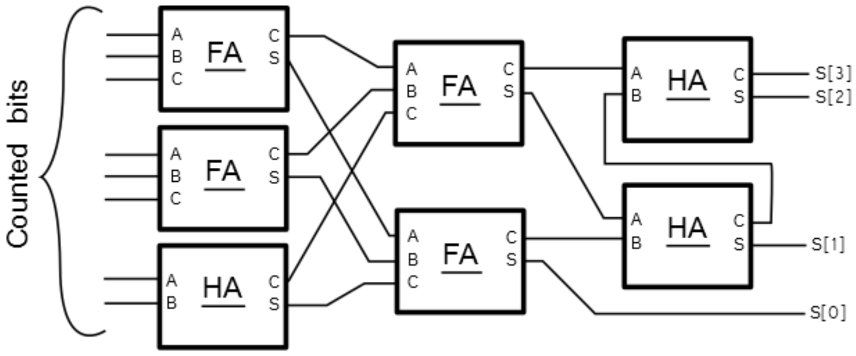

Figure 4.22: Eight-bit parallel counter.

The 16-bit parallel counter consists of three 5-bit parallel counters, which are partially summed using a carry-save adder. A ripple-carry adder sums the carry-save result, and the carry-in input serves as the 16th input of the unit. The 5-bit counters are made using two half-adders and two full-adders.

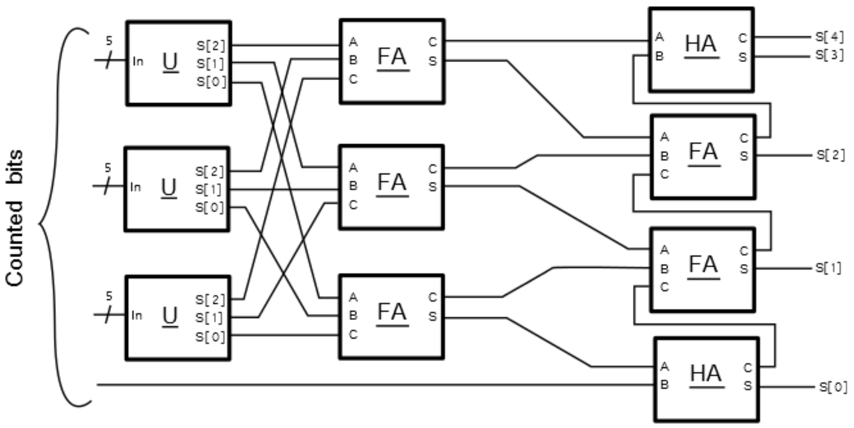

Figure 4.23: Sixteen-bit parallel counter. U is a five-bit parallel counter.

4.3.4 Majority Voters

The majority voter is a special case of an m-out-of-n bit voter, and sets its output equal to the most common input bit. For a system with just three inputs, it is rather trivial to determine the output based on a truth table.

$$Y = AB + BC + CD$$

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 4.5: Truth table for the three-bit majority voter.

However this quickly becomes tedious for larger systems, and even at the next increment of five bits it becomes cumbersome to design the logic in this way. Instead, it has been shown that m-out-of-n bit voters can be designed using a divide and conquer method, which yields lower gate counts than a gate level design.[9] For a five bit system, the majority voter may be designed as in figure 4.24.

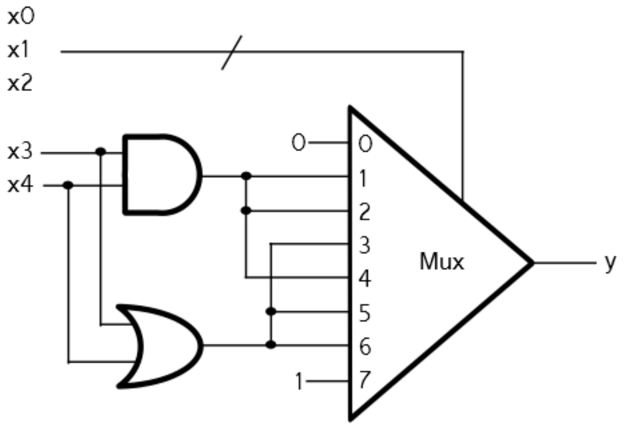

Figure 4.24: Schematic for the 5-bit voter unit, using a multiplexer.

Chapter 5

Logical Verification

Verifying logical correctness of the Verilog modules was done by generating a vector containing simulated ADC values. These values were determined by adding a variable degree of noise to a data packet, and then selecting samples at an interval of N times the chip rate. The selected analog samples were then discretized by either a 1-bit threshold ADC, or two-threshold ADCs in the case of the 2-bit correlators. The simulated ADC values are written to a text file, which is then read by a Verilog testbench. The testbench applies the ADC samples to the correlator, reads the correlator output, and writes the output results to a new text file. This output text file is read by a new MATLAB script, which compares the values from the testbench against those obtained from a mathematical model of the correlator unit in question. Discrepancies between the mathematical model and the results from the testbench are counted, and presented to the user, along with a plot of both waveforms. Figure 4.11 on page 26 is an example of one of these graphs.

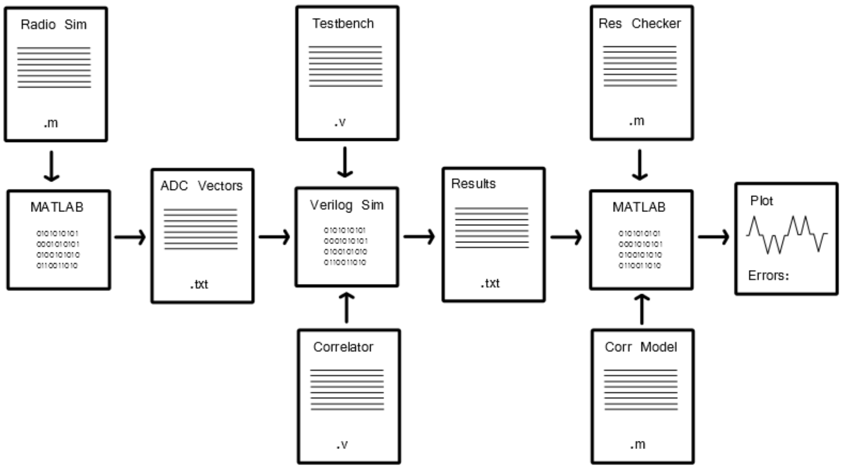

Figure 5.1: Verification process, from test vector generation to logical verification.

5.1 Test vector source

Test vectors are generated as described in chapter 3.1 on page 10. The desired sampling rate is selected, an arbitrary data packet is created, and the noise level is set, all by the user. The only parameter of importance when testing logical correctness is the sampling rate, as the mathematical model compared against will suffer from the same noise, and therefore produce the exact same output if the two models are correct. The sampling rate must be equal to that which the correlator in question was designed for.

The test vectors are not exhaustive; that is they do not check every single possible test vector combination or sequence thereof. The state of the correlator units is determined largely by the values held in the ADC sampling register, which for $N = 4$ is already 44 bits long for the 1-bit correlators, making an exhaustive vector application quite time consuming. Rather, a simulated real-world input sequence is applied to the correlator, which essentially resembles a random stimulus.

5.2 Mathematical Correlator Models

In order to verify the results from the Verilog correlator implementations, the correct output needs to be defined. For each correlator design, a MATLAB function has been created which replicates their behavior.

5.2.1 1-bit Standard

The "standard" correlator output is defined by formula 4.2 on page 16, and is obtained by counting set bits after multiplying the ADC register with the PN code. First the Barker-11 code is expanded with N extra ones for each 1 in the code, and N zeros for each 0. The ADC register is simulated by shifting samples in. Each sample is compared with each bit of the expanded Barker-11 code, and the number of matching bits is the correlation at that sample instant.

```
for k = 1:length(rx_data)
  % Shift in new data
  input_reg(2:end) = input_reg(1:end-1);
  input_reg(1) = rx_data(k);

  % Correct bit counts 1, wrong bit counts 0
  for l = 1:length(pn_code_1bit)
    if (pn_code_1bit(l) == input_reg(l))
      output_correct(k) = output_correct(k) + 1;
    end
  end
end
```

5.2.2 1-bit Voted

Like with the standard output samples are shifted into the simulated ADC register. Then groups of N bits are voted on using the mode function in MATLAB, which returns the most common value in a vector. The resulting vote is compared to the Barker-11 chip, and then counted if correct.

```
for k = 1:length(rx_data)
  % Shift in new data
  input_reg(2:end) = input_reg(1:end-1);
  input_reg(1) = rx_data(k);

  % Each chip is oversampled by some factor. A voter is used to determine
  % whether the sample value to use for each chip, before it is
  % correlated. In the end, the final correlation of each chip is
  % checked. Reduces resolution greatly, but should still provide a
  % decent correlation estimate.

  % Vote on, and check each chip
  for l = 1:length(pn_code_small)
    % Group together the samples, and vote.
    chip_group = input_reg((l-1)*sample_rate_ref + 1 : l*sample_rate_ref)
        ;
    voted_chip = mode(chip_group);

    % Sum up the sub correlations
    if (pn_code_small(l) == voted_chip)
      output_correct(k) = output_correct(k) + 1;
    end
  end
end
```

5.2.3 1-bit Dual Voted

The intended output here is the sum of several (M) voter units, which is set depending on the size of the sub correlator units, and also the sampling rate N. The size of the sub correlator has been set to $N = 3$, but can be set to any odd number if desired. The input ADC samples are split into M separate vectors, each containing a downsampled version of the original radio signal, but with different phase offsets. If the M vectors were interleaved again the resulting vector would be the original received data vector. The M sub correlators are fed their respective vectors of ADC data, and their output is determined as described in chapter 5.2.2. The output created by each sub correlator is up-sampled, shifted by the same phase offset as their input vector had, and finally summed together trivially.

```
% The Barker code
barker_11_1bit = [1 1 1 0 0 0 1 0 0 1 0];

% Stuff for the multi corr
sample_rate_mini = 3;          % Stuck at 2, due to clock constraints
mini_corr_units = sample_rate_ref / sample_rate_mini;   % How many mini
    corrs
pn_code_1bit_mini = reshape((barker_11_1bit' * ones(1, sample_rate_mini))
    ', 1, length(barker_11_1bit)*sample_rate_mini);
pn_code_1bit_mini = fliplr(pn_code_1bit_mini);

% Downsample the received data into two registers, each holding unique
    data
sub_corr_length = length(rx_data)/mini_corr_units;
downsampled_rxdata = zeros(mini_corr_units, sub_corr_length);
for j = 1:mini_corr_units
  downsampled_rxdata(j, 1:end) = rx_data(j : mini_corr_units : end);
end
% Sub correlation registers
sub_corr = zeros(mini_corr_units, sub_corr_length);
% Input registers for each sub correlator
```

```
input_reg_mini = zeros(2,mini_corr_units,length(pn_code_1bit_mini));

pn_code_small = fliplr(barker_11_1bit);
% Run through every input sample
for k = 1:sub_corr_length

  % For each sub correlator
  for j = 1:mini_corr_units
    % Shift in new data
    input_reg_mini(j, 2:end) = input_reg_mini(j, 1:end-1);
    input_reg_mini(j, 1) = downsampled_rxdata(j, k);

    % Vote on, and check each chip
    for l = 1:length(pn_code_small)
      % Group together the samples, and vote.
      chip_group = input_reg_mini(j, (l-1)*sample_rate_mini + 1 : l*
          sample_rate_mini);
      voted_chip = mode(chip_group);

      % Sum up the sub correlations
      if (pn_code_small(l) == voted_chip)
        sub_corr(j, k) = sub_corr(j, k) + 1;
      end
    end
  end
end
% Up sample each mini-corr...
mini_corr = zeros(mini_corr_units, sub_corr_length*mini_corr_units);
for j = 1:mini_corr_units
  mini_corr(j, :) = reshape((sub_corr(j, 1:end)' * ones(1,
      mini_corr_units))', 1, sub_corr_length*mini_corr_units);
end
% ...shift them apart...
for j = 2:mini_corr_units
  mini_corr(j, j:end) = mini_corr(j, 1:end-(j-1));
  mini_corr(j, 1:(j-1)) = 0;
end
% ...and add together
output_correct = zeros(1, sub_corr_length*mini_corr_units);
for j = 1:mini_corr_units
  output_correct = output_correct + mini_corr(j, :);
end
```

5.2.4 2-bit Standard

The standard 2-bit correlator output is similar to the 1-bit model, with the exception that the ADC register holding the positive samples is correlated against the positive PN code, and the register holding the negative samples is correlated against the negative PN code. The sum of both of these is the total correlation.

```
for k = 1:length(rx_data)
  % Shift in new data
  input_reg(:, 2:end) = input_reg(:, 1:end-1);
  input_reg(:, 1) = rx_data(:, k);

  % Correct bit counts 1, wrong bit counts 0
  for l = 1:register_len
    if (pn_code_expanded(l) ~= 0)
      if (pn_codepos_expanded(l) == input_reg(1, l))
        output(k) = output(k) + 1;
      end

      if (pn_codeneg_expanded(l) == input_reg(2, l))
        output(k) = output(k) + 1;
      end
    end

  end
end
```

5.2.5 2-bit Voted

The two bit voted model was only used for noise simulation purposes, and as such the output will vary from -8 to +8, depending on the current correlation. Samples are grouped together and voted on, before being checked against the current chip in the PN code. Voted groups equal to the positive PN code count positive one, and groups equal to the negative PN code count minus one.

```
for k = 1:length(rx_data_merged)
  % Shift in new data
  input_reg(2:end) = input_reg(1:end-1);
  input_reg(1) = rx_data_merged(k);

  % Correct bit counts 1, wrong bit counts 0
  for l = 1:register_len
    if (PN_Code(l) ~= 0)
      % Create voted outputs
      chip_group = input_reg(1, (l-1)*sample_rate_ref + 1 : l*
          sample_rate_ref);
      voted_chip = mode(chip_group);

      if (PN_Code(l) == voted_chip)
        output(k) = output(k) + 1;
      end

      if (-PN_Code(l) == voted_chip)
        output(k) = output(k) - 1;
      end
    end
  end
end
```

5.3 Verilog Testbench

The Verilog testbench is used to apply defined stimuli to the correlator units and record the resulting output. This is done each time an ADC sample is loaded. The same testbench is used for every correlator design, with the only changes being how long the delay between an input is applied and the respective output arriving. This varies from instantly to several ADC samples later, depending on the correlator architecture. Every implementation has the same inputs and outputs, as seen in figure 5.2. To facilitate re-usability of the testbench, parameters sampling rate, output bits and correlator type can be set. A 50MHz global clock is applied to the designs, and the chip rate is assumed to be 1MHz in most cases.

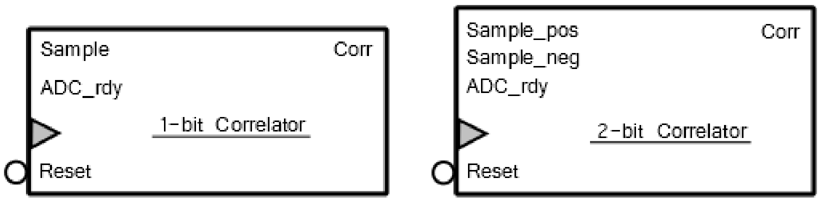

Figure 5.2: Block diagram common for each of the correlator units.

The parameter sampling rate sets how many clock cycles to wait before applying a new ADC sample, with a sampling rate of 1 this is every 50 cycles, and with 2 every 25, etc. The number of clock cycles to wait is rounded up. For some of the correlator designs a set number of clock cycles is used between ADC samples instead, this is typically for designs which would not otherwise be able to complete the correlation calculations before a new sample arrives. The parameter output bits is set to match the number of bits in the correlator output, which for most designs depends on the sampling rate. The final parameter correlator type is used to set which mathematical model the MATLAB script will compare the results against later on. The choice of 50MHz global clock and 1MHz chip rate is based on the expected real world values the correlators may be subjected to, determined in cooperation with the external supervisor.

5.4 Check Testbench Result script

This MATLAB script reads a text file containing input test vectors, and extracts the sampling rate used and the ADC samples to feed into the correlator model. Secondly this script reads the results text file created by the Verilog testbench, where it again extracts the sampling rate used, but also the type of correlator output to compare against, and the outputs created by the Verilog correlator implementation. If the sample rates read from the two input files aren't identical it can be assumed the two files are from different tests and should not be compared, so user is then notified of this so the results can be disregarded. Based on the correlator type read from the results text file, one of the mathematical correlator models is used to generate the expected output sequence. At the same time differences between this sequence and that read from the results text file are detected, and an error counter is incremented. Finally, both the error count and plots of both the expected- and the results sequence are presented to the user for visual inspection. By plotting the results a variety of errors can quickly be identified, such as delayed output or the location of an erroneous value.

5.5 Verification of the Parallel Bit Counters

The parallel bit counters were all verified using the same verilog testbench, which can be adjusted to any size of bit counter by setting the bit width parameter. The testbench applies every vector combination possible at the given bit width, and at the same time asserts that the output of the bit counter is correct. The correct number of set bits is found by using a for-loop to count the number of set bits in currently applied test vector. This testbench is exhaustive in its search, so any logical errors in the bit counter designs are uncovered.

```
always
  #9 begin
    if (currentBitVector == 0) begin
        // Final vector has been tested!
        $display($time, " << Testing complete, %d erros detected! >>",
            errors);
        $finish;
    end else begin
        currentBitVector = currentBitVector - 1;
        // Determine the correct answer
        setbitsCorrect = 0;
        i = 0;
        for (i=0; i<n; i=i+1)
          begin
            if (currentBitVector[i] == 1) begin
                setbitsCorrect = setbitsCorrect + 1;
              end
          end
        //Verify the adder output
        #1 if (setbitsCorrect != setbitsCnt) begin
            $display(" Error for input: %b", currentBitVector);
            errors = errors + 1;
          end
      end
  end
```

Chapter 6

Results

The various properties of the hardware implementations are presented here. All of the modules have been synthesized using Design Vision from Synopsys, E-2010.12-SP5 Version. Scan chains were inserted where possible for testability purposes. It is from this synthesis that numbers for area and power consumption are taken. The results reported here do not include the interconnects required to route the design, which will have a varying degree of impact on both the area and power consumption.

The operating speed in clock cycles has been based on analysis of the verilog code, and confirmed by simulations. The noise immunity numbers were acquired from the testbench described in chapter 3.3 on page 12, applied to the mathematical models presented in chapter 5.2 on page 49. The number of passes used was 20, as a compromise between enough values for a meaningful average, and the run time required to calculate a result.

6.1 Parallel Bit Counters

The area required for the individual parallel bit counter units is presented here. Besides the half and full adder cells, the 4-bit Karnaugh based and the 7-bit counter have the best area per counted bit.

Size and description	Chapter	Area [μm^2]	Area/Bit
2-bit, half-adder	-	16.6	2.1
3-bit, full-adder	-	25.6	8.5
4-bit, selector	4.3.1	69.1	17.3
4-bit, Karnaugh	4.3.1	56.3	14.1
4-bit, HA and FA cells	4.3.1	75.5	18.9
5-bit	4.3.3	84.5	21.1
7-bit, four FA cells	-	102.4	14.6
8-bit	4.3.3	152.3	19.0
8-bit, two 4-bit Karnaugh	-	180.5	22.6
11-bit, with 4-bit Karnaugh	4.3.2	226.6	20.6
11-bit, HA and FA	4.3.2	195.8	17.8
16-bit	4.3.3	414.7	25.9

Table 6.1: Area consumption of the parallel counters.

6.2 1-bit Correlators

The following sub chapters contain the results and properties of the 1-bit correlator designs. In table 6.2 below the correlator designs are presented with a number, description and the chapter describing it. Some of the correlators may not appear to have their own chapter, this is because their design is derivative of the chapter referenced. Units 1 and 4 are examples of this. Unit 4 is the reference design presented in chapter 4.1.1 with $N = 2$. Unit 6 and 7 are the voted correlator with $N = 5$ and 3, respectively. Each correlator unit has been synthesized using the least area-consuming internal components available. In the case of the dual correlators, unit 9 uses design 7 as a sub correlator, and unit 5 uses design 4 as a sub correlator.

6.2.1 Area Consumption

Table 6.2 shows the sequential, combinational and total area required by the designs. The sequential area is the area used by clocked cells.

Number	Correlator Design	Chapter	Total Area [μm^2]	Sequential	Combinational
1	Parallel, reference	4.1.1	2359.0	1226.2	1132.8
2	Parallel, muxed	4.1.1	2255.4	1532.1	723.2
3	Sub corr	4.1.1	2017.3	1712.6	304.6
4	Sub corr, alt	4.1.1	1157.1	634.9	522.2
5	Dual sub corr	4.1.1	2558.7	1365.8	1193.0
6	Voted, 5-bit	4.1.2	2291.2	1520.6	770.6
7	Voted, 3-bit, alt	4.1.2	1319.7	929.3	390.4
8	Voted, 3-bit	4.1.2	1894.4	1438.7	455.7
9	Dual, voted 3-bit	4.1.2	2858.2	1954.6	903.7

Table 6.2: Area consumption of each correlator design.

6.2.2 Noise Immunity

The values in table 6.3 were determined using a rough step size of 0.1, and a fine step size of 0.005. A data sequence length of 1000 bits was used. The noise factor is a scaling constant using to adjust the strength of the noise, and is the parameter which the algorithm increments to determine the noise immunity. This parameter has a lower relative standard variance than the average SNR, and as such should be taken as the key parameter when comparing topologies. A higher noise factor number corresponds to greater noise immunity, which is desirable.

Type	SNR [dB]	Noise factor	σ SNR [dB]	σ Noise factor
S, N=2	6.0679	0.4998	0.7232	0.0396
S, N=3	4.2572	0.6135	0.6028	0.0418
S, N=4	2.8827	0.7190	0.5039	0.0422
S, N=5	2.2487	0.7728	0.4742	0.0411
S, N=6	1.6984	0.8233	0.3656	0.0333
V, N=3	5.6639	0.5208	0.5450	0.0319
V, N=5	3.3261	0.6825	0.3698	0.0293
DV, N=6	2.5077	0.7498	0.4325	0.0373

Table 6.3: Noise immunity testbench results. S = standard, V = voted, D = dual.

The different correlator output types correspond to groups of correlator implementations. Standard, N=2 applies to the sub correlators used in the dual correlator, chapter 4.1.1 on page 20. Standard, N=4 applies to the reference design, the muxed reference design, and the dual voted design in chapters 4.1.1, 4.1.1 and 4.1.1. Voted, N=3 applies to the sub correlators used in the dual voted correlator, chapter 4.1.2. Voted, N=5 applies only to the design in chapter 4.1.2.

Dual voted, N=6 applies only to the design in chapter 4.1.2. Some of the output types do not correspond to a specific implementation, but serve as a comparison point between the voted and regular output types.

6.2.3 Operational Speed

The number of clock cycles required by each correlator design are decided by the architecture. These figures have been determined using logical analysis of the architecture, and confirmed by simulation results. These numbers are for the global clock cycles needed to calculate the result once a new sample has been loaded, i.e. the time from a new sample is loaded until the corresponding output is present. For the dual correlators the time required depends on the sub correlator used, which in both implementations here is a single clock cycle.

Number	Correlator Design	Chapter	Clock cycles
1	Parallel, reference	4.1.1	1
2	Parallel, muxed	4.1.1	5
3	Sub corr	4.1.1	24
4	Sub corr, alt	4.1.1	1
5	Dual sub corr	4.1.1	1
6	Voted, 5-bit	4.1.2	1
7	Voted, 3-bit, alt	4.1.2	1
8	Voted, 3-bit	4.1.2	12
9	Dual, voted 3-bit	4.1.2	1

Table 6.4: Required clock cycles for each design.

6.2.4 Power Consumption

The estimated power consumption of the modules was determined using Synopsys Design Vision. The global operating voltage was set to 1.08V, and the "analysis_effort" option set to "low". The dynamic power resulting from switching activity in the circuit is given in micro-Watts, whereas the static power dissipation from cell leakage current is given in nano-Watts. The most power efficient unit is design 9, the dual voted correlator.

Number	Correlator Design	Chapter	Dynamic Power [μW]	Leakage Power [nW]
1	Parallel, reference	4.1.1	16.1	25.0
2	Parallel, muxed	4.1.1	21.5	27.3
3	Sub corr	4.1.1	24.9	25.1
4	Sub corr, alt	4.1.1	8.5	12.1
5	Dual sub corr	4.1.1	8.4	26.6
6	Voted, 5-bit	4.1.2	20.6	25.0
7	Voted, 3-bit, alt	4.1.2	12.5	15.1
8	Voted, 3-bit	4.1.2	17.2	23.1
9	Dual, voted 3-bit	4.1.2	12.7	32.2

Table 6.5: Power consumption of the 1-bit correlators.

The dynamic power loss greatly dominates the total power dissipated by the modules. This is the power lost from switching activity in the designs, and is proportional to the well-known formula:

$$P_d = \alpha f C V_{dd}^2 \tag{6.1}$$

Where f is the switching frequency, C is the total capacitance being switched, V_{dd} is the supply voltage and α is a factor related to effective number of gates switching.

6.2.5 Summary

A quick side-by-side view of each correlator with all of its key properties.

Number	Correlator Design	N	Chapter	Area [μm^2]	Clocks	Noise	Power [μW]
1	Parallel, reference	4	4.1.1	2359.0	1	0.719	16.1
2	Parallel, muxed	4	4.1.1	2255.4	5	0.719	21.5
3	Sub corr	2	4.1.1	2017.3	24	0.500	24.9
4	Sub corr, alt	2	4.1.1	1157.1	1	0.500	8.5
5	Dual sub corr	4	4.1.1	2558.7	1	0.719	8.4
6	Voted, 5-bit	5	4.1.2	2291.2	1	0.683	20.6
7	Voted, 3-bit, alt	3	4.1.2	1319.7	1	0.521	12.5
8	Voted, 3-bit	3	4.1.2	1894.4	12	0.521	17.2
9	Dual, voted 3-bit	6	4.1.2	2858.2	1	0.750	12.7

Table 6.6: Overall comparison, same numbering as in table 6.2. N is the sampling rate.

6.3 2-bit Correlators

The results presented in the following sub chapters belong to the 2-bit correlator designs. In the table 6.7 below, the numbering of the designs is presented. The four, eight or sixteen trailing some of the design names indicates the size of the parallel bit counter used. The difference between design 11 and 12 is whether the 5-bit registers are implemented using asynchronous latches or synchronous registers. The grouping designs were only implemented for a sampling rate of $N = 3$, due to the nature of the designs, as described in chapter 4.2.4 on page 32.

6.3.1 Operational Speed

The speeds listed here are the number of global clock cycles needed by the design for every ADC clock cycle, in order to compute the result. The factor N is the sampling rate. The operational speed was determined by analyzing the source code and confirmed by simulation.

Number	Correlator Design	Chapter	Clock Cycles
1	Reference, four	4.2.1	1
2	Reference, eight	4.2.1	1
3	Reference, sixteen	4.2.1	1
4	Multiplexed, four	4.2.2	$1 + (N \cdot 4)$
5	Multiplexed, eight	4.2.2	$1 + (N \cdot 2)$
6	Multiplexed, sixteen	4.2.2	$1 + N$
7	Multiplexed Counter, four	4.2.3	$1 + (N \cdot 4)$
8	Multiplexed Counter, eight	4.2.3	$1 + (N \cdot 2)$
9	Multiplexed Counter, sixteen	4.2.3	$1 + N$
10	Grouping	4.2.4	1
11	Multiplexed Grouping, latches	4.2.4	44
12	Multiplexed Grouping, clocked	4.2.4	44

Table 6.7: Number designation and calculation cycles needed.

The fastest designs are the reference designs #1-3 and the grouping design #10, which can be clocked at the same frequency as the ADC. After this the fastest designs are the multiplexed ones using the largest parallel counter.

6.3.2 Area Consumption

The percentages in the column under sampling rate 3 are the sequential area required by each design. Since this figure varies little (±1%) as the sampling rate is changed, it has only been provided for the sampling rate $N = 3$ so all of the correlator designs can be compared. As with the 1-bit designs the interconnect area has not been calculated, so the actual implementations will require routing before the total area can be determined.

		Sampling rate					
		2	3		4	5	6
Design	1	5009.9	7494.4	(78.2%)	9977.6	12436.5	14929.9
	2	4947.2	7337.0	(79.8%)	9850.9	12303.4	14702.1
	3	5013.8	7420.2	(78.9%)	10068.5	12551.7	15073.3
	4	4835.8	6915.8	(89.6%)	9048.3	11157.8	13249.3
	5	4815.4	6909.4	(89.7%)	9038.1	11119.4	13158.4
	6	5049.6	7092.5	(86.9%)	9222.4	11278.1	13304.3
	7	5203.2	7522.6	(82.5%)	9849.6	12144.6	14428.2
	8	5249.3	7633.9	(77.5%)	9976.3	12303.4	14603.5
	9	5409.3	7960.3	(80.8%)	10398.7	12824.3	15251.2
	10	-	27578.9	(22.8%)	-	-	-
	11	-	9026.6	(43.0%)	-	-	-
	12	-	12224.0	(66.0%)	-	-	-

Table 6.8: Total area consumption at various sampling rates, given in [μm^2]

From table 6.8 it can be observed that design #5 consumes the least area for all sampling rates. All of the multiplexed counter designs (#7 through 9) are consistently worse than the reference in terms of area. Design #11 using latches is the smallest of the grouping designs. Table 6.9 below shows the individual area consumption of the components used in the grouping correlator.

Component	Area [μm^2]
Encoder	521.0
Count Decoder	175.4
Last Sample Decoder	17.9

Table 6.9: Individual component area for the grouping correlators.

6.3.3 Noise Immunity

The values in table 6.10 were determined using a rough step size of 0.1, and a fine step size of 0.005. A data sequence length of 1000 bits was used. The noise factor has the lowest relative standard deviation, and as such should be used when comparing the noise results.

Type	SNR [dB]	Noise factor	σ SNR [dB]	σ Noise factor
S, N=2	2.9495	0.3363	0.4270	0.0163
S, N=3	2.0039	0.3750	0.4735	0.0200
S, N=4	0.6793	0.4365	0.4952	0.0254
S, N=5	0.0017	0.4718	0.4549	0.0248
S, N=6	-0.1686	0.4820	0.5747	0.0311
V, N=3	2.7462	0.3443	0.5005	0.0196
V, N=5	0.7527	0.4328	0.3464	0.0174

Table 6.10: Noise immunity testbench results for the 2-bit model. S = standard, V = voted.

The best noise immunity is obtained when using the full resolution output, and a sampling rate of 6. The voted designs are comparable, but still worse than the equivalent full resolution design using a sampling rate of one less (N-1). All of the implemented 2-bit designs utilize the full output resolution, so their noise immunity is given by the S entries, at a given sampling rate. Designs 10, 11 and 12 (the grouping correlators) have a set sampling rate of $N = 3$, making their noise immunity 2.0159dB.

6.3.4 Power Consumption

The estimated power consumption of the modules was determined using Synopsys Design Vision. The global operating voltage was set to 1.08V, and the "analysis_effort" option set to "low". Notice the significant difference in power consumption between the regular designs and the grouping the designs.

		Sampling rate				
		2	3	4	5	6
Design	1	47.8	73.3	94.3	117.5	141.0
	2	47.4	71.8	94.0	117.5	140.5
	3	47.7	72.2	95.1	118.5	142.2
	4	54.8	79.2	104.2	128.5	152.6
	5	54.3	78.9	103.9	128.4	152.6
	6	53.7	78.5	103.0	120.6	151.5
	7	50.2	74.8	95.1	116.9	138.5
	8	49.8	74.3	95.3	117.4	139.6
	9	51.4	77.2	95.8	118.5	141.0
	10	-	2.6	-	-	-
	11	-	13.5	-	-	-
	12	-	13.7	-	-	-

Table 6.11: Dynamic power consumption, given in [μW].

Chapter 7

Discussion

Results and methodology are discussed in this chapter.

7.1 Noise Immunity Testbench

The noise immunity testbench determines a property which is statistical in nature, and as such the final result can only be determined with a finite level of certainty. It is for this reason the standard deviation is provided with each result. The algorithm creates a new random sequence at each noise level, because it is assumed transmitting a random data sequence rather than a predetermined one should not impact the final noise level result. The reasoning behind this assumption is that the memory in the correlator is only as long as the PN code, which again is the length of a single data bit. Thus sending a long sequence of the same data bit value should be no different than sending a continuously toggling sequence, because the memory of the correlator can at most hold a single bit, or half the value of two bits in a sequence, which is negligible compared to the 1000 bits used in the sequences.

The parameter used in the algorithm is the noise level, which is a factor multiplied with the white noise signal, to set the SNR. Since the generated white noise is supposed to be completely random, variations between runs will occur. For example, it is possible that the ADC only samples when the impact from the noise is minimal, or conversely, ever sample is taken during a noise spike. The actual noise level will as such be more or less than the what the noise factor indicates. The algorithm will then assume that the current noise level is acceptable, and move on. The effects of this can be seen from the larger standard deviation in the SNR than that of the noise factor.

A way to seemingly remedy this would be to alter the algorithm such that it operates on SNR rather than the relative noise factor. This was attempted, however the standard deviation of the SNR did not behave in a predictable manner. At some sampling rates the standard deviation would be small, and at others large, with no discernible pattern. As such, the algorithm was left to operate on the noise factor.

7.2 Area results

The synthesis results show that area is largely dependent on the sampling rate, and thus indirectly the noise immunity, for all designs. See tables 6.2 on page 59 and 6.8 on page 63. Multiplexing was used successfully for both the 1-bit and 2-bit designs, and resulted in the designs requiring the least area. It is also worth noting that the most area efficient parallel bit counter (4-bit Karnaugh in table 6.1 on page 58) did not result in the smallest 2-bit correlator design when multiplexed. Instead the 8-bit counter gave the best design in terms of area, presumably due to the reduction in multiplexer and state machine complexity.

From table 6.2 the portion of each design required for the 1-bit ADC registers can be seen. The only sequential elements in the reference design 1 are the ADC registers, which make up roughly half the design. Comparing this to design 4 and 6 which are also without additional sequential elements, the ADC register area alone can be estimated using the formula $N \cdot 306\mu m^2$.

It is important to remember that the synthesis did not perform routing of the designs, and as such the interconnect area has not been included. For the reference designs it is assumed this wouldn't contribute with much additional area, as connections are only one-to-one. For the multiplexed designs this may add some overhead, especially for those using the 4-bit parallel counters which must be multiplexed across a potentially large register. The multiplexed grouping correlators is where interconnect area is assumed to be the most significant, as contrary to the other designs where samples are simply shifted along shift registers, each 5-bit register here requires a multiplexer with a connection to the single encoder block.

7.3 Noise results

The results from the noise immunity test bench are more or less as expected, where the sampling rate is the main deciding factor in the ability of each correlator design to suppress noise. The performance of the voted designs versus those of full resolution seems to show that to achieve the same performance as a full resolution design working at N, a sampling factor of N+1 is needed for the voted design.

The noise immunity results from the 1-bit and 2-bit designs aren't directly comparable to each-other because of the different PN code used. Whereas the 1-bit design used the 11-chip long Barker code, the 2-bit designs used a 36-chip long chirp code. A longer PN code gives better noise immunity in itself, and the additional ADC resolution will also improve the noise immunity. For identical PN codes the noise immunity could be compared across both the 1- and 2-bit designs, but only when using the SNR. The noise factor is merely a relative scalar the algorithm works on, and as can be seen when comparing the results, does not correspond to a given SNR. Consider for example from table 6.3 on page 59 a noise factor of 0.5 causes a SNR of 6.1dB, whereas in table 6.10 a noise factor of 0.5 causes a SNR of −0.2dB. Why this is the case has not been determined at the time of writing this thesis.

A factor which greatly influences the performance of the mathematical models in the simulations is the choice of threshold level. It must be chosen high enough to not trigger from random noise, but at the same time low enough trigger when a weakened signal is present. Currently the thresholds have been chosen based on some trial and error to determine roughly what level gives the highest noise immunity. Once set, the same threshold is used for all of the correlator designs. As the voted designs have a different output resolution, the thresholds were set to the same relative level as the full resolution designs no matter the sampling rate. This ensured by having the thresholds determined by a formula which is scaled by the sampling rate. In other words, if all designs had their output normalized, the thresholds would be the same. Some work on determining the optimal threshold levels should be done.

7.4 Power Results

The designs presented here have not been designed specifically for power reduction, although this may seem to be an important factor for a lightweight radio communication link. The reason for this is the possible power savings for the various correlators is low, when compared to the other units required for the radio receiver. The major source of power consumption will be the analog to digital conversion section, and it was decided in cooperation with the external supervisor that expending extra effort on reducing the power consumption of the correlators was not of interest, given the low gains possible for the overall receiver unit.

With that said, a common source of power consumption in every presented correlator design is the ADC shift register, which causes a great deal of switching activity for every new sample. Proposed designs[5] have demonstrated reduced power consumption by shifting the PN code instead for each new sample, and utilizing a special ring buffer type of ADC register. All of this requires a significant increase in area however, which directly contrasts the desired goal of minimal area.

From the power consumption analysis performed on the correlator designs, some interesting observations can be made none the less. The leakage power is proportional to the area required by each design, as each gate will leak a set amount of current, and the area is proportional to the number of gates in a design. See tables 6.5 on page 61 and 6.2 on page 59. The 1-bit dual correlator designs both consume less dynamic power than the others, and only dissipate as much dynamic power as the sub correlator they are based on. This is reasonable, as the designs use two sub correlators, each of which are only active half of the time, making the dynamic power equal to that of a single sub correlator. The leakage/static power consumption of these designs is twice that of the sub correlators, as would be expected. Prior to running the power analysis, it was anticipated that the voted designs would be power efficient due to reduced output resolution. The idea being that less resolution would requires less switching activity, which the dynamic power is proportional to according to formula 6.1 on page 61. However the results show that the voted designs are not any more efficient than those with full output resolution.

A rather surprising result from the 2-bit correlators (table 6.11 on page 65) is the low dynamic power consumption of the grouping designs. The average for all of the other 2-bit designs is 75μW, whereas the multiplexed grouping correlators only require 13μW. Design 10 only dissipates 2.6μW, which is significantly less than the others. Why this is the case is unknown. The number of registers in the grouping design is the same as the others (see the percentages in table 6.8 on page 63) , and like the other designs they will all switch state when a new ADC sample is received. In addition, the large amount of combinational logic will surely dissipate some power with each ADC sample.

As the synthesis results do not include routing, the power lost in the required interconnects has not been factored in. For some designs, such as the grouping correlators, it is anticipated that significant routing is required, which may increase the dynamic power consumption greatly.

7.5 Choice of PN codes

The reasoning behind the decision to use the exact PN codes used here was considered outside the scope of this thesis, and they were therefore provided by the external supervisor. The PN codes used by the correlators is of little consequence when comparing area, power or even noise immunity in relative terms, because the same code is used by all of the 1-bit or 2-bit designs. When comparing the 1-bit to the 2-bit designs however, the PN code is of great importance and must be the same, which was not the case in this thesis. For this reason the results from the 1-bit and 2-bit correlators cannot be compared, if for example only the effects of higher ADC resolution want to be studied.

Chapter 8

Conclusion

Several correlator designs have been presented, for both 1 and 2-bits of ADC resolution, and using differing methods for both storing ADC samples and counting the correlation estimate. All of the designs have been proven to function, using mathematical models to verify correct functionality. A novel noise immunity testbench was developed to subject the correlator models to varying levels of noise, in order to determine when a data transmission would begin to show a BER of 1 in 1000. A correlator design based on majority voting was presented, but ultimately it was not favorable over the full resolution designs in any of the properties area, noise immunity, power dissipation or speed.

Surprisingly low power consumptions were reported for the grouping correlator designs. The reason behind these results is not known however, and further research is required to draw a conclusion on the power consumption.

The overall most area efficient designs for both ADC resolutions were based on multiplexing a parallel counting unit to sections of the ADC register. Of the multiplexed designs which were the most area efficient, it was not the design using the smallest 4-bit counter, but the 8-bit unit which used the least area. The power analysis revealed that the dual correlator designs are very power efficient, requiring half the dynamic power of the other designs, while being only marginally larger in terms of area and static power dissipation.

8.1 Future Work

The grouping correlator designs should require less area than the others, if some clever means of exploiting this can be found. The time delayed grouping correlator presented in chapter 4.2.4 on page 38 was never implemented, but combines the register savings of the grouping designs with the shift-register simplicity of the other designs. The only hurdle to implementing this design would be determining a clever manner to decode the encoded samples, given that they are spread across two registers, giving a 10-bit input which needs to be decoded efficiently.

The strange power dissipation results for the grouping correlators, and especially 2-bit design #10 need to be investigated further. In the event they are correct, they represent significant power savings compared to the other designs presented here.

Further work could be done to investigate how power efficient the dual correlator designs are compared to other low power correlators. As the main design focus in this thesis was on reducing area consumption and not power dissipation, no designs were made purposely with the intent of low power. Several low power designs have been presented in the literature, so finding such correlator designs to test against should not be difficult. The dual correlator designs can be altered to use any number of sub-correlators desirable, as shown in chapter 4.1.1 on page 20. Using numerous sub-correlators should potentially allow for very low power dissipation.

Chapter 9

Bibliography

[1] Tutorial 1890: An introduction to spread-spectrum communications, (February 18, 2003).

[2] RF testing of WLAN products, application note 1380-1, (January 10, 2007).

[3] F.K. Bowers and R.J. Klinger. Quantization noise of correlation spectrometers. 1974.

[4] Ching-Hung Chiou et al. A programmable pipelined digital differential matched filter for dsss receiver, (November, 2001).

[5] S. et al. Goto. A low-power digital matched filter for spread-spectrum systems, 2002.

[6] J.R. Jordan. Correlation algorithms, circuits and measurement applications, (February, 1986).

[7] Sung-Won Lee and In-Cheol Park. Low-power hybrid structure of digital matched filters for direct sequence spread spectrum systems, (July, 2003).

[8] Jan De Nayerlaan. Spread spectrum introduction, (October, 1999).

[9] Behrooz Parhami. Design of m-out-of-n bit-voters, 1995.

[10] Laurence B. Milstein Raymond L. Pickholtz, Donald L. Schilling. Theory of spread-spectrum communications - a tutorial, (May, 1982).

[11] Earl E. Swartzlander Robert F. Jones. Parallel counter implementation, (June, 1993).

[12] I. et al. Saini. Design of a high speed and low power digital matched filter for cdma system, (December, 2007).

[13] S. et al. Sajic. Low-cost digital correlator for frequency hopping radio, (June, 2011).

[14] T. Sugawara and Y. Miyanaga. A design of parallel matched filter for path search, (October, 2004).

[15] Katherine Brown Sundararajan Sriram and Anand Dabak. Low-power correlator architectures for wideband cdma code acquisition, (October, 1999).

[16] Jian Song Wei Li, Kewu Peng. Low-complexity implementation of pn correlator for wireless transmission systems, (April, 2009).

[17] Tan F. Wong. Spread spectrum & cdma, 2002.