Digital Detection System
For RF-Identification

Alexander B.C. Tijhuis

Dissertation submitted for the degree Magister Scientiae in Electrical and Electronic Engineering at the North-West University

Supervisor: Prof. Willie C. Venter

2004

Potchefstroom

Preface

The development of digital systems, computers and processors is taking a huge run nowadays. Because of this extreme improvement, new technologies and ideas can be developed. Especially those ideas which were not possible a few years ago. This project is also one of those ideas. Now that the technology is far enough, we tried to develop a digital equivalent for an analogue device. This report is the final result of this investigation and development.

I would like to thank a few people in special for helping me during my project. Thanks to Professor Willie Venter as my study leader and for coping with my stubbornness. Thanks to Jana Maritz for her help auditing this report and being a great support during the complete time of the project. Furthermore, I would like to thank Jan Bollen without whom I would not have come to South Africa in the first place. At last, I would also like to thank Castilions, which sometimes gave me the possibility to find creative solutions to difficult problems.

Alexander Tijhuis
Potchefstroom, South Africa, 2004

Summary

A South African company, iPico, developed an analogue detection system for RF-Identification tags. This device is however not as robust as it should be. Therefore it was decided to attempt to design and build a digital equivalent of this system.

The development of this digital system was divided into two phases; firstly a simulation program was developed and tested. This was to determine if this digital system would be able to perform as good as, or even better than the analogue system. The final simulation software is built up out of numerous components, including time domain and frequency domain filtering and an intelligent detection scheme to filter out the data.

The second phase was the implementation of the algorithms into hardware. This hardware consists of a DSP processor and an AD converter from Texas Instruments. Using this hardware, it was possible to implement the algorithms.

The final results of a few thorough tests illustrated that the digital system would be able to beat the old analogue system, whereby the digital system can still be further optimized. There are, however, a few remarks about the execution time, hardware optimizations and the non-optimized algorithms for specific undocumented situations.

Die Suid Afrikaanse maatskappy, iPico, het 'n analoog deteksie stelsel ontwikkel vir die deteksie van RF-Identifikasie kaarte. Hierdie stelsel is egter nie so robuust soos aanvanklik beplan is nie. Daarom is besluit om 'n poging aan te wend om 'n digitale weergawe van die analoog stelsel te ontwerp.

Die ontwikkeling van hierdie digitale stelsel is in twee fases verdeel. Eerstens is 'n simulasie van die digitale stelsel ontwikkel en getoets om te sien of die digitale stelsel sou kon werk, of selfs beter resultate sou lewer. Die finale simulasie sagteware is opgebou uit 'n groot aantal komponente, inklusief tyd gebied, frekwensie gebied filters en 'n intelligente deteksie skema om die data uit te filter.

Die tweede fase was die implementering van die ontwikkelde algoritmes in hardeware. Hierdie hardeware sluit 'n DSP verwerkings eenheid en 'n AD omsetter van Texas Instruments in. Deur gebruik te maak van hierdie hardeware, was dit moontlik om die algoritmes te implementeer.

Die uiteindelike resultaat van 'n paar deeglike toetse het aangetoon dat die digitale stelsel daartoe in staat is om die analoog stelsel te verslaan, waarby die digitale stelsel nog verder op verbeter kan word. Daar is wel enkele opmerkings oor die uitvoeringstyd, hardeware optimalisering en die nie geoptimiseerde algoritmes vir spesifieke ongedokumenteerde situasies.

Table of Contents

Illustrations

All figures, tables and formulas that are used in this report are noted here with their author. For more information about what specific book or report items are from, see chapter 7, References. All items that do not have this information are made by the author of this report.

Through the report, there are several figures without legend. These figures are a presentation of a specific signal strength against a certain time span. Because of its non-fixed character, both in time and strength, there is chosen not to include numbering in these figures.

List of Figures

List of Tables

List of Formulas

1. Introduction

An analogue reader system for the detection of wireless identification (RF-ID) tags was developed by the company iPico (http://www.ipico.co.za). This system should detect multiple ID tags up to a range of four meters. Unfortunately it is not as robust as expected, and very sensitive for disturbance from the surroundings.

1.1. Research aims and objectives

Using digital techniques and DSP processors, an equivalent to the analogue reader system has to be developed. First of all, a simulation must be performed to prove that the techniques that are used will have a big chance to perform the task just as well as, or better, than the original analogue reader.

When the techniques are proven, the algorithms will have to be implemented in hardware, making a prototype for the digital reader. This prototype will need to perform all its tasks by itself, except for some debug and control functions that are handled by a host computer.

The final goal is to have a working prototype that can prove that the digital RF-ID reader is able to detect the ID-tags in the same range of strength and quality as its analogue equivalent.

1.2. Method of Investigation and Construction

Using a combination of old and new techniques, it will be tried to achieve a maximum quality in filtering and detection. Filters that are widely used in analogue techniques can be converted to digital equivalents and be implemented in the algorithms, after being optimized for the specific application. As it already became clear in the problem statement, using only the techniques based on analogue systems does not fulfil the job completely. Therefore, additional digital techniques are also used. These techniques involve time-division algorithms, which can only be implemented advantageously in digital systems, and not in analogue.

It will be these algorithms that will form the base of the solution to achieve better performance and robustness in the digital system.

2. Literature survey

In this chapter, insight will be given into what knowledge and technologies are available about the analogue reader system. This analogue reader system formed the basis for the working of the simulation program.

Thereafter, some information will be given about typical noise problems that can be expected during this project. And finally, some theories will be explained that form the basis for the filtering in the detection program.

2.1 Existing Analogue Components

2.1.1. EM 4322/4022 Chips

These are the RF-ID chips on which the actual program is based. Although some differences between the 4322 and 4022 chips exist, they are very small. In this part, the basics of the EM 4022 chip will be discussed, which are the same for most of the dual frequency tags from this range. When there are differences between the tags, these differences will be addressed at the proper time.

Fig 2.1.1: ID-tags

Read-only Dual frequency Identification Device

This is the complete name for the range of devices that is formed from the chips. An example can be seen in figure 2.1.1. The reason why they are read-only is because the chips are programmed during the manufacturing. After this, the configuration and the ID code cannot be changed anymore.

The term "Dual frequency" comes from the system that is used on the tag. As these are passive devices, they need to receive energy before they can transmit their ID. This information is sent to the device on one frequency, while the data is transmitted on another frequency, thus making it dual frequency. The frequencies that are used on these tags are 125 kHz for the energy transport and 7 MHz for the data transport.

2.1.1.1. Architecture of the Chip

This chip is packed in a housing that contains a receiver coil, a transmitter coil, an energy capacitor and the chip itself of course. This complete package is also known as the "ID-tag". This name will also be used later on in the dissertation. The receiver coil is used to receive power from the reader system that will be temporarily stored in the capacitor until the chip boots up. The transmitter coil is used to transmit the data signals. These functions will be explained more thoroughly later on.

On the chip itself, there is place for an energy regulator, an oscillator and the ROM-logic. The energy regulator is built to manage the power that is received. Its main functions include ensuring that the capacitor doesn't get overloaded and damage itself or the chip in the process, and to ensure that the ROM-logic doesn't boot up until there is enough power in the capacitor to be able to broadcast an entire ID. The oscillator is used to give the exact frequency that is needed for the data signals to the chip. It has to be noted that this is not the frequency on which the data is broadcasted, that frequency is determined by the coil. It only determines the bit rate of the data signal. The ROM-logic contains the logic to be able to send out the ID information, and the register that contains the ID code.

Data and speed
When there is enough energy to broadcast the entire ID, the main chip will power up and it will start combining all the data for a complete ID broadcast. This complete broadcast of an ID will contain the following bits;

- 11 Start bits
- 3 Synchronization bits
- 64 data bits, containing;
    - 2 bits reserved for future usage
    - 8 bits manufacturers' code
    - 38 bits unique ID
    - 16 bits CRC code

The broadcasting of the signals can be done on a few different speeds;

- 32kbit/sec
- 64kbit/sec
- 128kbit/sec
- 256kbit/sec

In practice, the 64kbit/sec version can already be used with an analogue reader. The version of the reader that supports up to 256kbit/sec is at present time not available for commercial use.

2.1.1.2. Encoding of bits

The 64 bits that come from the chip are encoded before they are broadcasted. This is done to make it easier to transmit the signals and to make the broadcast more reliable. There are two techniques that are used for this encoding. The EM4322 ID-tags use the Glitch data encoding, while there is a possibility to choose between the Glitch and the Manchester encoding on the EM4022 chips. This configuration will be programmed on the chip during manufacturing. Both data encoding techniques are summarily explained here.

Manchester Data Encoding
Manchester encoding is based on the difference between the first and the second half of a bit period. When the first bit period has a high value, it means that the bit is a "1", when the second half of the bit period is high; the bit is a "0". This system is often also known as a "0" on a rising edge and a "1" on a falling edge.
Although this encoding type is quite unprofitable in its use of bandwidth, it is very robust; more than one level of a bit period has to be disturbed before the bit is received wrong. An extra advantage of this scheme is that it is possible to extract the clock rate out of it. Keeping this in mind, it is not necessary to have an extremely fixed and synchronized clock rate on the devices or to send the clock rate with the ID broadcast.

This is of course better and easier for the design of the chips, as well as for the maximum velocity the ID-tags can have before running into problems with the Doppler Effect.
To illustrate that this encoding scheme is very well designed; Manchester encoding is also used on Ethernet networks, the world's most used computer network type.

Glitch Data Encoding
There is not a big difference between the Manchester encoding and the Glitch encoding. Instead of dividing the bit period in two parts, as with Manchester encoding, the Glitch bit period is divided into four parts, or quarters. When the bit value is a "1", it can be seen as a high output value on the first quarter of the bit period. When there is a "0" bit value, it will show as a high output on the third quarter of the bit period.

To make it a little bit easier to understand, below in figure 2.1.2 is a small scheme that gives the values of the Clock, Data, Manchester encoding and the Glitch encoding together.

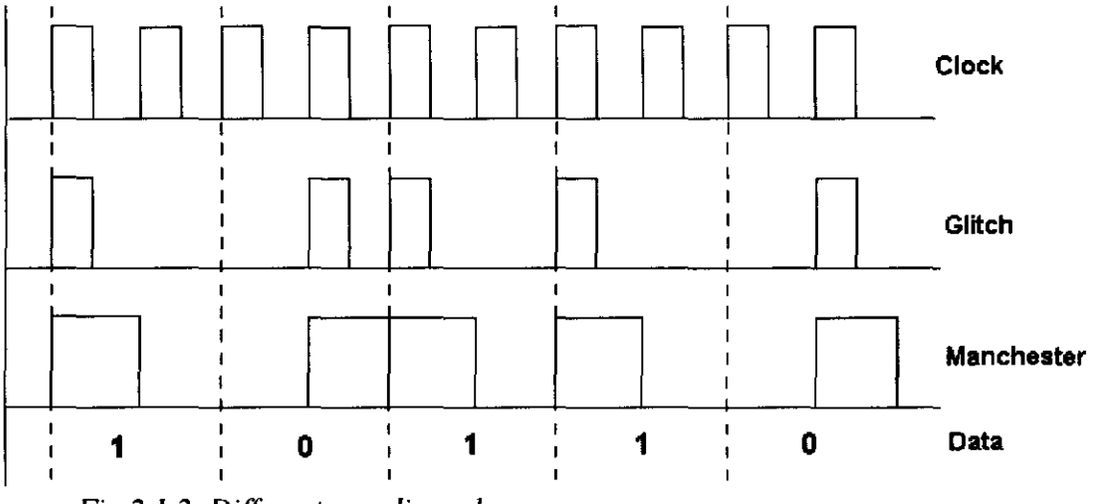

Fig 2.1.2: Different encoding schemes

The reason why Glitch encoding was chosen on most of the tags, is because less power is needed to broadcast the signals when using this encoding scheme. In the Manchester encoding-scheme, there has to be output half of the time during an ID-broadcast, with the Glitch encoding-scheme, this is only a quarter of the time.

2.1.1.3. Output of an ID-tag

This stream of encoded bits is sent to a modulation transistor that triggers the transmitter coil. This part of logic puts the data in specific pulses that can be modulated. These output pulses are then broadcasted using the second coil-capacitor-system. The capacitor and the coil together determine the exact frequency on which the signals will be broadcasted. A way to calculate this frequency is to use the following formula:

$$F_0 = 1 / (2*PI* \sqrt{(L*C)})$$

F_0: Frequency on which the coil/capacitor system will resonate
L: Value of the coil
C: Value of the capacitor

Formula 2.1.1: Resonation frequency of coil/capacitor system

Because the coil is generally far from ideal, it has a Q-factor that specifies the quality of the coil, which is a value for the time in which the signal will die out. This Q-factor is used on the chip to make sure that the signal mutes quickly enough before the next bit has to be transmitted. In some cases, it is even strengthened with an additional resistor, because the Q-factor of the coil itself it too small. In other words, the coil is too ideal to function properly, how ironic.

The following graph (figure 2.1.3) is a rough representation of the generated signals. Please note that this is only generated using a simulation program, with the real components there can be differences due to changes in the Q factor and the accuracy of the components.

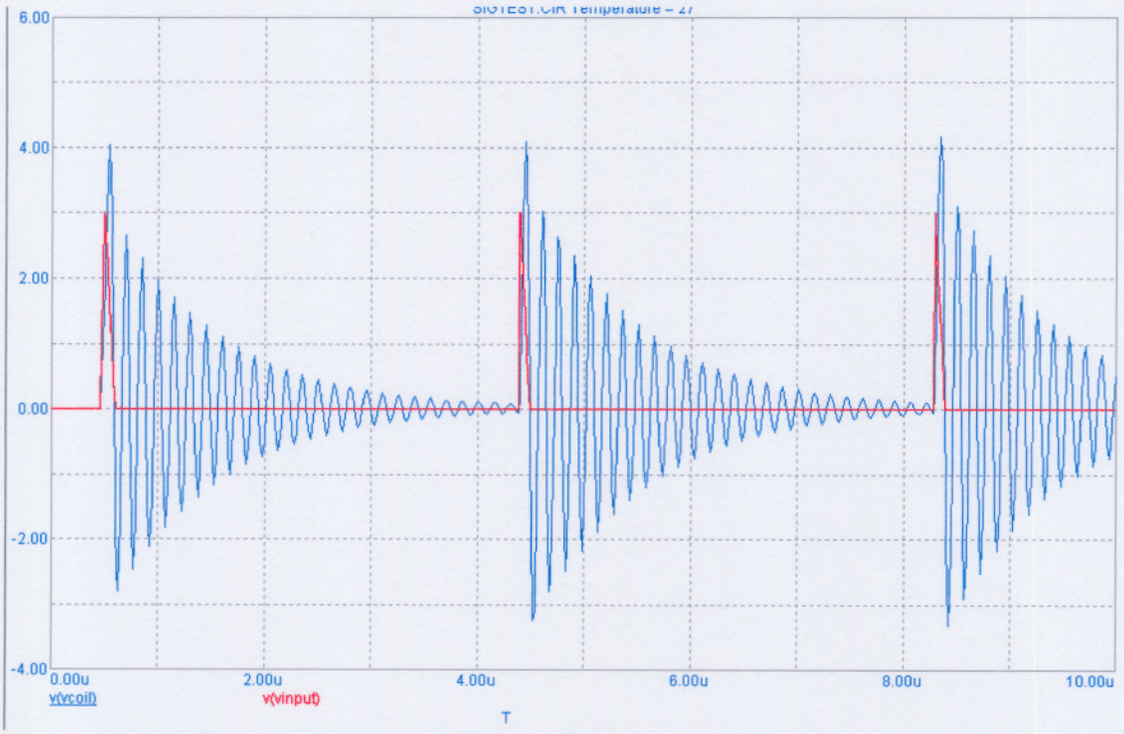

Fig 2.1.3: Input Spikes and Output Waves

These 7 MHz signals should then be received by a reader.

2.1.2. IP3020 Reader

This is a Dual Frequency single channel analogue reader that can be used for the detection of the tags that were discussed earlier (figure 2.1.4). This is also the reader that formed the problems which led to this project.

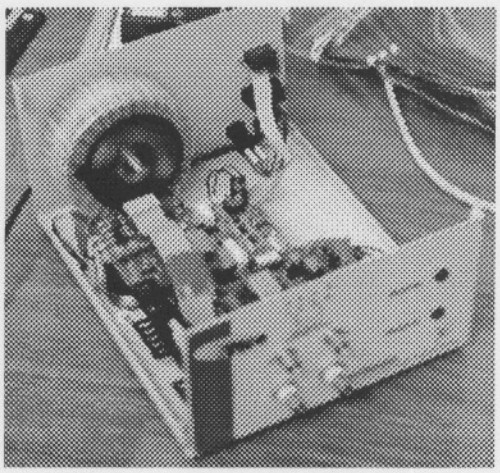

Fig 2.1.4: IP3020 Reader

The analogue reader consists roughly of the following parts;

- Receiving and broadcasting antenna
- 125 kHz Generator unit
- Filter between 7 MHz incoming wave and 125 kHz outgoing signal
- Filtering and amplification of the incoming signal
- Demodulation
- Peak detection
- Bit detection and CRC encoding

The RF-ID reader constantly broadcasts a 125 kHz sine wave to the environment in the vicinity of the antenna. This signal is strong enough to make it possible for the ID-tags to receive the amount of energy that is needed to broadcast their ID information.

When the reader receives one of these ID codes, as discussed in the last paragraph, the reader will go through the following steps;

2.1.2.1. Antenna

The antenna that is used on the reader consists of two loops of coaxial cables (figure 2.1.5). When a tag gets in the neighbourhood of a reader, it will receive its power through the antenna that is tuned to the specific frequency so that the power is absorbed as efficient as possible. When it has received enough power to be able to transmit a complete ID-signal, it will start sending its data in the form of AM-modulated pulses back to the readers' antenna, as discussed before.

The same antenna system that is used to broadcast the 125 kHz energy wave will then pick up the 7 MHz data signal. Directly after the signal enters the device, the energy wave and data wave are separated from each other.

Fig 2.1.5: Antenna of reader

2.1.2.2. Envelope detection

The data signal will then be fed into an envelope detection, that 'demodulates' the AM modulated incoming signal, whereby the 7MHz waves are converted back into the rough block signals of a certain bit speed. This rough signal is then converted to a TTL signal (0 and 5 V)

2.1.2.3. Decoding

The TTL signal is transported to a decoding chip, which handles the decoding of the Glitch/Manchester system and the CRC-check. When this is all done, the final ID code can be transferred to a PC, using a RS232 connection.

2.2. Typical Noise Problems

When developing the system, it has to be considered that there are different kinds of noise that can occur in practice. It is important to know and to categorize these different types of noise. This is to make a good choice in digital filters possible. Roughly, these disturbing signals can be divided into three categories:

- White Gaussian noise
- Sine waves
- Spikes

2.2.1. White Gaussian noise

This noise (figure 2.2.1) contains all possible frequencies in the same amount. Although this kind of noise is practically impossible, it gives a good representation of the noise that can normally be found in the air and in most analogue components. Because all frequencies are present in this type, it will also be the most difficult to filter out.

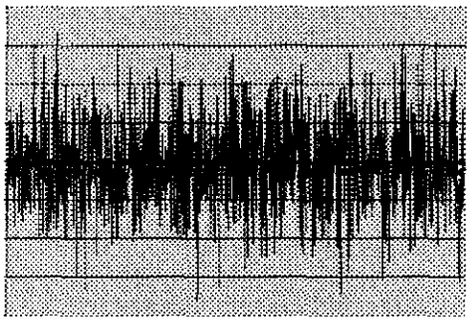

Fig 2.2.1: White Gaussian noise

2.2.2. Sine waves

Most of the cables that transport power are not shielded well enough to prevent radiation. These cables radiate mostly sine waves. Because the power that run through these cables are quite strong compared to data signals, the radiation can get relatively big.

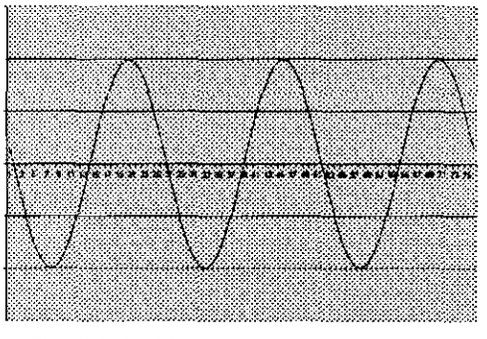

Fig 2.2.2: Sine wave noise

A second source of sine waves can be the reader itself. The power that is sent to an ID-tag to power it up has a frequency of 125 kHz. This frequency runs quite close, or even partially on the same connections, as the incoming signal. These power signals can be much stronger than the ID-tag signal. To illustrate how strong these signals can be; one of the known practical situations of a reader system involved an ID-signal strength of 50 micro volts. Added to this signal, there was a 0.7 volt sine wave (figure 2.2.2); a strength difference of almost 83dB.

2.2.3. Spikes

When there are other devices in the neighbourhood of the antenna, it is possible that the antenna picks up signals that are generated by these devices. This can come from switching power supplies, signals from cables, computers and even from electric drills. The pattern these devices mostly produce often look like spikes (figure 2.2.3).

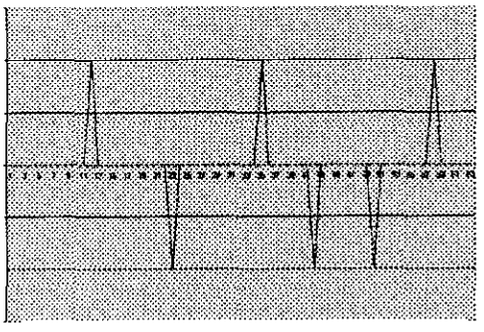

Fig 2.2.3: Spike noise

During an earlier test from iPico, the biggest disturbance in the environment was generated by a laptop that was connected to a device using a serial connection. This disturbance was big enough to bring the quality of the reader back at least a factor ten.

Seen on the fact that this type of noise is likely to become one of the main problems, a little bit more insight will be given. In figure 2.2.4 an example is given of an incoming signal that is disrupted with this type of noise. In the first quarter of the graph, all the signals are only noise, there is no data stored that is interesting. It can be seen that this noise is a few times stronger than the actual data signal. Even although this noise does not occur during the transmission of the ID, this type of noise is still very likely to interfere with the detection scheme by either overpowering the analogue components or by disturbing the synchronisation scheme.

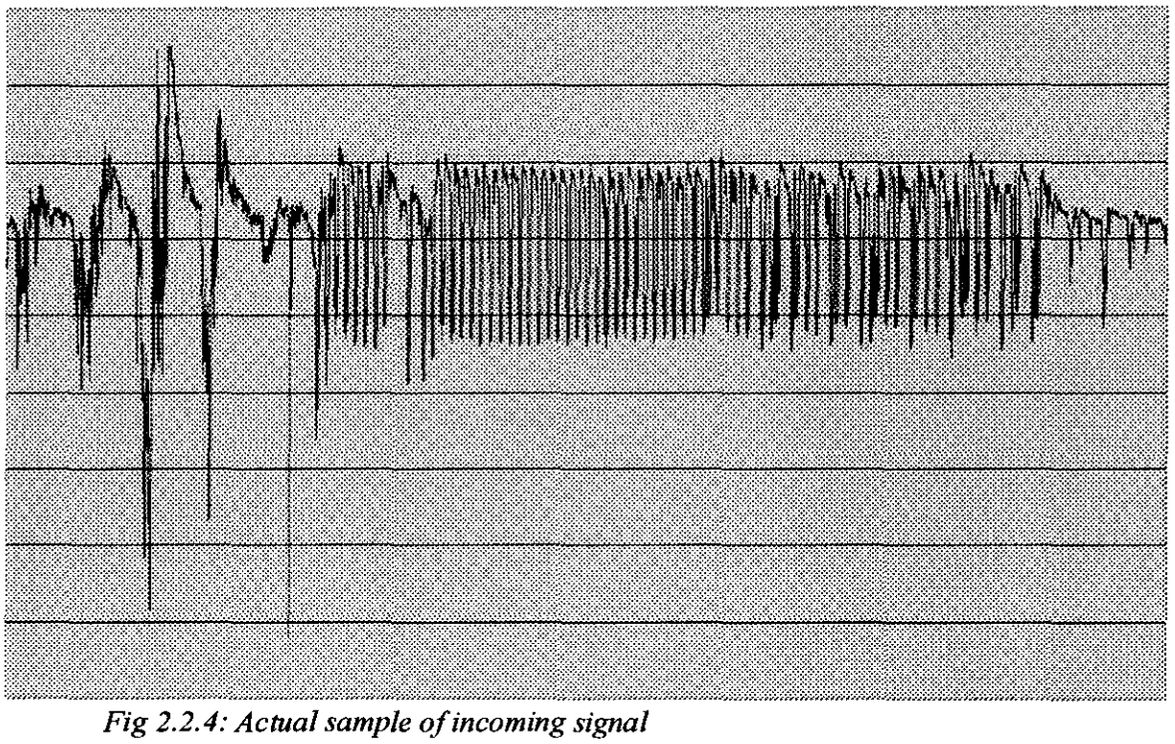

Fig 2.2.4: Actual sample of incoming signal

2.3. Theories Used

Before certain theoretical issues are explained, one thing must be noted first; the information given in this section is only a short summary. It also does not include all information of the side effects that one has to be aware of, before you can use and/or implement these filters. The intention of the explanations given is only to give a small background on the used techniques. If more information is needed on specific subjects, please contact the author (alexander@abct.net) of this document or visit the website http://www.dspguide.com where a detailed explanation of Digital Signal Processing and DSP filtering can be found.

2.3.1. Convolution Theory

"Convolution is a mathematical way of combining two signals to form a third signal. It is the single most important technique in Digital Signal Processing. Using the strategy of impulse decomposition, systems are described by a signal called the impulse response. Convolution is important because it relates the three signals of interest; the input signal, the output signal and the impulse response. "

> *Taken from the book "The Scientist and Engineer's guide to Digital Signal Processing", by Steven W. Smith.*

Although these few sentences explain most of what there is to tell about convolution it is not the best way to explain how this technique is used to solve the specific problem of the RF-ID reader.

2.3.1.1. Filter Kernel

The filters that are constructed using the convolution theory consist of an array of values that is called a "kernel". This kernel contains a representation of the filter that has to be implemented. Figure 2.3.1 is a possible implementation of an 8-size filter kernel.
Using this kernel, filters can be designed for a high or low pass system, time domain filters and matching filters. In some cases these filters are calculated directly from actual data, while sometimes they are designed in the frequency spectrum before being ported to the time domain. Using these different schemes of calculation, complex filters with extraordinary characteristics can be designed.

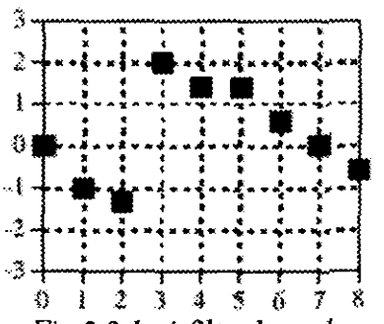

Fig 2.3.1: A filter kernel

When this kernel is used for the actual filtering, the following formula is executed:

$$Y = X[] * H[]$$

Y: Output value
X[]: Array with input values
H[]: Array with kernel values
Formula 2.3.1: Convolution function

Every value that is in kernel array will be multiplied by one sample from the input values and added to each other, thereby forming one output value that consists of the certain amount of multiplications.

In the next two paragraphs there will be a more thorough explanation of two different types of kernels that form two different approaches of a digital filter. There will also be an example of the complete process of calculation to make everything a bit more clear.

2.3.1.2. Quality vs. Computation

It has to be noted that the quality of the filter is proportional to the size of the filter kernel, especially with the previously mentioned type of filters. The bigger the filter kernel, the stronger filtering is possible. As usual, every good side has a bad side; when the size of the filter kernel is increased, more memory is needed and the amount of calculation power necessary will also increment (figure 2.3.2).

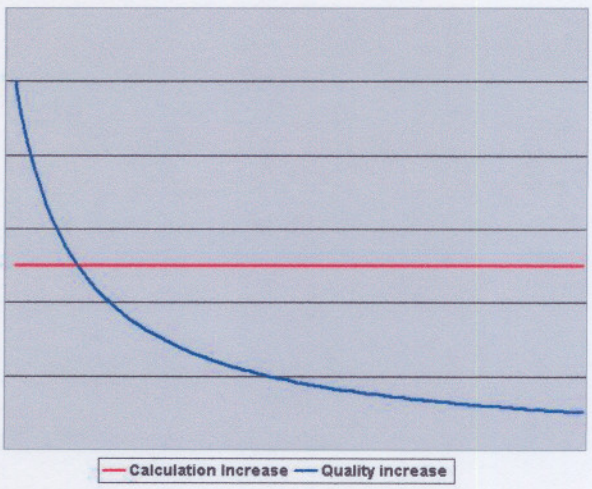

Fig 2.3.2: Calculations vs. Quality

For example; every additional sample gives a fixed amount of extra calculations that have to be done, but the increase in quality is not a fixed percentage. When increasing the kernel with one value, the quality increase will be two percent on a 50 point kernel, but only one percent on a 100 point kernel.
Therefore, it becomes an important issue to find an optimum between the two variables; the calculation time available and the size of the kernel.

2.3.2. Matched Filter

The Matched filter technique is based on the convolution theory, as well as the Windowed Sync filter, which will be discussed in the following paragraph. The biggest difference is that the Matched filter does not use the frequency domain and the impulse response to calculate the filter kernel. Instead, the kernel is based on a copy of the original signal. Actually quite simple, no direct mathematics is needed, but for some applications it is the best filter.

To explain how the Matched Filter works, an example is given that is based on a system that has to detect block pulses. The block pulse as shown in figure 2.3.3 is chosen because this is the most simple and easy explainable waveform when using a matched filter.

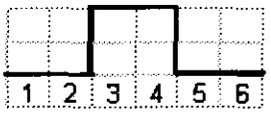

Fig 2.3.3: Input signal

When the outline of an input signal is known, it is possible to make the size of the filter kernel exactly the same size, or as close as possible to the input signal. In this case, the period of the input signal is six samples long; therefore the filter kernel can also be made six values (figure 2.3.4). This also expects that the block pulses, that have to be detected, are 2 samples long, to make sure that the filter is "Matched".
If the kernel does not contain the exact same size as the incoming signal, the filter will still work, but the signal to noise ratio of the output will always be weaker than when the kernel is "Matched".

Again, it has to be mentioned that the bigger the filter kernel, the stronger the filtering. Although the amount of samples is fixed because the length of the signal has to be matched, the filter kernel can still be increased in size. To do so, the amount of incoming samples has to be increased by using a higher sample rate. If the sample rate is higher, the incoming signal contains more samples per period, and the kernel size also can be increased. This is, of course, only possible when the sample rate *can* be increased.

In the example of the block pulses, the kernel will look like the figure at the right. Indeed exactly the same as the input, assuming that this is an ideal situation and there is no noise on the incoming signal.

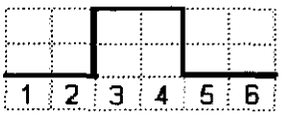

Fig 2.3.4: Filter Kernel

Of course in reality, these graphs are shown as numbers instead of figures, on which the calculations are performed. To get a better idea of the filtering process, this will now be demonstrated.

2.3.2.1. Theoretical Example Matched filtering

Below are three arrays; X[], H[] and Y[]. X[] contains the incoming samples, H[] contains the filter kernel that has been made and Y[] will contain at the end the calculated output. At the moment the first sample is sent in, the Y[] array is still completely empty.

The "-" means that this sample is undefined. Because this is not a desired situation, these samples are normally calculated as 0; as in this case. These undefined samples only occur when the system is started or ended, so the amount of undefined samples is always a fixed value that depends on the size of the filter kernel.

During one calculation, the signals of X[] and H[] are multiplied where after the results of the multiplication are added to each other, which generates one value of the Y[] array.

```
X[]  0  0  1  1  0  0  -  -  -  -  -
H[]                    0  0  1  1  0  0   *
                       0  0  0  0  0  0   +
Y[]  0

X[]     0  0  1  1  0  0  -  -  -  -
H[]                    0  0  1  1  0  0   *
                       0  0  0  0  0  0   +
Y[]  0  0

X[]        0  0  1  1  0  0  -  -  -
H[]                    0  0  1  1  0  0   *
                       0  0  0  0  0  0   +
Y[]  0  0  0

X[]           0  0  1  1  0  0  -  -
H[]                    0  0  1  1  0  0   *
                       0  0  0  0  0  0   +
Y[]  0  0  0  0

X[]              0  0  1  1  0  0  -
H[]                    0  0  1  1  0  0   *
                       0  0  1  0  0  0   +
Y[]  0  0  0  0  1

X[]                    0  0  1  1  0  0
H[]                    0  0  1  1  0  0   *
                       0  0  1  1  0  0   +
Y[]  0  0  0  0  1  2

X[]                 -  0  0  1  1  0  0
H[]                    0  0  1  1  0  0   *
                       0  0  0  1  0  0   +
Y[]  0  0  0  0  1  2  1

X[]                 -  -  0  0  1  1  0  0
H[]                    0  0  1  1  0  0   *
                       0  0  0  0  0  0   +
Y[]  0  0  0  0  1  2  1  0
```

X[]						-	-	-	0	0	1	1	0	0			
H[]						0	0	1	1	0	0						*
						0	0	0	0	0	0						+
Y[]	0	0	0	0	1	2	1	0	0								
X[]						-	-	-	-	0	0	1	1	0	0		
H[]						0	0	1	1	0	0						*
						0	0	0	0	0	0						+
Y[]	0	0	0	0	1	2	1	0	0	0							
X[]						-	-	-	-	-	0	0	1	1	0	0	
H[]						0	0	1	1	0	0						*
						0	0	0	0	0	0						+
Y[]	0	0	0	0	1	2	1	0	0	0	0						

Table 2.3.1: Calculation of Matched Filter Output

Please note that on the sixth calculation, the filter is completely "Matched" and the calculation generates the highest output value. The output signal of this calculation now looks like figure 2.3.5.

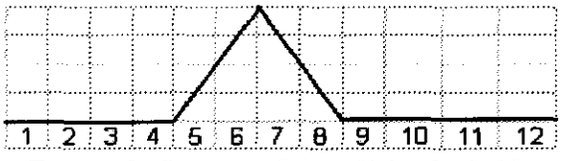

Fig 2.3.5: Output values of Matched Filter

Although the shape of the output signal is different than the input signal, the strength of the signal has doubled in this occasion. Keeping this in mind, it can be expected that the bigger the filter kernel, the stronger the output will be compared to the noise.

2.3.2.2. Practical example Matched filtering

Until now, the filtering has only been done with an optimal situation. In practice, a filter only needs to be implemented when the situation is not optimal, when there is noise. To get an idea of how this filter will perform with noise, below in figures 2.3.6 and 2.3.7 are two graphs of an input and an output signal of a Matched Filter where white noise and other disturbances are present in the input signal.

In this case, the Matched filter's kernel has a length of fifteen values.

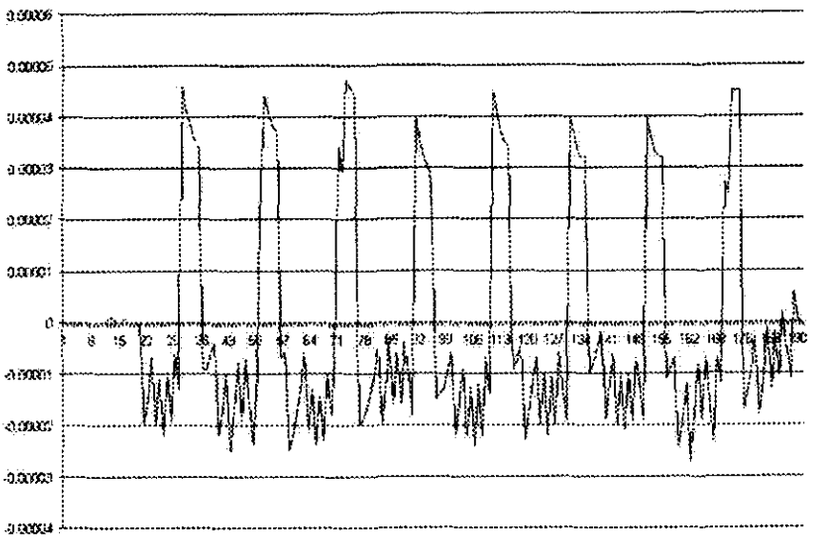

Fig 2.3.6: Input Signal of the Matched Filter

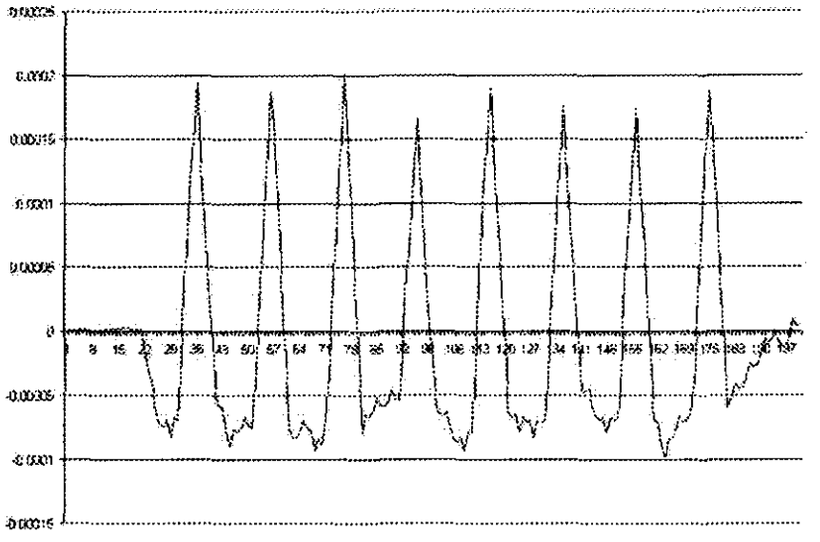

Fig 2.3.7: Output Signal of the Matched Filter

It is clear that the signal to noise ratio has improved. In the incoming signal, there is noise strength of about 0.015mV against signal strength of 0.06mV, thus giving a signal to noise ratio of about 6 dB. In the output, the noise has been reduced to 0.002mV against signal strength of 0.025mV. This means a signal to noise ratio of about 11 dB. This may not seem that impressive, but keep in mind that this is filtering out white noise!

2.3.2.3. When a Matched Filter?

A Matched Filter is not in all cases the best solution. The following advantages and disadvantages have to be kept in mind.

Advantages of the Matched Filter

- Strong filtering of white Gaussian noise.
- Specific signals are detected stronger, irrespective of the frequency.
- Strong filtering is possible when the signal and the noise frequencies are close to each other.

Disadvantages of the Matched Filter

- The output of the filter can have a different outlook than the input signal.
- Bad filtering when the frequencies are far from each other.
- The signals that have to be detected must be known in order to generate the filter.

2.3.3. Windowed sinc filter

"Windowed-sinc filters are used to separate one band of frequencies from another. They are very stable, produce few surprises, and can be pushed to incredible performance levels. These exceptional frequency domain characteristics are obtained at the expense of poor performance in the time domain, including excessive ripple and overshoot in the step response. When carried out by standard convolution, windowed-sinc filters are easy to program, but slow to execute."

> *Taken from the book "The Scientist and Engineer's guide to Digital Signal Processing", by Steven W. Smith.*

2.3.3.1. Sinc function

The windowed-sinc function is used as a basic for the development of an FIR (Finite Impulse Response) filter that has to filter most of the unwanted sine noise out of the incoming signal. As the name already implies, this type of filter is derived from a sinc function. A sinc function (figure 2.3.8) is the inverse Fourier transformation of an ideal step response, thereby making it the perfect base-line to form an extreme powerful filter in the frequency domain.

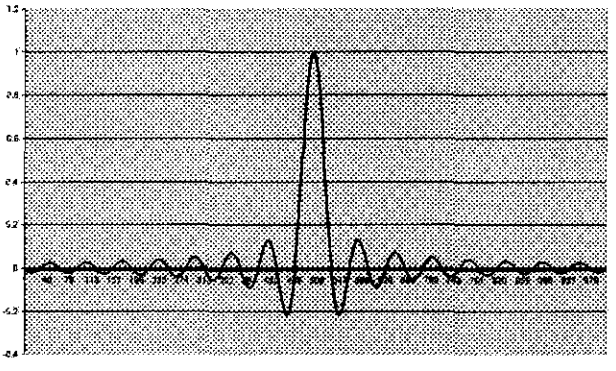

Fig 2.3.8: Sinc function

The standard formula for a sinc function looks like:

$$h(i) = \sin(2*PI*f_c*i) / i*PI$$

f_c: Cut-off frequency

Formula 2.3.2: Sinc function

2.3.3.2. Window function

Unfortunately, a typical sinc function continues in both positive and negative direction to infinity. When this is recalculated into a working filter kernel, it would form an extreme task to process for a digital system. When the sinc function is truncated to only take the most important values of the curve, it will directly influence the quality of the filter with a large ripple and a slower roll-off.

This problem can be handled by introducing an extra element in the design of the filter kernel; the window. The window function consists of a curve that starts at zero, rises to a certain limit and drops down again to zero. When this window is multiplied with the values of the sinc function, there will be no problem with the truncation anymore, because the values at the edge of the kernel are already very close to zero. There will, however, still be a loss in the roll-off and the stop-band attenuation, because the characteristics *did* change.

There are several algorithms developed that can be used as a window function. A few of these are;

- Hamming window
- Blackman window;
- Raised cosine window;
- Bartlett window

Every one of these windows have different characteristics that will influence the filter kernel in different ways; some will give a better roll-off, but weaker stop band attenuation, while other windows influence the filter just the other way around.
In the case of this project, an important parameter will be the stop band attenuation of a filter; therefore a window has to be chosen that will influence the quality of this parameter as little as possible. According to measurements, the Blackman window (figure 2.3.9) is able to achieve the strongest stop band attenuation; up to -74dB.

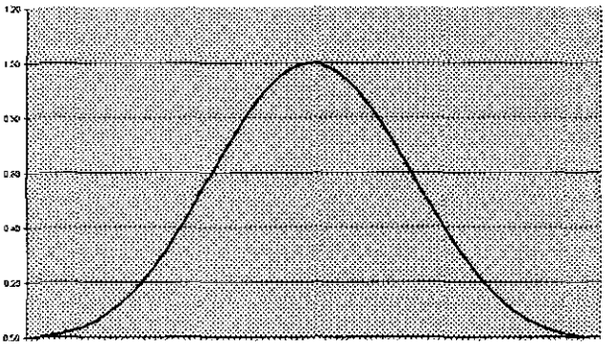

Fig 2.3.9: Blackman Window function

$$f(i) = 0.42 - 0.5 * \cos(2*PI*i/m) + 0.08 * \cos(4*PI*i/m)$$

m: Size of the window
Formula 2.3.3: Blackman Window function

2.3.3.3. Combining the techniques to a working filter

Now that the basics of the sinc and the window functions are known, it will be quite easy to calculate a working model of the function.
When the two parts are combined, the following formula is formed;

$$f(i) = \frac{\sin(2*PI*f_c*(i - m/2))}{i - m/2}\left(0.42 - 0.5*\cos(2*PI*i/m) + 0.08*\cos(2*PI*i/m)\right)$$

m: Length of the filter kernel
f_c: Cut-off frequency
Formula 2.3.4: The Windowed-Sinc function using a Blackman window.

High-Pass and Band-Pass filtering
Using the formula as discussed above, a low-pass filter is designed. This is not always wanted. In a lot of cases a High-Pass, a Band-Pass or even a Band filter is needed. Although this is the opposite of the Low-Pass filter that is explained above, this filter kernel can be used to form these different types of filters. Actually, because it is completely opposite, it becomes quite easy to perform this conversion.

Using "Spectral Inversion", a Low-pass filter can easily be transformed into a High-pass filter. All that has to be done is to inverse every sample that has been calculated using the formula above and add one to the sample in the middle of the kernel (this is one reason why a filter kernel should always be an uneven amount of values). In the time domain this converts the Low-pass to High-pass, while keeping the same cut-off frequency. To make a Band filter, one can simply make a high pass and a low pass filter and then add them together.
So at the moment a Low-pass filter is designed, the other possibilities are quite easy to produce.

2.3.3.4. When a Windowed-Sinc filter?

Also with this filter, some advantages and some disadvantages exist.

Advantages:
- Extreme strong filtering possible
- Different types of filters are possible (Low-Pass, High-Pass, etc.)
- Theoretically every cut-off frequency can be chosen, without changing the sample rate.

Disadvantages:
- Slow execution
- Most useful with frequencies further from each other
- Can use a lot of memory
- Difficult to use for filtering out white noise

Based on these advantages and disadvantages and also the theory explained above, the most effective uses that can be found for this type of filter are in the discrimination of a small data signal from a huge sine wave and the discrimination between two signals with frequencies close to each other.

3. Simulation

The first task during the development was to design and implement a simulation program, which should be able to perform the same functions as the "old" analogue system. This was necessary to determine if it would be possible to design a digital system capable of performing the same functions as the analogue system.

In this chapter the development of this simulation program will be discussed. To start with, all the components that form the generation of the incoming signal will be discussed. The simulation of the incoming signals was necessary, due to the fact that actual signals were not available at this stage. The major draw-back of this approach is that it is not possible to be sure if the signals that are generated are a close match to the real situation. On the other hand, the great advantage of this system is that it is possible to generate signals with specific characteristics which are very difficult to generate with the actual device, such as repeating the same signal and strength over and over again. This can be very helpful for testing specific parameters in the simulation.

After the generation software, the detection part will be discussed. A lot of components for this program can be copied from the generation software, only being executed in opposite order. Also the most important parts of the filtering and bit detection will be put in the spotlight, giving more information on the practical implementation of the theories discussed earlier.

The optimization of the simulation, possible future optimizations and an explanation of the final simulation software is also briefly discussed in this chapter.

3.1. Explanation of the Developed Systems

At the beginning of this project, the first aim was to develop a simulation that would be able to detect ID's broadcasted using the Dual Frequency system (see paragraph 2.1). This detection would receive its signal from a connection directly behind the antenna of the analogue device. This would mean that the signal received from the analogue device is a data signal, modulated with a 7 MHz sine wave. To make high-quality detection of this signal possible, a sample rate of approximately 25 MHz has to be used, keeping the theoretical minimum of two times the highest frequency in mind.

Parallel to the development of the simulation for the digital Dual Frequency detection, a simulation for a UHF(Ultra High Frequency) system was requested. The reason for this request was the good results of one of the first tests of the simulation of the Dual Frequency detection. Seen on the fact that the UHF system was basically the same as the DF(Dual Frequency) system, both simulations were implemented in one program, only with two important differences.

The first difference between the UHF and DF systems was the sample rate. While the DF system was working with the raw incoming signal, the UHF system would work with a signal after demodulation, forming block pulses instead of waves. In this case, the sample rate could be determined by taking the highest data frequency, instead of its modulated frequency. Therefore, the sample rate used on this system was 5 MHz.
The second difference is the way the filters are initialized. Using different frequencies also means different cut-off frequencies and different filter designs.

When the project evolved it became clear that a sample frequency of 25 MHz would not be feasible. A solution then came out of an unexpected corner; what if the incoming signal would also be tapped from the analogue scheme after the demodulation, as it was done on the UHF system. Realizing so, the sample frequency also came down to a limit based on the data frequencies. Seen on the fact that the bit speeds of the ID-tags for the UHF system and the DF system are basically the same, the same value could be used. Therefore, apart from a few practical differences between the systems, both the UHF and the DF system can use the same algorithms and sample rates.
This development was only implemented during the hardware implementation and not during the development of the simulation.

The software discussed in this chapter is based on the two systems mentioned above; the Dual Frequency system with a sample rate of 25 MHz and the Ultra High Frequency system with a sample rate of 5 MHz. After this chapter, only the Dual Frequency system will be exposed, the further development of the UHF system is redirected to another person, although strong connections remain between both projects.

3.2. Generation Software

Generation of the data signals and detection of the ID's was done with different programs. This is because of a few reasons;
The problem itself consists of the detection of the specific signals that are picked up out of the air, at the moment the analogue signal is sampled. After this point in time, no disturbing signals or noise can be added anymore. This means that at the moment the program starts, there is only one stream of samples with which the final output values have to be calculated. To make this very strict, it is easier to have a separate program for the generation, hereby ensuring that no signals can be added in between.

Another reason for choosing separate programs is to increase the reliability of the results. When one program is responsible for the generation of the signals as well as detection of the IDs, information that is generated, could still be in the memory when the detection takes place, and therefore influencing the results with a subsequent decrease in reliability. Thus again, using two different programs has more advantages and the results more trustworthy.

The last reason was to make it possible to change the incoming samples before starting the detection. For example, if it was necessary to give extra noise or specific waveforms in the incoming signal, it was possible to read the data file into for example Matlab, add the noise or other waveforms and give the output file back to the program. The detection program was not able to detect any difference between the original input file and the edited input file. This method of editing the signals was used a lot during the evaluation of specific noise problems.

3.2.1. Amount of Sample bits

Although the routine that uses this value (the amount of sample bits), is only executed just before the sample is written to the file, it is preferable to explain its significance at the beginning. The importance of this number is due to the impact it has on the roughness of the signal. When it is chosen too low, the detection software will not be able to detect the signal, and when it is chosen too high, the software will need a larger computation power to sample the signals as well as to filter out the signals. That is, if the analogue to digital converters are actually available with this higher number of sample bits.

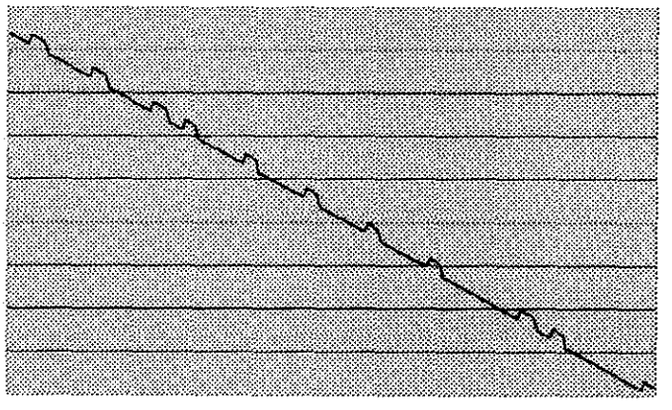

Fig 3.2.1: 20 bit Input signal

To demonstrate how important the value of the number of bits can be, three figures are included with different amounts of sample bits. All three figures are the same signal; block/data pulses on the falling edge of a large sine wave without white Gaussian noise. Figures 3.2.1, 3.2.2 and 3.2.3 are the signals sampled using respectively 20 bits, 12 bits and 8 bits. The input range over which the bits are divided is the same in all three examples. The only change due to the variation in the amount of sample bits, is the step-size per bit; the voltage step that one bit represents. The more sample bits, the smaller this step.

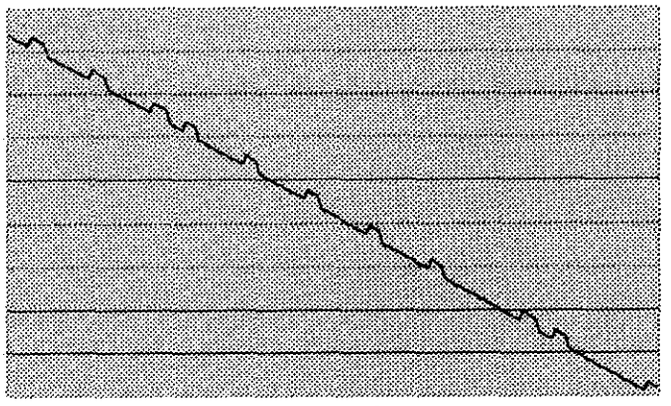

Fig 3.2.2: 12 bit Input signal

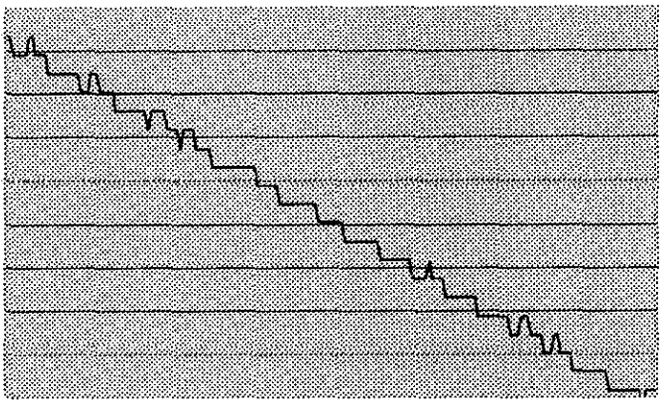

Fig 3.2.3: 8 bit Input signal

The difference between the 20 bit figure and the 12 bit figure is minimal. Only when looking very close at it, a small difference can be found between the figures. There is, however an extreme difference when looking at the same signal sampled at 8 bits. If detection would be without any problems in the first two cases, the third case would very likely not give a proper output at all.

For more information about the chosen sampling values and bits, please refer to paragraph 4.2; "AD Implementation".

3.2.2. Input: 12 Hexadecimal Characters

Now the limits of the system are known, the actual input can be given to the system.

For one ID, there are 12 Hexadecimal characters needed which can be given in by hand or can be randomly generated. The 12 Hexadecimal characters are one by one converted to bits, which form the ID string. The reason for choosing Hexadecimal characters is due to the ease in which it can be inserted and viewed in comparison to a long string of ones and zeroes.
For the quick calculators; this is indeed less than the 64 bit ID that was explained earlier. The ID is namely constructed as follows;

- 2 bits for future use
- 8 bits manufacturers' code
- 38 bits unique ID
- 16 bits CRC-code

The first 48 bits are given in by the user; the other 16 bits of CRC code are generated. Even the 2 bits for future use and 8 bits of the manufacturers' code are inserted manually or randomly because it gives maximum capability for debugging specific values and code strings.

3.2.3. Calculation of the complete bit stream: CRC code

CRC stands for Cyclic Redundancy Check. In other words, it is a routine that can be used for error detection. While many error detection schemes can be fooled when a few bits change, this is not the case with the CRC routine. Due to a specific system of shifting and EXOR-ing bits through a registry, a mistake very rarely slips through. That is, only when a proper scheme of shifting and EXOR-ing is chosen. These schemes are called polynomials.

The CRC-code is calculated over the total of 64 bits. The polynomial that is used for this calculation is X16 + X15 + X2 + X0, and is also known as the CRC-16 polynomial. A schematic view of this polynomial can be found in figure 3.2.4.

The register that is used for this CRC calculation has to be seeded with 1's when the CRC code of the EM4322 chip has to be calculated. If the code for an EM4022 has to be generated, the CRC register needs to be seeded with 0's before the calculation starts.
At the beginning of the calculation, the first 16 bits (starting with the most significant bit) are EXOR-ed into the register. After this, the rest of the bits are shifted into the register, following the normal procedure for the CRC code generation.

When the CRC code has to be generated, the last 16 bits of the 64 bits (that should actually contain the CRC code) are all 0's. The code that remains in the registers after all 64 bits are shifted through is the CRC code and is placed back in the bit stream, filling up the places that were '0' before the CRC generation.

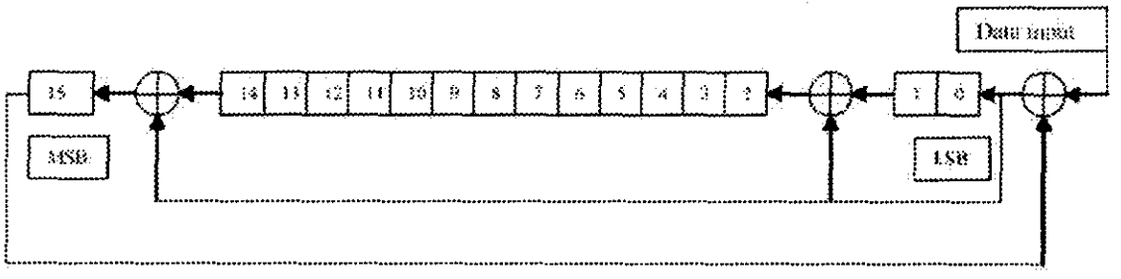

Fig 3.2.4: Schematic view of CRC-16 polynomial

3.2.4. Generating the Signals

The samples can be built up out of three parts; ID-signal, Additional waves and Noise. This is done to give as much output possibilities as needed. These parts are calculated separately. If the additional waves or noise are not needed, the addition of this part will be zero. When all three of these parts are generated they are added together forming one final sample that is written to the file.

3.2.4.1. ID-signal

When a specific sample of the output of the ID-signal should be low, a "0" is generated for the ID-signal; no values are added to the output sample by the ID-part. When there should be some information, the signal should be present, a value will be calculated. This calculation can again be done in two ways; the output should be a block pulse (UHF) or a 7MHz wave (DF).
If the output is a block pulse, it is simply given back as a "1" multiplied with the desired strength of the signal. In this case, that will be 50 micro volts, the smallest data signal expected. These high values are generated for in total one-fourth of the bit period, based on the Glitch encoding scheme, as discussed in paragraph 2.1.

In the case of the 7MHz wave, the output sample is calculated with the following formula:

```
SampleTemp = PulseStrenght * sin ( 2 * PI * I * 7 / 25 ) / ( 5 ^ ( I / 35 ) )
```

Formula 3.2.1: Generation of 7 MHz muting waves

"I" is the value of the FOR-loop at the moment this calculation is done. This calculation will also be done for one-fourth of a bit period, about 35 samples, calculating from a sampling rate of 25MHz. This explains the value of 35 in the divider at the end of the equation. The value of 7/25 is the frequency of the signal divided by the sample rate; 7 MHz.

3.2.4.2. Additional waves

Sine waves can be superposed on the ID signals as mentioned earlier. In some cases they can be noise from the outside, but sometimes these signals are generated from within the reader system. Especially these waves have to be kept in mind, because their strength can become quite extreme compared to the data signal. This is why the possibility of adding sine waves to the samples exists in the simulation. By default, there is a 0.7 V sine wave added to the system of the block pulses and to the 7MHz system, a 0.0 V sine wave is added. The 0.7 V was added due to the fact that interference of two UHF readers with each other was expected. This is not the case with the dual frequency system. Although

the wave has an amplitude of zero in this last case, and it looks like it is not present, it is easily added later on if deemed necessary.

3.2.4.3. Noise

The generation of the noise that is used in the samples, is calculated using six random numbers from which the average is determined. The average values that are generated this way form a close match to white Gaussian noise.

3.2.5. Result: Complete Output File

After every calculation of a sample, the sample is written to an output file, called "OutputWave.dat". This file will contain every sample that is needed for the detection later on and will not contain any additional information. Doing so, the file can be easily read into different programs to alter the values before detection, if needed. A small sample of a possible output file is given in figure 3.2.5.

During the process of calculating all the values, a second file is generated that is called "IDnumbers.dat". This file will contain the binary and hexadecimal values of the ID's. If it is requested, this file can be used to get hold of the original ID's that were in the samples. In the same file, there is also a value that gives the amount of ID's that was generated and the type of signal it was; block pulses or 7MHz waves.

These two output files now contain all the data needed for the next part; the detection.

```
-0.130000
0.024000
0.194000
-0.174000
0.048000
-0.074000
0.018000
-0.112000
0.062000
-0.066000
-0.046000
-0.110000
-0.108000
0.062000
-0.046000
-0.186000
-0.164000
-0.172000
0.042000
0.116000
0.190000
0.024000
```

Fig 3.2.5: Output Values

3.3. Detection Software

3.3.1 Filtering Signals

When initializing the program to perform the detection of the ID's, the file "IDnumbers.dat" that was additionally generated by the generation software, is opened. Two values are searched for in this file; the amount of ID's that should be present in the sample-file and the type of signal it is; 7MHz waves (DF) or block pulses (UHF). The second parameter plays a principal role in the functioning of the system; the sample rate is determined using this value, the filter kernels for both filters are tuned according to this value and the demodulation is changed.

3.3.1.1. High Pass filter

When these initializations are done, the High Pass filter will contain a kernel that is suited for a cut off frequency of 50 kHz when there are block pulses to detect (figure 3.3.1). There will be a cut off frequency of 1 MHz when 7 MHz pulses should be detected (figure 3.3.2).

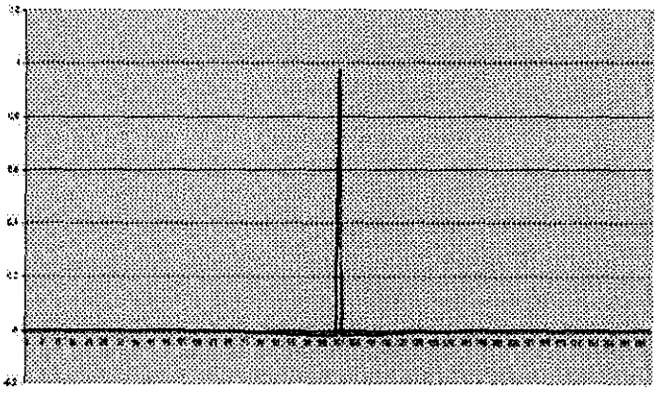

Fig 3.3.1: 50 kHz filter kernel (UHF)

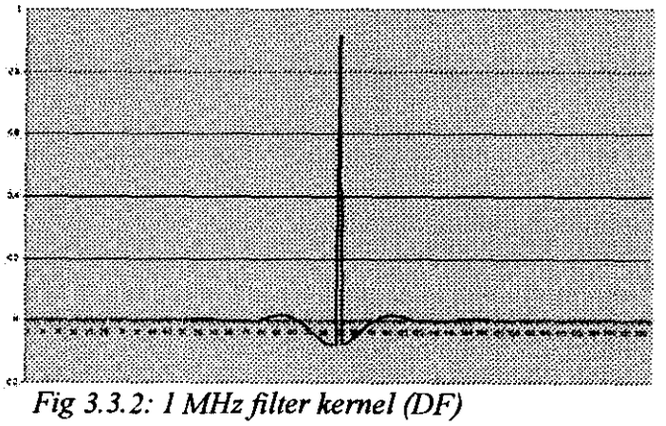

Fig 3.3.2: 1 MHz filter kernel (DF)

The reason for choosing a High Pass filter and not a Band Pass filter, as well as the major difference between the needed frequency and the cut off frequency, is due to some side effects of filtering. The filters that are used have a significant affect on the frequencies above the cut off frequency. It also has to be kept in mind that the filter kernels will be reduced in quality at the moment the size of the filter kernels is reduced. When the quality drops, the quality at the cut off frequency can change; these values further from the frequency of the actual data stream, ensure that the data signal is not affected more than is absolutely necessary.

At the moment the complete system will be implemented in hardware, the optimal kernel sizes and implementation parameters have to be found according to the amount of

computation power and memory available. It is mainly the High Pass filter that will cost the most computation power in the complete system.

3.3.1.2. Matched Filter

Although almost everything of the Matched Filter has already been explained, some aspects are interesting to note specifically for this system.
One of the last notes about the High Pass filter was that this would be the most computation-consuming part of the system. In the Matched Filter, this is actually one of the quicker calculations. As an indication, the filter kernel of the High Pass filter works with a main array of 200 points. The Matched Filter on the other hand has a size of 35 points with the 7MHz system. In the case of the block pulses, this kernel decreases to only 15 points. This can be accomplished with a small reduction of signal to noise ratio. Although it can be seen that the Matched filter is more beneficial, both filters do have their own specific applications.

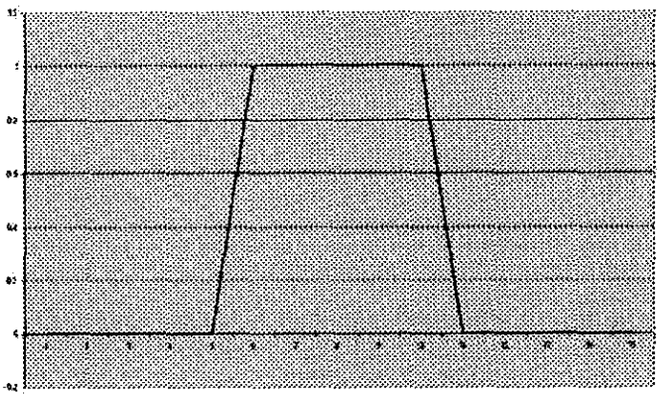

Fig 3.3.3: Matched Filter kernel (UHF)

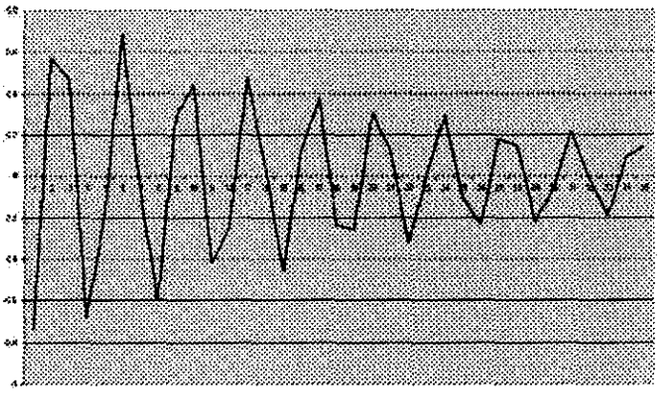

Fig 3.3.4: Matched Filter kernel (DF)

In figures 3.3.3 and 3.3.4 it is evident that the Matched filter kernels are shorter than the ones used at the High Pass filter. Although difficult to see because of quantization, this kernel represents a 7 MHz sine wave.

3.3.2. Bit Detection

3.3.2.1. Calculating average values

After filtering, the samples can be used for the detection. The first task is to calculate the average value of the first 3000 samples. This is done to form an idea of the strength of the complete signal. During the calculation of these points, a few calculations are made; the average noise strength, the average strength, and the average top value.

These values can roughly be calculated using different if-statements; looking for the lowest values, looking for the average lowest, etc.

3.3.2.2. Calculating the Bit Threshold

This value is calculated in order to estimate a decent value on which the bit detection can be based. This value is called the "Bit Threshold". By using this variable Bit Threshold value, it is easier to use one system in a number of environments. When a lot of noise is present, the Bit Threshold will automatically be initialized on a higher level than when there is less noise, while still keeping a value that is not too high to overlook bits.
The input for the Bit Threshold will be the signal directly from the filters. The output will be a zero or a one for each sample. The ones and the zeroes that are generated using this Bit Threshold only tell if a certain sample contains a strong signal or not. No information on the actual data can be deducted from these bits. Figure 3.3.5 illustrates this process.

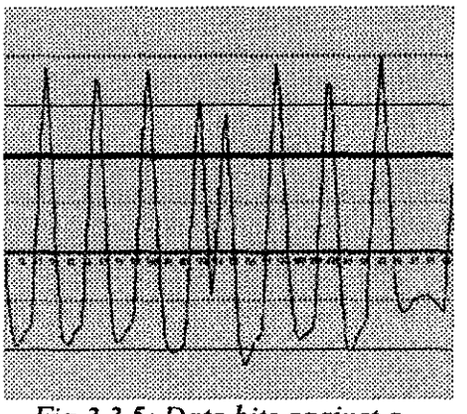

Fig 3.3.5: Data bits against a Bit Threshold

3.3.2.3. Determine Bit Period

Before the detection of an ID, the first "1" values that are generated by the Bit Threshold will be used to determine if they are indeed part of a data bit and if so, what the period between the ones and the zeroes is. In most cases, this first detected data bit will be a start bit, which has, for a fixed amount of bits (eight), the same distance between each other, thus making it possible to calculate the exact bit period. It has to be kept in mind that due to velocity and other factors, the speed of the signals can change and with that, also the bit period.
If a strange value that is out of the range for the bit period is calculated, the complete detection will be discarded and the detection routine will wait for a possible new ID.

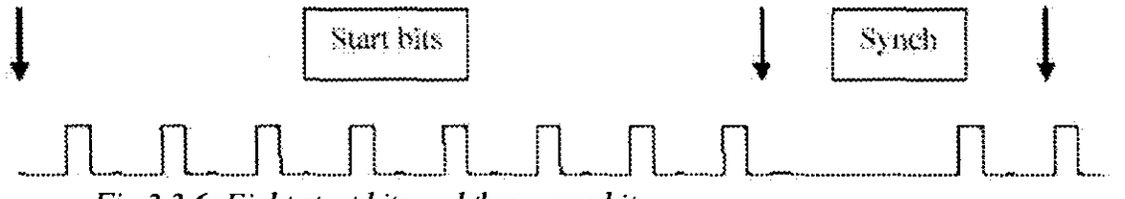

Fig 3.3.6: Eight start bits and three sync bits

When the value for the bit period is stable for at least four of the eight start bits, the system will start to look for the synchronization bits. These three "bits" are completely empty for two bit periods, followed with an encoded "1". Using this empty space, where

no sample should be detected as a "1", the detection algorithm can be synchronized on the first data bit, skipping extra start bits that might be present. This is necessary due to the fact that it might not necessarily be the first start bit of an ID that was detected. When the last synchronization bit is reached, the encoded "1", the detection system will skip exactly one bit period, ending up at the start of the first ID bit. Figure 3.3.6 illustrates this routine.

3.3.2.4. Detecting ID bits

When it starts to detect the data bits, the values obtained from the Bit Threshold are put in an array where all the samples for a complete bit period are stored. This information is then sent to the data decoding routine after the end of a bit period. During this detection of the rest of the ID, the average bit period and the amount of zeroes detected are reviewed once again. If this is far out of range, or when the values are too disturbed, the ID detection will be discarded and will reset to the beginning of the detection process.

3.3.3. Data decoding

3.3.3.1. Glitch-Manchester Decoding

The bits that were stored in the array for a complete bit period are compared for the first and the second part of the bit period. If the sum of the first part of the bit period contains a higher value than that of the second part, the output will be a one. If it is the other way around, the output bit will be a zero. This output bit is then one of the 64 ID bits, which is stored in a new array to wait for the CRC check.

The reason why this solution is chosen, and not, for example, a solution where the zeroes between two pulses are counted, is due to the robust nature of this system against noise. When by accident one pinch came through the filters and would get through to the detection, it would have been seen directly as a one when this system would be used. In the first solution, this pinch will be "averaged out" against the other values of the bit period.

3.3.3.2. CRC-check

After detection of all 64 data bits, the last error check, the CRC-check, will be performed. As already explained in the section about ID generation (paragraph 3.2.3), the CRC code runs over the complete 64 bits, where the last 16 bits are the CRC code.

The complete ID will be shifted through the register, in the same manner as when the CRC code was generated, using the same polynomial. The only difference is that now the last 16 bits are not zeroes but the CRC code that was generated. If the complete ID is without errors, the remaining value in registers should all be zeroes again. If there is one

or more 1's left in the register, the ID was not detected correctly. Although the number of errors in the detection of the ID can not be determined, it can be told if any errors are present. As also mentioned before, this CRC system is known to be one of the most robust systems in error checking. Quite a number of bits need to be detected wrong in a specific order, before the CRC calculation gives a wrong result.

3.3.3.3. Hexadecimal Output

The output of the CRC calculation will now be converted to Hexadecimal characters, whether the ID is incorrect or correct. In the case of a correct detection, only the first 48 bits are converted to Hexadecimal characters. If the CRC calculation detected an error, the complete string of 64 bits will be sent to the Binary to Hexadecimal converter. This way, it will still be possible to determine where the mistake was made; it might just have been in the CRC code itself. This system could easily be changed to a system that will only send out the correct ID's, discarding the ID's with a wrong CRC code if deemed necessary.
All these values are displayed on the screen as well as written to a file together with the time of detection, this makes further investigation possible. The output that is written to the screen can be compared to the values that were given to the file started with, "IDnumbers.dat". Using the number of ID's that should have been in the sample file, it can be determined how successful the detection was.

With the final calculation of the Hit Rate, the complete detection is finished and the success of the filtering and detection on the specific input can be assessed.

3.4. Optimization of Simulation

Note from the author: In the second phase of the development of the simulation software, Mark van de Pol, a trainee from the Saxion Hogeschool Enschede in the Netherlands, joined the team to sort out specific problems that arose during the testing and development of the simulation software. These were mostly practical problems that were not expected during the theoretical development of the algorithms. The following optimizations are a product of both our ideas.

There were roughly three major problems that came up during the development. These problems were encountered when actual signals were captured and analyzed. From these signals, the problems were derived and identified. This process of finding and identifying was partially done by our research group at the university, as well as by the employees of iPico.

The three problems were:

- Different sizes of signals
- Changes in the bit rate per ID-tag
- Changes in the bit rate during the broadcast of an ID

3.4.1. Different sizes of signals

In the original simulation, the size of the signal was always set to the minimum that would be available in the practical environment (figure 3.4.1). This way it was possible to check the quality of the algorithms to the edge of necessary needs; if the smallest signals were detected, larger signals should not cause any problems (figure 3.4.2). Another reason for keeping the same signal strength was that the incoming signals were expected to be approximately the same size when arriving at the reader.
The first statement is indeed true, although it had a few limitations on the performance of the system when it detected mostly ID's from a short distance away. If the system is always looking for a signal that is just above the noise level, it could be disrupted by a noise spike quite easily, while it could be that the data signal is actually much stronger. If the system would look for stronger signals, the signal to noise ratio is much better, therefore easier to detect ID's; the same signal, only looking at the complete signal, instead of only at the bottom of the signal.

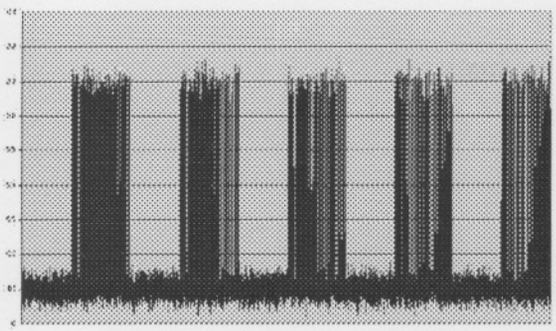

Fig 3.4.1: Same strength ID's

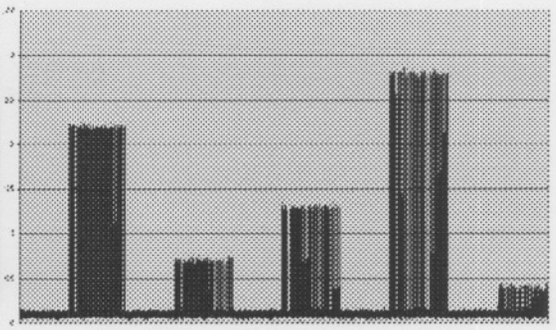

Fig 3.4.2: Different strength ID's

This problem can be tackled by calculating the average values, the step also done during the initialization of the system. In the first case, only the average of the first few thousand points was calculated to get base and top values for noise. When this process is continued during the rest of the detection, changes in noise strength can be detected quicker. Unfortunately, this does not count for the detection of the average top value of the signal. When a change in this value is detected, the system already needs to initialize the detection routine, therefore running out of time to slowly and properly adapt to the new values. Therefore, the system needs to adapt quickly. At the moment a pulse is detected, noise or data, the detection system will assume it is a data signal, until proven otherwise. When the pulse is handled as a data signal, the top of the first start bit can be found. This value can then be used, together with the calculations of the noise values, to determine a new Bit Threshold, specifically calculated for this ID signal. At the moment the ID is detected correctly, or discarded as noise, the Bit Threshold will be reset to its initial value.

Although this routine has a downside, which is that the system does not know anymore what to expect, the overall quality of the system is very likely to improve, because the detection becomes more robust for unexpected signals.

3.4.2. Changes in Bit-rate per ID-tag

The speed with which the information is transmitted from the tag to the reader is defined by the oscillator on the chip of the tag. Seen on the fact that every physical component that exists is not ideal, and the assumption that the cheaper the product, the bigger the fault margin, the aimed bit speed is often not met.
According to measurements from iPico, these deviations in quality can result in a thirty percent shift of the bit rate in both ways. When this signal is sampled, there will also be a possibility of thirty percent more or thirty percent less samples per ID.
As discussed earlier, the system that is used can find small irregularities in bit rates, looking from the start bits. This is, however, only when the changes are small, roughly between zero and five percent. The main task leading from this problem is to find a scheme that is able to derive the bit rate from the start bits, without making errors, seen that this will directly influence the rest of the detection.

The solution that is found for this problem is a combination of two techniques; calculating the average of a few bit periods, and searching for the largest deviating value. From the start bits, there are four bit periods taken, and stored in an array. If one of these values deviate more than a certain percentage from the other ones, it will be discarded. From the remaining values, the average is taken and used as the bit period.
This system has proven to be very stable and robust. Even when spikes occur in the start bits, it can still be filtered out, without affecting the average bit period.

3.4.3. Changes in Bit-rate during an ID broadcast

Unfortunately, the change in bit rate between ID's is not the only irregularity caused by the oscillator. A second problem that was found is even more complex than the first.
When a tag is powered up by the reader, the power is kept in a capacitor until there is enough energy to broadcast one complete ID, as explained in chapter 2. The oscillation frequency of the oscillator is also depending on this value; when there is too little power, the oscillator will not be able to keep its frequency constant. As the power decreases, the frequency will also decline. In ideal circumstances, this voltage should not decrease to a point where the frequency can drop, but unfortunately, this can happen according to measurements of iPico.
According to these same measurements, the bit rate, which is derived from this oscillation frequency, can roughly vary between +5 and -15 percent during the transmission of an ID. Explained more rationally, the bit rate at the end of an ID can be anywhere between five percent faster and fifteen percent slower than at the beginning of the transmission of the ID code.

3.4.3.1. Possible Solution

The function that determines the bit rate at the beginning of an ID is, as a stripped version, used again to determine the bit rate during the detection of the ID bits.

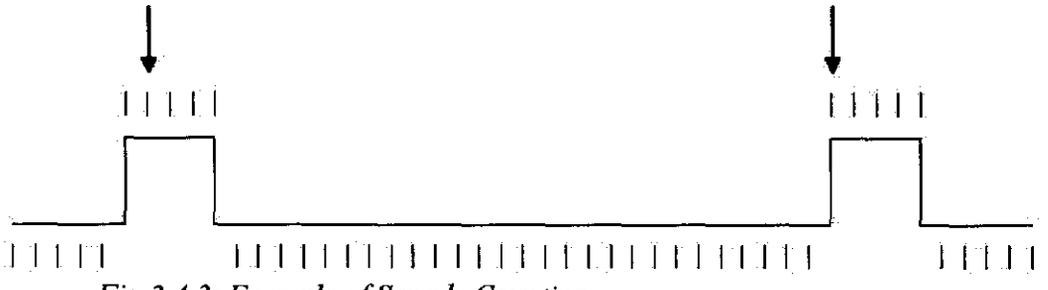

Fig 3.4.3: Example of Sample Counting

In figure 3.4.3, at the first arrow, the sub routine starts counting, during a high pulse. When this pulse stops, this will be noticed but the sub routine keeps counting. At the moment the first high pulse is detected again, the routine stops counting. The amount of

samples that is counted at that moment forms a value for the average bit period, therefore for the average bit rate.
This value is stored in an array that contains the values from the last three times this bit period value was determined. The average of the values in this array is compared to the value that is forming the bit period at the moment after it is determined if the used bit period should be changed or not.

3.4.3.2. Side Effects

There are, unfortunately, two things that have to be kept in mind while using this method. If there is a noise spike during the counting, it can be seen as the next pulse. When this happens, it can negatively alter the average value that is used to determine the value for the bit period.
To ensure that this does not happen, certain limits are set within which the counted value must fit. For example, if the average value at that moment is 27, a newly counted value cannot be lower than 24, or higher than 30. The range of these margins are quite crucial for this process; if they are set too small, bit rate changes will not be noticed, if they are set too big, the bit rate could easily be influenced by noise like spikes.
To find an optimum for these values, empirical testing has been done, there is regrettably no way to calculate these margins. Testing has proven that certain limits are better than others, without being able to form a hypothesis. This is mostly due to factors like quantization of the signal by sampling and unexpected noise levels.

The second problem involved the way the bits are encoded; the glitch encoding. In the figure above, the amount of samples between every bit can easily be determined, exactly one bit period. When data is broadcasted in ones and zeroes, the time between two bits can change, depending on the way the bits are arranged.
Although the timing at the place in the bit period is different (figure 3.4.4), the time between two high pulses is the same when a one follows a one and when a zero follows a zero, in both cases this is three quarters of a bit period. In the case when a zero follows a one, this gap is bigger, due to the situation of the high pulse within a bit period. This length between the two high pulses becomes five quarters of a bit period. (Note that this is indeed longer than one bit period) In the last case, where a one follows a zero, the spacing is only one quarter of a bit period.

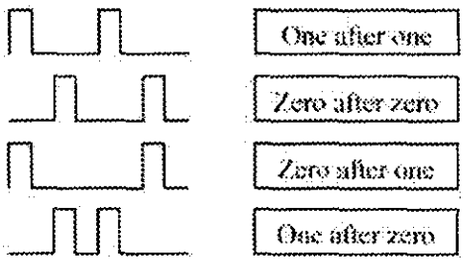

Fig 3.4.4: Different bit arrangements

3.4.3.3. Margin Calculation

When this scheme is merged with the previous problem where margins were introduced to count the amount of samples per bit period, an interesting quantity of margins is formed.
In all three cases, the 1-1 and 0-0 situations are seen as one, an upper and lower margin is calculated, derived from the bit period that is calculated at the beginning of an ID. This scheme of margins is clarified in the figure below, where in the bit period is 20.

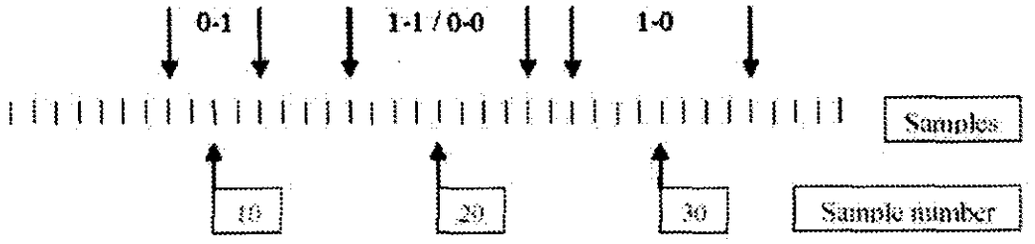

Fig 3.4.5: Limits for margins Bit Period

When a counted value is determined to be a valid number for the amount of samples per bit period, the value is multiplied with a certain number to get a value that can represent the actual bit period. For example, when a value is counted that falls within the margins of a 1-0 detection, the value will be multiplied by 0,75.
This calculated value can then be used again for the array of previous bit period counts to possibly form a new value for the running system.

As it can be seen in the last few paragraphs, there are a lot of margins set and assumptions made to produce a system capable of detecting changes in the bit rate within a data signal. This is one of the reasons why this scheme is only used next to the original detection scheme; if it fails, nothing will happen to the original detection, if it does work properly, the quality of the detection rises.
Nevertheless, it has been proven by simulations that the system developed is capable to reach the limits that were discussed earlier, although it is with loss of quality in situations where these deviations are small.

3.5. Software Outlook

For the final software, a graphical interface is designed to show and test the detection algorithms. This software is built up out of two components. The first component is the IDgenerator (figure 3.5.1), which contains all the routines for the generation of the signals, the other component is called PulseMatch (figure 3.5.2), which contains the routines for the actual detection of the signals. The name PulseMatch is derived from the type of matched filter used in the software, which formed the base for the development of the detection software, as discussed before.

All the controls that are mentioned here are only briefly discussed. If more information is needed, please refer to the last few paragraphs, which contain a more thorough explanation.

3.5.1. IDgenerator

Sample bits:
This is the amount of sample bits that will form a value of the quantization that will be applied on the signal. The output values that are written to the file will seem as if they were sampled with this amount of bits.

ID's:
This is the total amount of ID's that will be generated. Only the length of the output file will be influenced by changing this value, and not the space between ID's.

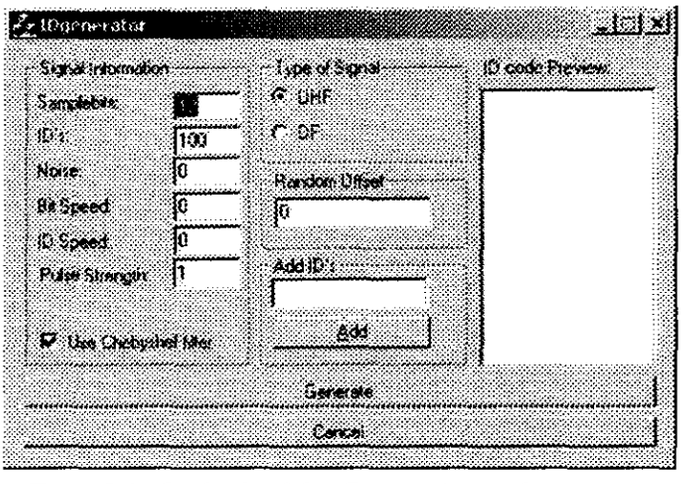

Fig 3.5.1: Outlook of the Generation software

Noise:
Changing this value will have an impact on the amount of noise that will be added to the signal during the generation. The noise represents white Gaussian noise. The value is derived from the original pulse strength, 50 micro volts. When the value 100 is given, the strength of the noise will be more or less 50 micro volts.

Bit Speed:
This is the amount of change in bit speed/rate within the ID itself. A positive value indicates that the bit rate decreases; while a negative value indicates the opposite. A value of 15 will mean 15 percent longer time of transmission of the last bit than that of the first.

ID Speed:
This value represents the percentage that the bit rate of the first bits of an ID differs from the original bit rate. A positive value represents a slower bit rate, while a negative value represents a faster bit rate.

Pulse Strength:
This is the number of times that the signal is larger than the initial value. The initial value is 50 micro volts.

Type of Signal:
If the ID-bits are generated as Block pulses or as 7 MHz waves. This also determines the sample rate of the generated samples; 5 MHz for block pulses and 25 MHz for the 7 MHz waves.

Random Offset:
When a random function is triggered the first time in a program, it will always output the same value if it is run on the same location. If you want to simulate a practical situation, this is mostly not desirable, although it can sometimes come in handy to validate old simulations. If the same Random Offset is given, the output values will be the same. With different offset's, completely different ID's will be generated.

Add ID's:
If there are specific values of ID's that have to be tested, they can be included in this part. When there are less ID's added than the value given at "ID's" the rest will be generated randomly.

ID-code Preview:
The ID's that are generated are shown here. These values are also written to the file IDnumbers.dat to enable later examination and debugging of the detection software.

3.5.2. Pulse Match

Input File:
Here the name of the file which contains the samples of the input signal can be chosen. The file normally generated by the IDgenerator is OutputWave.dat, but if there are some other calculations done on the samples in between, it is possible that the filename has been changed. Please note that the values in the file IDnumbers.dat are not changed, these values are still the same as when they were generated.

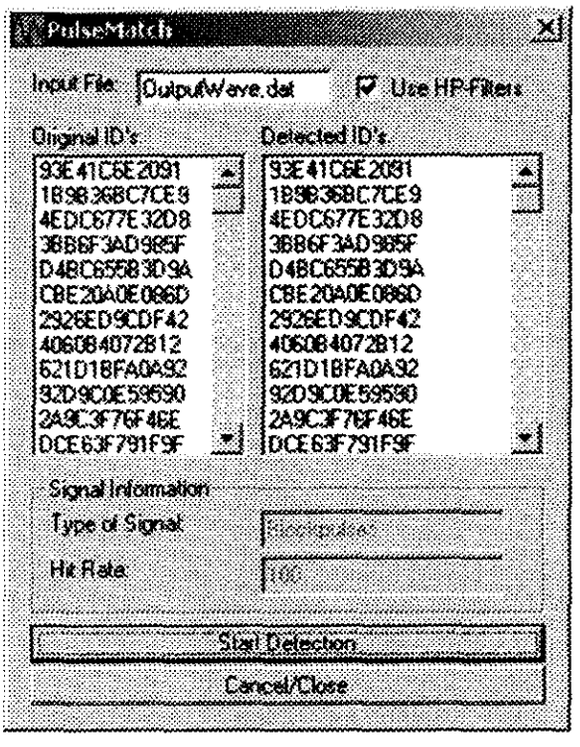

Fig 3.5.2: Outlook of the Detection software

Use HP-filters:
If it is necessary to exclude the high pass filters in the detection simulation, they can be switched off with this control. When the signal consists of block pulses, the high pass filter will be set on 50 kHz (one-hundredth of the sample rate), when 7MHz waves are present, the high pass filter will be set on 1MHz.

Original ID's:
The ID's that were written to the file IDnumbers.dat are shown here again. Although the detected ID's are not examined against the original ID's if they are correct, it can provide the user with a better view in the errors that are made in the detection, and trace specific problems. The reason why the errors are not determined per ID is that problems will arise if a complete ID falls out during the detection. From that point synchronisation with the original ID's will be impossible and the results may become unexpected and unreliable.

Detected ID's:
The ID's that are fully detected are shown here. If an ID is detected partially or if there were problems with the synchronizing at the beginning of an ID, the ID will be discarded completely. If an ID is detected completely, but there are errors in it, that were revealed while performing the CRC-check, the complete ID including the CRC code will be shown.

Type of Signal:
Here is the type of signal shown that should be found in the file with samples. The value of this box is retrieved from the file IDnumbers.dat. The value of the "Type of Signal" is also used to determine the sample rate of the file and the contents of the filter kernels.

Hit Rate:
This is the percentage of ID's that is detected without any errors, divided by the number of ID's that should be in total in the file. The ID's that contain errors, but can still be read as well as the ID's from which the CRC bits are incorrect, are not included in the Hit Rate. These hit rates are not logged.

4. Hardware

Now that the complete set-up of the detection scheme is developed, it will be possible to look for a hardware solution that is capable of executing the algorithm that is developed.

To begin with the hardware will be discussed, together with the reasons for choosing this specific type of hardware and the limitations of these products.

Thereafter the Analogue to Digital converters that are used to make it possible to sample the incoming signal, and how to connect this to the main hardware, will be discussed. This is especially important because these components are in some cases very difficult to set up. Besides that, these components are an important obstacle for optimizations and the amount of noise in the incoming signal.

The first implementation of the algorithms in hardware was completely based on the simulation software, but it contains a lot of non-optimized code and algorithms. The next paragraph will deal with optimizations that were done specifically for this hardware implementation.

Subsequently the most important parts of the source code, an explanation of their functions and where they are derived from will be addressed. This is mainly to gain insight into what actions are executed and what specific software components are based on it.

Finally, the complete system will be shown, including a scheme of all hardware and software components that makes a correct detection possible.

4.1. Getting to know the Hardware

4.1.1. Why this Specific Hardware

The algorithm that is developed, as discussed in the last chapter, consists of many different kind of operations; Boolean operations, integer based operations and floating point operations. Therefore, it is crucial to look for a solution that is able to perform all these tasks.

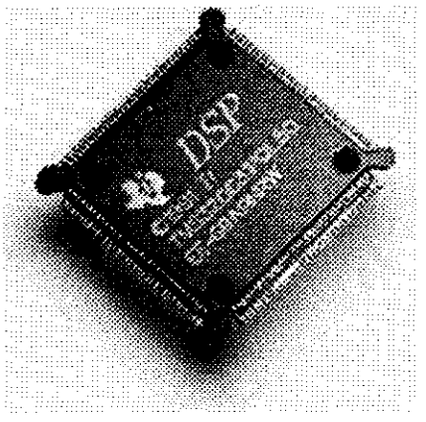

Fig 4.1.1: DSP processor

The most important decision, although also the most obvious, is that a DSP processor is chosen (figure 4.1.1). These processors are designed to handle different streams of data and performing as many instructions as possible per second on them.

From Texas Instruments, currently the leading DSP processor company, there are three lines of processors in production;

- C64xx
- C55xx
- C67xx

All three lines of processors are still in development, which means that their capabilities and speeds are still increasing. There are, however, a few major differences between the three of them.

The C55xx processor range is aiming at the most cost-and power-effective part of the market. With processor speeds that are a little below the high end range, and possibilities that are also limited, these processors are developed to fill the gap of devices that need more capacity than a basic processing unit, but do not need the high range performance.

The C67xx processors have all the extras that the C55xx series lacks, and is specifically designed for floating point operations. This means that it will be possible to process nearly every type of data. The speed of this type of processor is also one of the highest in the class of DSP floating point processors. At time of writing, a processor is available of 225 MHz.

The C64xx processors are based on architecture for fixed point operations (short and integer data types). By limiting the type of information that can be handled, the architecture can be optimized and higher speeds are possible. Roughly, it can be said that the speed for these processors are a factor two higher than those of the C67xx series.

Based on these specifications, the C67xx processor was chosen, due to its compatibility with different data types and therefore easier porting of source code from the simulation software to the hardware environment.

4.1.2. TMS320 C6713

The best processor in this range at the moment is the C6713. Similarly to some other processors, this one is available on a development board, called a DSK (Development System Kit). On this board, there are several other components that make it possible to test the processor thoroughly, as well as the developed algorithms.

Here are some of the key components that can be found on the development board, called TMS320C6713 DSK.

- TI C6713 processor, 225 MHz
- 8 Mbytes SDram (16 Mbytes on newer revisions)
- Expansion connectors for additional input/output boards, including optional PCI connector
- JTAG emulator hardware (necessary for debugging processor)
- USB connection
- 512 Kbytes bootable flash ram
- On-board audio connectors with 96 kHz Analogue-Digital converter

Known limitations and problems of the DSK

- Very fragile, even for a development board. During the testing period, a few of the boards were blown without a valid reason.
- Very little knowledge concerning hardware changes between versions of the board. For example between the 6711 and 6713. According to Texas Instruments, there should not be any difference between the main architecture of both boards. Unfortunately, when some problems became more apparent, it was discovered that there were hardware changes between both versions.

4.1.3. Code Composer Studio

The DSK's of Texas Instruments can be programmed using a program called Code Composer Studio (figure 4.1.2). This program has two main functions; it compiles the programs for the processor and it maintains the link between the board and the host computer for debugging and data transferring purposes.

Fig 4.1.2: Code Composer Studio

Besides its basic functions like compiling and debugging tools, there are a few

specific tools which are interesting for the RF-ID detection application. These are;

- C-compliant
- Text-based/Visual-based compiling
- DSP/Bios
- RTDX
- Optimization levels

These components will now be explained more thoroughly.

4.1.3.1. C-compliant

While the main language for computer programs has been the language C for sometime now, the mainstream smaller processors were still developed using assembler. Code Composer Studio searches for an optimum between both situations by making it possible to work in both languages. The code that is transferred to the processor after building the application is still written in assembler language, but the main language wherein the program is written, is in C.

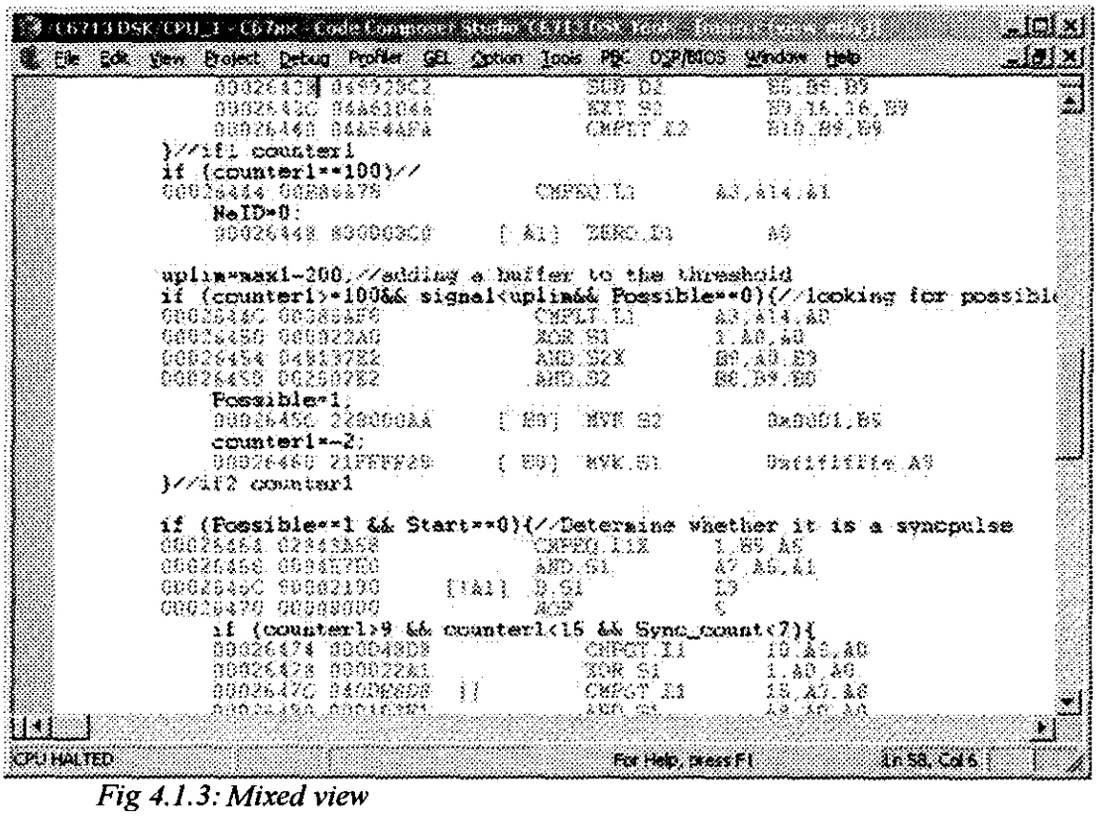

Fig 4.1.3: Mixed view

This has some advantages. It is easier to work in C, because of its better overview and faster programming. However, it can be that the code written in C is not optimized when compiled to assembler code. A special function is built in Code Computer Studio that

shows the C code as well as the assembler code on one screen (figure 4.1.3). Using this, it is possible to see immediately how many instructions in assembler are needed to perform one line of C code.

The assembler code can, however, not be altered directly, this has to be done by altering the C code. To make this step easier, some functions are written specifically for this compiler. When these functions are used, the most optimized code will be generated automatically when compiling. Also in the help files of Code Composer Studio a lot of examples of optimizations for this specific step between C and assembler can be found.

Concluding; by using the C language, it is possible to quickly write a program and by using assembler, it is possible to thoroughly optimize the written code.

4.1.3.2. Text-based/Visual-based program linking

Although a lot of optimization can already be done when the code is examined in the assembler language, there is another form of optimization that can easily be added to a program. When a program is built, it goes through several steps; first the compilation of the program, then the linking and then placing the program into the memory. This linking process can be done in two different ways in Code Composer Studio; using a so-called "Text-Linker" and a "Visual-Linker". Standard, a program uses the Text-Linker. The function of the linker is basically to allocate specific parts of the memory for the program and put all the blocks in such a way that the program knows where to find the code.
The "Text-Linker" does this process roughly in a straight-forward way; only based on a few parameters that were given to the project and the specific types of memory. Although it is the easiest, it is not always the most optimized way.

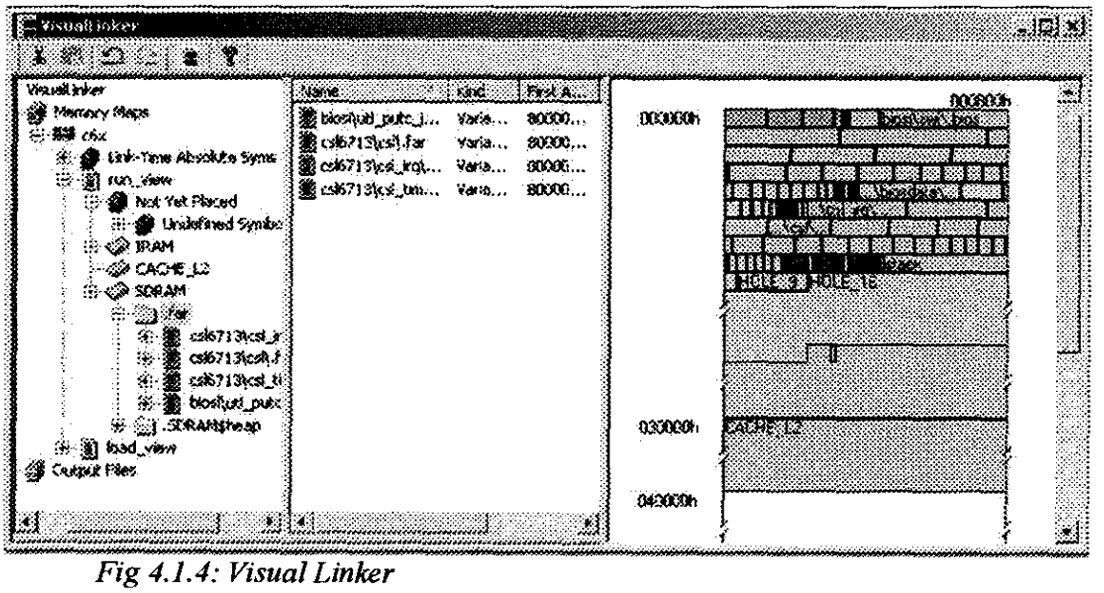

Fig 4.1.4: Visual Linker

When using the "Visual-Linker" (figure 4.1.4) more input can be given to the linker program, which makes more optimizations possible. Using this, certain variables of parts of the source code can be placed in different parts of the memory. When there is for example a block of source code that is only executed once every few seconds, it can be placed in a part of the memory that is further away from the processor, in comparison to a piece of source code that has to be executed every few milliseconds.

A part of the memory that is further away from the processor is for example the SDRAM (external) compared with the IRAM (internal). If a read action to an address in the IRAM would cost three clock cycles, a read action to a SDRAM address could take five clock cycles. By determining how often specific information is needed, program blocks can be stored accordingly, thereby optimizing the code and shortening the execution time.

4.1.3.3. DSP/Bios

See paragraph 4.1.4 for further information

4.1.3.4. RTDX (Real Time Data eXchange)

When a module or a program is running satisfactorily, it is preferable that as little as possible interaction with Code Composer Studio takes place when receiving output data from the development board. This can be done by using a RTDX solution.

Via a certain routine within the program, information from the development board will be captured by the development environment and placed in a specially allocated buffer in the memory of the computer. This buffer can then be accessed again by a program, separate from Code Composer Studio.

4.1.3.5. Optimization Levels

The last interesting option within Code Composer Studio is the different optimization levels. When a program is normally compiled and built, it is done straight-forwardly, without any optimizations. When using one of the optimization levels within the build options, a few aspects will be looked at when the program is compiled and built. This includes looking for possibilities for pipelining. Pipelining makes it possible to execute more than one instruction at the same time; for example a calculation within the CPU and the retrieval of a variable from the memory to the cache.

When using these optimization levels correct, they can deliver a performance increase of up to 75%, depending on the type of program and the amount of variables and components that are used.

4.1.4. DSP/Bios

Although it is actually a part of the Code Composer Studio software, the implementation of the source code into the hardware is intended for this component as well. Because of its importance, some key components will be highlighted quickly to give an idea of what the core functionality of this module is and what their function is for the detection program.

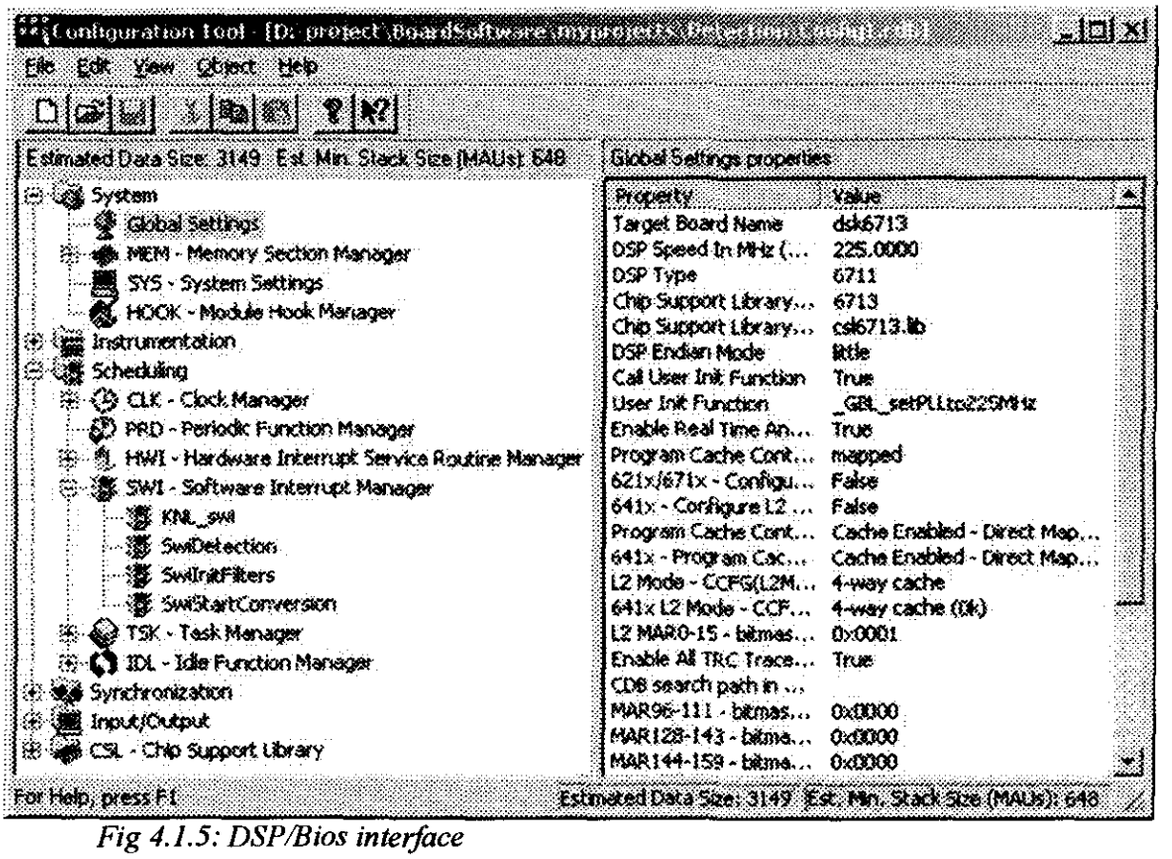

Fig 4.1.5: DSP/Bios interface

Using the DSP/Bios system (figure 4.1.5), most of the hardware components can be set-up easily, as well as a lot of the functionality within the processor. For example, using this interface, the different types of memory can be easily addressed. Even better, for every specific development board and simulation environment there are already templates available that contain all the components which that specific board contains. Therefore, no setup is needed to get the board up and running.

This flexibility makes it possible to let the board run a simple program within five minutes after setting it up, but also makes it possible to include more complex schemes that are using different input and output channels.

Some of the components that are used in the simulation implementation will be discussed briefly on what their function is, and why they are important for the implementation.

4.1.4.1. EMIF (External Memory Interface)

This module takes care of a flawless interface between the major memory components on the test board and the processor. On the C6713 board, there is an eight megabyte SDram module that is configured and managed by the EMIF module. Settings like refresh rate and number of banks and address lines, configurations specifically for this type of RAM, can be easily given to the EMIF module, without spending unnecessary time in finding out how to exactly connect these modules to the main board.

When the system itself is set-up to work with a specific board, in our case the 6713-board, the software will ensure that all the parameters for the EMIF module are loaded into the memory before the program is downloaded on to the board (see "the usage of .GEL-files in the help files of "Code Composer Studio")

4.1.4.2. Memory Section Manager

Although these values are mostly already configured when the right template is chosen for the development board, the Memory Section Manager still performs an important task, especially when the Text-Linker is used for program linking.

Using this tool it is possible to configure where specific parts of the software have to be placed in the memory and in what memory modules. When there is for example a huge buffer that will be used during the program, but only a small part of it is used now and then, it will be better to place these values far away from the processor to make more space for the more often used variables and code.

4.1.4.3. LOG

Its task seems small, but the possibilities are quite large, when using the LOG component to send data from the board to the host. This component can address a certain buffer on the board for logging to the host. These messages can be written rapidly into the program and can be placed anywhere in the source code, making it a great tool to use for debugging and testing. An extra advantage of this LOG system is the fact that it does not interrupt the processor during its task. Because this component is configured to be on a very low priority, it uses very little processing time, and the log information will only be sent at the moment there is time for it, or when the buffer is full.

Combining these characteristics creates a very handy tool for debugging software while running. Although this component is very basic, it becomes important when the behaviour of a program has to be monitored while it keeps running.

4.1.4.4. SWI (Software Interrupt)

Within the DSP/Bios, there are two different ways of generating an interrupt; hardware interrupts and software interrupts. While hardware interrupts are mainly used for communication between components, software interrupts can be easily used within program routines.

As will be discussed in paragraph 4.3, batch processing and cyclic detection systems become important in this type of detection systems. As a side effect of these type of systems, functions become nested. Especially in the situation where a program should run indefinitely, this behaviour should be monitored carefully and where possible, avoided. If a nested routine is not closed properly, it can have an influence on the processors' stack, and can form a buffer under run.

This problem can be dealt with by using software interrupts. When for example a buffer with incoming samples is empty, the routine should be stopped and a different routine started. When this is done using software interrupts, all routines are handled decently, without standing the chance that parts of the old routine remain in the cache of the processor.

4.1.4.5. EDMA (Enhanced Direct Memory Access)

As DMA (Direct Memory Access) is already known for its improvements in data transfer and CPU helper, because it does not necessarily go through the processor, the EDMA version goes one step further in this process to optimize these profits. Using easier techniques to address an EDMA channel and more efficient data transports, the amount of time/processor time, becomes close to zero.
This technique is used in the program to form the connection between the AD converter and the main memory.

4.2. AD-converter Implementation

As already discussed in the introduction of this chapter, the Analogue to Digital converters (hereafter referred to as AD converter or ADC) can be an important factor in the detection process. Not only can its sample rate influence the processing time and the quality of the incoming signal, but there are also major differences in how these devices are connected and how they send their data through to the main processor. A lot of these features can have an influence on the quality, performance and possibilities of the reader system.

Therefore, here are the AD converters that are used in the project, with their set up, connection scheme, specific problems and qualities.

4.2.1. Used Boards

The reason why there is more than one board used in the project, is because of the problems that the first board generated.

When the 6713 processor was being explored and its capabilities looked at, it became clear that the extension busses that are already on the board are perfect to connect an AD converter to, also seen on the fact that there is a range of products available from Texas Instruments and third party companies that are using this extension bus. Choosing one of these ADC boards would make designing special input and outputs, protocols and synchronization schemes superfluous.

The board that was chosen is the THS1206 EVM (Evaluation Module), which is based on the twelve bit 1206 ADC chip (figure 4.2.1). This board is capable of sampling up to six mega samples per second.

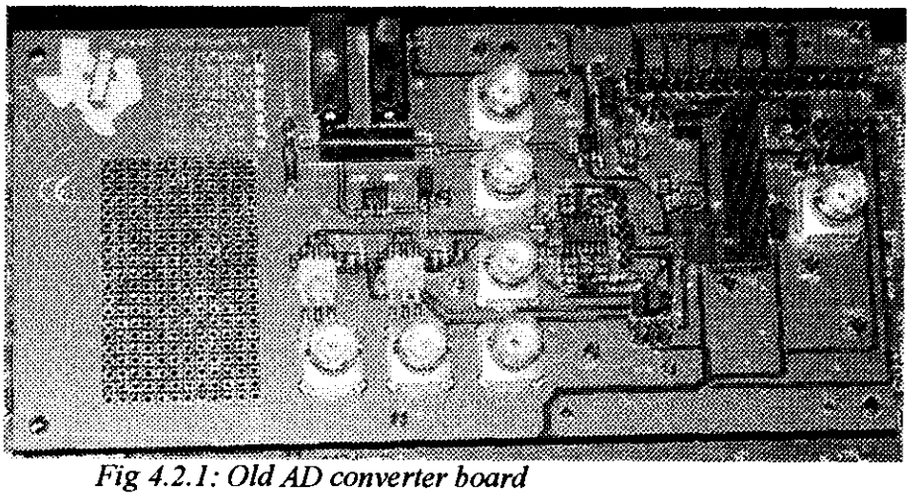

Fig 4.2.1: Old AD converter board

Quickly after this board arrived and was set up, it became clear that it was not that easy to connect to it and to receive data from the input signal. Due to unfamiliarity with the hardware and bad documentation, it took quite some time before a signal was received from the input. This was finally accomplished thanks to many different people from the Texas Instruments support centre.

When this incoming signal was further examined, it was discovered that the problems were not over yet; the incoming signal was shifted bitwise, the amount of noise added by the ADC was quite a lot and the board was far from stable. Although most of these problems could be solved, it still took a lot of processor time which could be put to better use for the detection algorithms.

At the moment the news came that a new version of the THS 1206 board had been developed, it was immediately decided that this one would be purchased to see if there were any improvements.

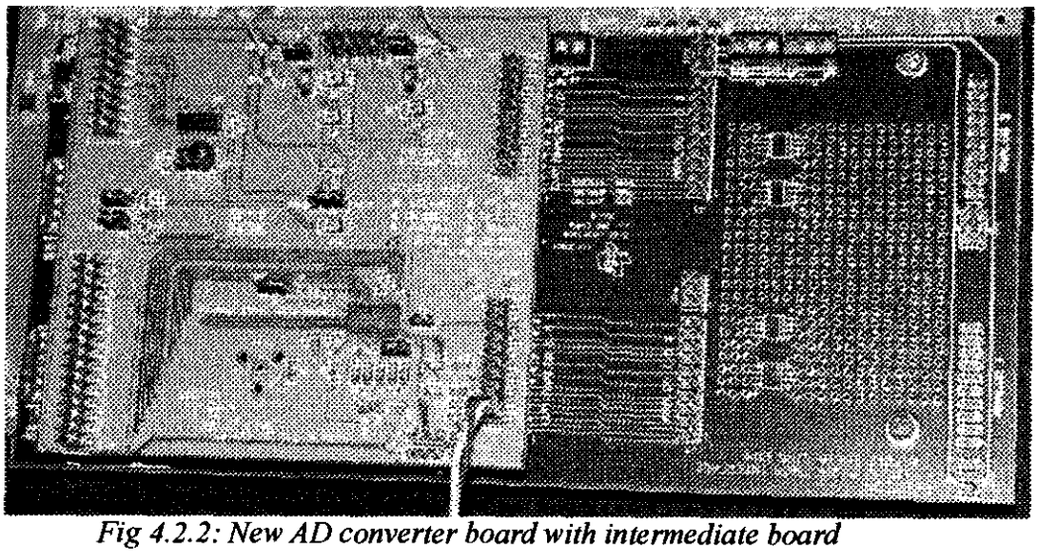

Fig 4.2.2: New AD converter board with intermediate board

This new AD converter consists of two boards, an intermediate board and the actual data converter (figure 4.2.2). The main function of the intermediate board is to form a solid connection between the data converter and the main board, which was previously done by the ADC board. The new ADC board is more simplified but contains the same version of the THS 1206 chip.

Although there were also some problems with the new boards, it was still an improvement on the old system, as it will become clear later on. First, some information about the main functions of the boards and how to connect them.

4.2.2. Hardware Information

4.2.2.1. THS 1206 EVM, rev. 1.2

This is the first board that was purchased and at the moment of purchasing also the only one available with the THS 1206 chip on it.

Key components:
- Four single inputs
- Max. 6MHz sample rate when one channel in use, max 1.5MHz when four channels are in use
- BNC connectors.

Explanation of its function within the project:
- Sampling the incoming signal with one input
- Converting the input to a twelve bit value
- Putting the twelve bits of the sample value on pins of the connector
- Wasting time and desk space

Known limitations and problems with the AD board:
- Weak documentation
- Difficult to interface with newer version of DSP processor boards
- Low input impedance on sample channels
- Unstable

4.2.2.2. THS 1206 EVM with 5-6K intermediate board

This is the second board that was purchased after it became clear that this was the new version of the old system.

Key components:
- Same amount of inputs and frequencies as old version
- Pin connection instead of BNC connections
- More than one board can be placed on intermediate board

Explanation of its functions within the project
- Basically the same as the old board

Known limitations and problems
- Hardware changes have to be made before the board can work properly
- Based on two boards, instead of one, therefore bigger system and less simplicity
- The pins for the input channels are likely to be more sensitive to noise

4.2.3. Jumper Settings

The jumpers that have to be set to get the communication between the ADC and the TMS board are the following:

4.2.3.1. Old ADC:

J1, J2, J7, J13 – Shunt pins 1-2
J3 - Shunt pins 2-5
J4, J5, J6, J11 - Open
J10 - Closed
J12 – Shunt pins 2-3

4.2.3.2. New ADC:

W1, W2, W3, W9, W10 closed
W5, W6 - shunt pins 1-2 (left two pins)
W11 open
W7 and W8 soldered closed pins 1-2 for THS1206M
Supply voltage from DSP, CLK from Timer 0, Input AINP
AD converter address set by W4 on EVM and W1 on 5-6K Board:
0xA0024000, shunt left two pins (pins 1-2), W1 Open
0xA0028000, shunt centre two pins (pins 3-4), W1 Open
0xA002C000, shunt right two pins (pins 5-6), W1 Open
0xA0000024, shunt left two pins (pins 1-2), W1 Closed
0xA0000028, shunt centre two pins (pins 3-4), W1 Closed
0xA000002C, shunt right two pins (pins 5-6), W1 Closed

4.2.3.3. 5-6K Interface Board REV B:

W1 open: AD converter address: 0xA0024000, 0xA0028000, 0xA002C000
W1 closed: AD converter address: 0xA0000024, 0xA0000028, 0xA000002C
W2, W3 - Shunt pins 2-3, Use DSK Digital 3.3 and 5V supply
W4, W5 - Shunt pins 2-3, Use DSK Read and Write Strobe
W6 - Shunt pins 1-2, Use DSK DC_CE2 (not used on EVM)
W7 thru W12, Shunt pins 1-2, Serial ports unused, INT's NOT inverted. Can invert parallel INT source via W7.
J14 - Shunt pins 1-2 - Use TOUT0 (marked as TOUTa)
J13 - Shunt pins 1-2 OR 3-4 - Use DSK ExtInt 5, marked "b"

4.2.3.4. Further information needed for set-up

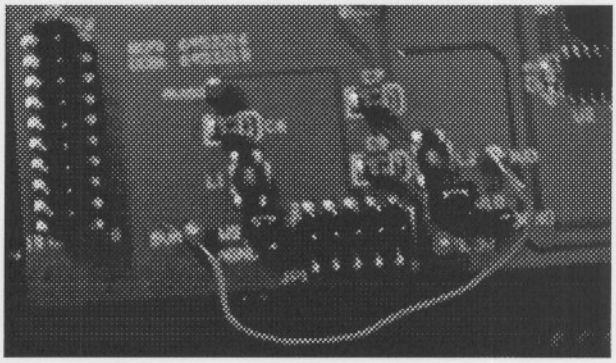

Fig 4.2.3: Connection between 5VA and 5VD

The old Analogue-To-Digital converter board was "quite easy" to install without extra connections that have to be made in order to function properly. The new board has some issues that have to be noted.
The most important point is that the analogue circuit on the AD board does not get power from the main board by default for some reason. It is not known yet if this is a design flaw or if it is on purpose. What is known is that when this connection is not made manually the AD converter will only give the value 4095 (Highest value possible). To correct this problem, the following connection has to be made. A wire has to be connected to two test-pins on the AD board, between test pin +5VA and test pin +5VD, as shown in figure 4.2.3. These test points can be found on the lay-outs of the board and on the board itself. They are labelled accordingly.
According to Texas Instruments, a wire can also be connected on J2 of the intermediate board, but this did not give the proper result in our scenario. This could however be investigated more properly when the need arises. For more information about J2, please refer to the manual of the 5-6K intermediate board. (Texas Instruments reference number SLAU104A)

4.2.4. Important differences between the ADC boards

The next paragraph gives a quick overview of the differences between the old ADC and the new one, as well as a few quick points to give an idea of both boards.

4.2.4.1. More than one board on new intermediate board

For the old board, the connection address is 0xA0020000. The new board has more than one address that can be set to receive data. This has been done to make it possible to use more than one interface board on the system. The address can be set using jumpers on the AD converter and the intermediate board.

4.2.4.2. No BNC connectors on new board

On the old board, there is a BNC connection for every input. On the new board, there is only one connector for the inputs. Although this is a bit inconvenient, the reason for this

is quite logical. As discussed, the new type of interface boards can be stacked onto each other. When stacking these onto each other, all the connectors are linked with the new board that is on top. When BNC connectors would have been used, it would become very difficult to port these through to the next board. Therefore a simple pin connector is much easier.
To make the connection accessible for data input, there are special connectors available with a BNC connection, which can be plugged into the ADC.

4.2.4.3. Less noisy

Although extreme testing has not been done yet, it can already be said that the new board contains less interference with the surroundings. The old board produced quite a lot of noise when it was sampling, measured from the signal that was found on the oscilloscope and the signal that was captured. It is not known what causes the main differences, but the signal quality improves significantly.

4.2.4.4. More stable

Once again, although stress testing has not been done yet, it can already be mentioned that the new AD converter is also more stable than the old board. While the old board freezes after a few runs of samples, the new board stays active.

4.2.4.5. Different input values

Something that is quite important while working on the two AD boards is to know exactly what is expected on the input of the AD converter. The old board has an extra buffer circuit installed, which shifts the input from -1V, +1V to +1.5V, 3.5V. The new board hasn't got this circuit installed, therefore the input value that should be connected to the AD board, has to be between +1.5V and +3.5V. Failure to do so, will result in the value 0 or 4095 (minimum or maximum value) when the current is not too large. If the input runs too far out of range, the chances are that one of the protection diodes will be blown.

4.2.5. Communication between AD-board and Main board

The communication between the parent board and the daughter board is completely handled by the main board. The daughter boards are only capable of putting the right values on the right pins, the transfer to the memory and determination of type of data is done by the main board using EDMA.

These routines for transferring the data is implemented in the main program, based on a few variables in the DSP/Bios, such as External Memory Interface settings and Timer

settings. During the start-up of the program, the initialization, the values that are needed for correct sampling are determined and an EDMA channel is configured which can later be opened when needed. Also during this initialization, two registries containing eight to ten bits each are written to a registry in the THS1206 chip. This is the only thing that can be set on the ADC-board. These registers contain information about what inputs should be sampled and how the data should be presented on the output pins.

When the initialization of the ADC routines is finished, the only task that has to be done to sample data is calling one function.

```
dc_rblock(&Ths1206_1, ad_buffer, AD_BLOCK_SZ, &BlockReady1206);
```
Formula 4.2.1: Function for receiving data from ADC

In this function, all basic information is given; the address on which the AD converter can be found, the buffer in which the captured data should be placed, the amount of samples that is requested and the function to call when the sampling is finished.

When this function is finished, the samples can then be found in the specific array whereby it can be used for further processing.

4.2.6. Calculation of the Sample rate

An important part of the system is the sample rate, as explained before. The settings for the sample rate are configured in the "TIMER" module of the DSP/Bios.
Because the system that is used to set the sample rate at the moment is not the most reliable system, a calculation based on the bit rate of the ID tags is given as verification.

4.2.6.1. Calculation using CPU speed

In the datasheets of the DSP board, the following formula is given to calculate the sample rate:

$$F_s = CLKSRC / 2 * PRD$$
Formula 4.2.2: Sample rate based on PRD registry

F_s : Sample Frequency
CLKSRC: Clock Source
PRD: Value of the Period Register in the TIMER module

PRD can be set during the initialization of the board, while CLKSRC is derived from the main processor, according to the following formula:

$$CLKSRC = CPU\ speed / 4$$
Formula 4.2.3: Clock source based on CPU speed

Where the CPU speed is 225 MHz in the case of the C6713 processor.

When these two formulas are combined and simplified, the following can be derived:

$$F_s = 28.125 / PRD$$

For optimal detection, the sample rate must be roughly twenty-four times as high as the bit rate of the ID-tags; this because most of the detection algorithms are optimized to work with this amount of samples per bit.
The type of ID's that are used at the moment has a sample rate of 64 kbit/sec.

$$F_s \text{ needed} = 64 * 24 = 1.536 \text{ MHz}$$

When converting this to a value that can be used in the PRD register:

$$PRD = 28.125 / 1.536 = 18.31$$

Seen on the fact that the register can only handle whole hexadecimal values, the calculated value must be truncated and converted to a hexadecimal value.
The value that must be used during the initialization of the board is;

$$PRD = 12 \text{ HEX}$$

4.2.6.2. Verification of the value using the reverse formulas

Used Period Value: 12 HEX
Clock source (CLKSRC) = CPU speed / 4 = 56.25MHz.
Sample Frequency = Clock source / 2 * Period Value = 1.5625 MHz.

4.2.6.3. Verifying the sample rate using an ID-tag

ID tags that are used are 64kbit/sec
1 bit is transferred in 1/64k sec. = 15.625 micro sec.
An average ID bit is detected using 24 samples.
15.625 micro sec / 24 samples = 651 nano sec.
Every 651 nano seconds a sample is taken, so the sample frequency is 1.536MHz.

4.3. Data Processing

Before the final set-up and software is discussed, one last remark has to be made about the construction of the software. In most of the programs that are written using the convolution theory, it is expected that there is a fixed amount of samples, or batches of samples. Keeping this in mind, certain variable parameters can be made fixed and programming is much easier.

4.3.1. Batch processing

In the problem of the ID-tags, there will be an unknown amount of samples, close to infinity. The system will start on a certain moment in time and it will end at the moment the system isn't necessary anymore. Keeping this in mind, there are only two ways of implementing the system left; using batch-calculations or using complete cyclic structures whereby the output is calculated after every sample. With batch structures it means that a certain amount of samples is captured and the output is calculated at once.

This system has two major drawbacks. The first one is that the amount of samples needed to capture a complete ID-tag is somewhere in the range of 1300 and 2000 samples. This will need a large amount of memory when it is being filtered. It is even not considered that there will be a big problem with synchronizing on the beginning of the ID-signal, before storing samples in the registers.

The second problem that will occur is the computation power that is needed on the moment a batch of samples is captured. A different subroutine will be called and all the output is calculated at once. This means there is the amount of samples per batch times the amount of instructions per sample that have to be done at once, preferably before the next sample is captured.

4.3.2. Cyclic Processing

Unlike batch processing, cyclic processing will lead to less of these problems. All the instructions that are needed to calculate one output sample are done directly at the moment the sample is captured. Although more instructions are probably needed for this process, the peak computation power needed can be much lower. As an extra, big registers are not needed; only the values that are necessary to calculate the specific output value have to be in the registers. This can mean a factor ten reduction in the memory needed.

One of the disadvantages of this system is that the bigger the amount of instructions that is needed per sample and the more complex algorithms that are used, the easier mistakes can occur while developing the system. These errors will then be very hard to trace back.

This is however not the biggest disadvantage. As the simulation software was implemented in the hardware, a different problem arose that made it necessary to look for a different approach of the problem. It was discovered that the time it takes to transfer one sample from the ADC to the main memory is already more than there is available for the complete calculation of one sample. Next to that, it was concluded that more than half of the computation time went to shifting registries needed for the digital filtering.

These two problems forced us to look for an alternative to the cyclic processing.

4.3.3. Combining both systems

A solution has been found using the best of both previous solutions. While it takes relatively long to transfer one sample from the ADC to the main memory, the time it takes to transfer the complete buffer of fourteen samples available on the ADC is only a few percent longer. This dramatically decreases the time needed per sample to transfer it to the memory.

Furthermore only one large array is chosen that contains all the input samples and a few smaller ones that are used during the filtering. The large array for capturing the input is chosen to avoid getting problems with synchronization on the beginning of one ID. The size of the buffer is so big, roughly between 20000 and 250000 samples, that there is only a small chance that an ID is only half-captured. The smaller buffers are there to avoid having to shift all the samples every time after a calculation. These are much smaller than the first buffer to save memory, roughly between 32 and 512 samples.

Using this combination of small and large buffers, an optimum can be found in memory capacity, CPU power and data transfers. By using this combination it became possible to let the detection run in real time whereby there is no shortage of computation power anymore.

4.4. Source code

The final software that is used to program the test board consists of many components. To give some more insight into the distribution of the main tasks in the software, the most important files are given here with a small explanation.

4.4.1. AD-related

dc_conf.c
This file contains most of the parameters that will be used at start-up to configure the AD-converter and the communication with it.
Some of the parameters are:

- Selection of input
- Type of communication with the main board
- Address of output

t1206_ob.c
This file contains the functions that do the actual work to get the data to and from the AD-converter. It also holds routines to configure the EDMA channels.

4.4.2. Board-related

Config1.cdb
All the settings for the hardware and the timers are set in this file. Using a graphical interface, the settings can be changed, or new components can be added. For more information about this topic, please look at paragraph 4.1.4. A few components that are added specifically for this application in this section are:

- Timer
- LOG trace
- EMIF

Config1cfg.cmd
This file contains the information that is needed for the linker, when the program is linked in text-mode. This file is generated at the moment the file "Config1.cdb" is saved. This file contains information about how the memory is built up and where the main variables should come.

4.4.3. Program-related

PulseMatchDefined.h
The values that are defined in this file are used all around the program and form the base line of the main behaviour of the program. For example, the kernel sizes for the filters are

defined here. When these are changed it will directly lead to a change in the quality of the filter and the processing time of the complete algorithm.

PulseMatch.c
The main routines to start the program are assigned here; the start-up of the program, the initialization of all filters and I/O blocks used and the actual detection of the 64-bit code.

Decoding.c
When there is enough information gathered to decode information out of it, the functions in this file are called. It holds the routine from the Glitch encoding back to 1's and 0's as well as the routines to decode the bits back to hexadecimal values.

Furthermore, this file contains the routine that is needed to calculate the CRC code, and to check if there are any errors found in the complete ID. It is used every time an ID has been detected.

SendToHost.c
Using these algorithms, the correct ID's are sent to the host where the development board is connected to. These functions are called from the CRC-check and will only be executed when the correct ID has been detected.
This part of the code can be developed later on to manage more data exchange between the host and the development board, like changing settings or transferring debug data to the host.

4.5. Final Setup

To get a clear picture of all the components that are discussed until now, and what their core functions are, a quick overview of all the components is given here. All these components together form the complete system.

The whole system starts with a signal that is broadcasted by the ID-tag (figure 4.5.1).

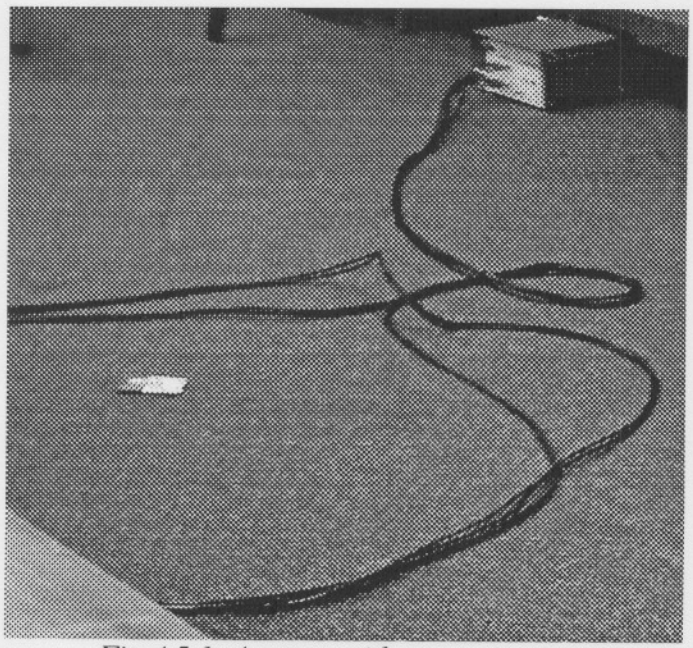

Fig 4.5.1: Antenna with tag

As the data is received by the analogue reader, the signal is tapped to the digital system. This is done quickly after the envelope detection of the incoming signal (figure 4.5.2). Using a small buffer scheme, the signal from the analogue reader is converted to a signal between +1.5 and +3.5 Volt.

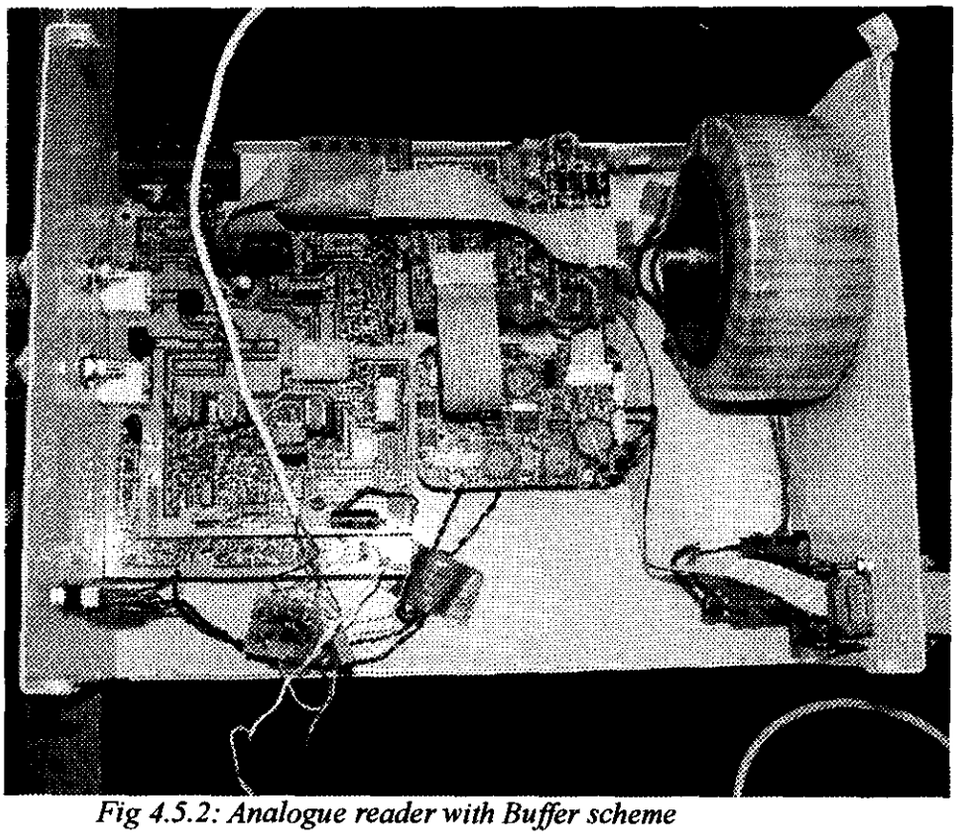

Fig 4.5.2: Analogue reader with Buffer scheme

This buffered signal is then captured by the analogue to digital converter.

The TMS processor then performs the calculations necessary for the digital filters and the detection scheme (figure 4.5.3).

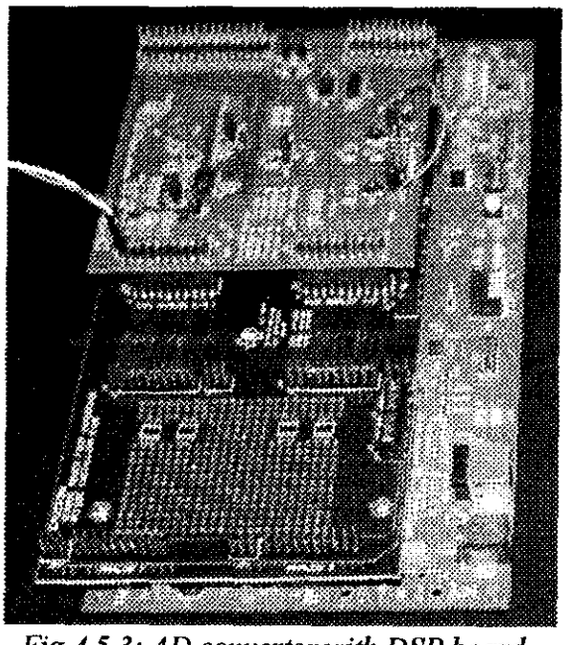

Fig 4.5.3: AD converter with DSP board

Using a USB connection, the calculated results are sent to a host computer. This can show the results on the screen, using Code composer, or in this case an interface specifically developed for this application. This interface is shown in figure 4.5.4.

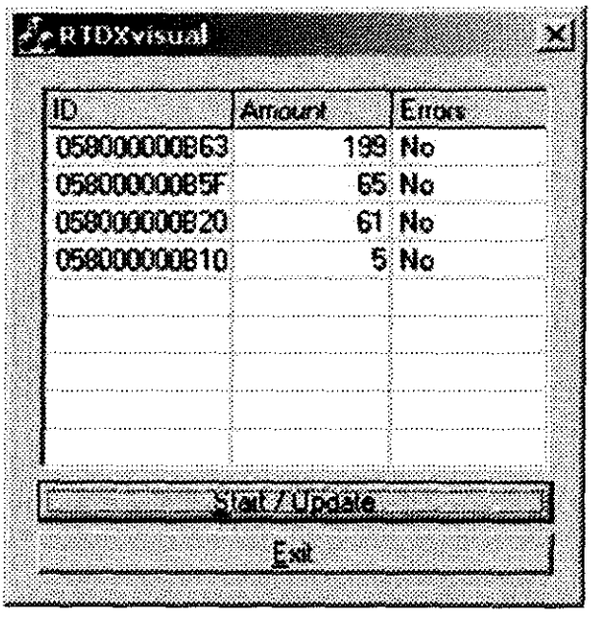

Fig 4.5.4: Application for showing final results

5. *Testing and Results*

This chapter is divided into two main parts. The first part consists of the testing of the simulation software, and second part consists of two tests that were done on the final prototype.

The reason for choosing this separation is because the simulation and the hardware versions are roughly based on two different approaches. The simulation is a closer match to the theoretical limit of the algorithm, while the hardware product is aimed more on a product that simply shows that the task can be run on real signals without worrying about the performance of the system. In the final version of the hardware prototype, both areas will be matched closer to each other, forming the best solution possible.

The first part, which deals with the performance, is divided into a number of separate tests. Firstly, a test is shown that was done at the beginning stages of the development, showing the power of the filtering. Thereafter, a few thorough tests will be shown regarding the software generated signals, and the detection quality of the simulation software. Finally, a test will be done that compares the analogue system with the digital simulation system, using actual data signals.

As already discussed, the tests in the hardware implementation are aimed more on the actual working of the system and less on the quality, seen on the fact that the quality already has been proven during the tests on the simulation system.

The first test will give some insight into the signals that are captured by the AD converter and how they are transferred to the processor. This is to give insight into the quality and performance of this conversion system. The second test is carried out to demonstrate that the system is actually working. With this last test it can be proven that the aim to implement the system into hardware was achieved.

5.1. Testing of Filters

As quickly mentioned in the introduction of this chapter, the first test that will be discussed has been done in the earlier stages of development of the simulation software. This test was done to see what the quality of the developed filters would be and if they would be able to do their job.

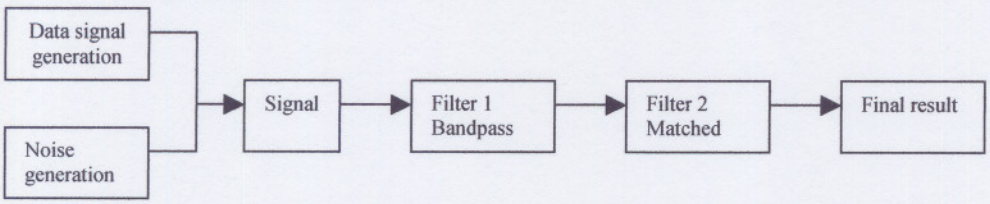

Fig 5.1.1: Block scheme of Signal generation and detection

The goal is to filter out and detect a data signal that is broadcasted between 6.5MHz and 7.5MHz, which is buried in two other frequencies.

5.1.1. Data Signal Generation

The input signal is generated with Micro-Cap, using the scheme shown in figure 5.1.1.

The input signal V4 produces spikes every few nano seconds that are converted to 7MHz waves using the capacitor and the coil. The extra resistor is built in to simulate the Q-factor of the coil. These two signals can be found in figure 5.1.2.

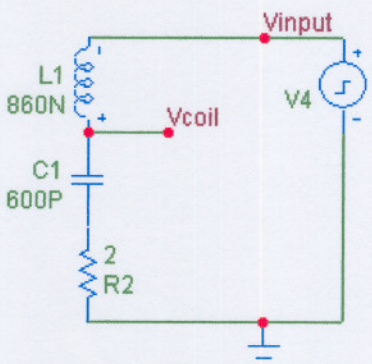

Fig 5.1.2: Simulation of the broadcasting coil

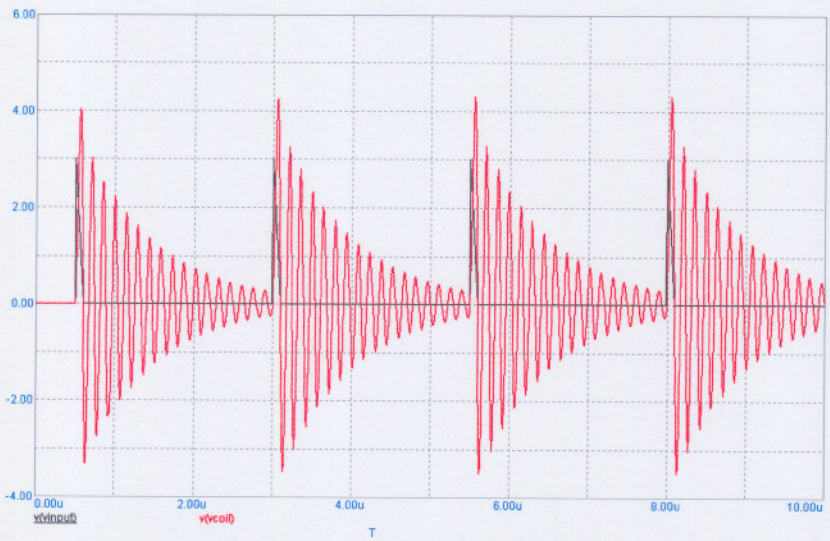

Fig 5.1.3: the signals from V_{input} and V_{coil}.

5.1.2. Noise Generation

This wave signal is then sent to an amplifier that works as a buffer. In the real tags, the signals are sent through the air. Using this scheme, a similar effect can be achieved. To this signal, two other sine waves of 6MHz and 8MHz are added, which are generated by V6 and V8 in the scheme shown in figure 5.1.3. These two wave signals don't carry any interesting information and are only added to disturb the data signal.

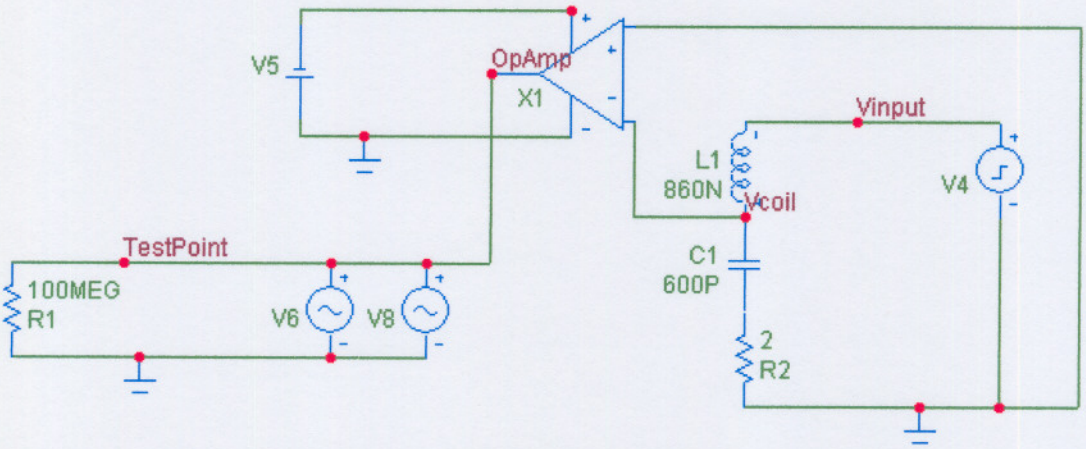

Fig 5.1.4: Simulation scheme for generating the waves

The strength of the signals is: 6 MHz : 8 MHz : 7 MHz -> 4:4:1. In other words, both noise signals are four times as strong as the data signal. In this case, the strength of the signal is measured by the maximum amplitude of the signals. In the case of the data signals, that is on the first wave of every pulse, therefore the actual energy in the data signal is even smaller than the comparison given above. (Note: the values that can be found in the graph above, the signals of V_{input} and V_{coil}, are not applicable on this calculation because these are taken before the amplifier)

Then using a sample rate of 100 MHz (100.000.000 samples/sec), the output is generated.
On the right is a graph of this output function, with a total of 1000 samples (10 nano seconds). In figure 5.1.4, you can find the 6 MHz and the 8 MHz waves, but the 7 MHz data signal cannot be found. It can be said that the data is "buried" in the noise.

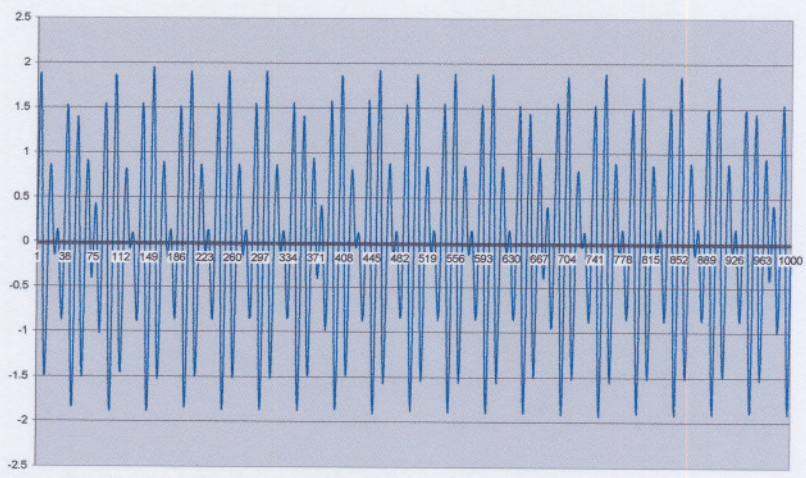

Fig 5.1.5: Data signal with noise

In this figure it is possible to see the pulses of the five ID numbers in the original signal, but they are small in comparison to the transient at the start of the signal. The transient is due to the start-up characteristics of the filter and is ignored in the detection of the ID bits.

By cutting off the transient at the start of the signal, the pulses of the five ID numbers in the input signal are shown in figure 5.2.3.

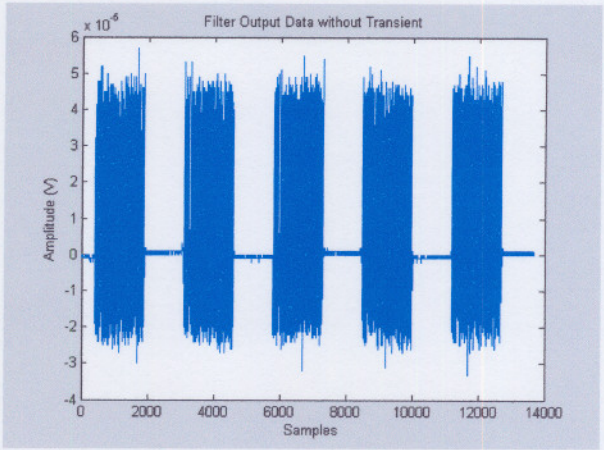

Figure 5.2.3: Windowed Sync Filter Output Data without Transient.

Detection:
In an analysis, 10 different input signals, each with 1000 ID's, were used. For each of the signals, the ID's were different and generated randomly. The resulting detection ratios are shown in table 5.2.1.

Signal	Detection ratio
1	98%
2	97%
3	98%
4	98%
5	97%
6	97%
7	97%
8	98%
9	98%
10	98%

Table 5.2.1: Results of the first test

Noisy Detection
To evaluate the operation of the implemented detection algorithm under various signal to noise ratios, the same 10 input signals, each with 1000 IDs, as in the previous section were used, but in this situation white Gaussian noise was added to the signals. An example is shown in figure 5.2.4. The blue (dark) signal is the original signal and the red (light) signal is the signal with noise added.

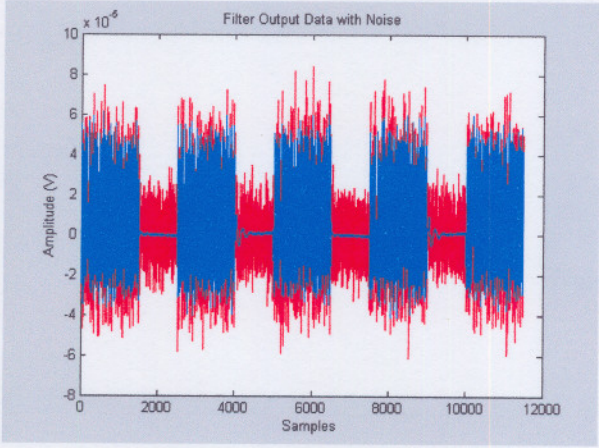

Figure 5.2.4: Noisy Input Data.

Table 5.2.2 summarizes the results of the detection of the IDs in the 10 signals under different signal to noise ratios:

SNR	Infinite	15 dB	10 dB	8 dB	6 dB	4 dB
L1	98	96	92	89	84	74
L2	97	95	95	93	87	76
L3	98	96	91	88	83	74
L4	98	98	96	93	86	75
L5	97	96	94	91	86	74
L6	97	97	96	93	89	79
L7	97	95	90	85	77	66
L8	98	98	95	93	88	77
L9	98	95	90	85	78	65
L10	98	97	94	91	87	75

Table 5.2.2: Detection percentages in noisy situation

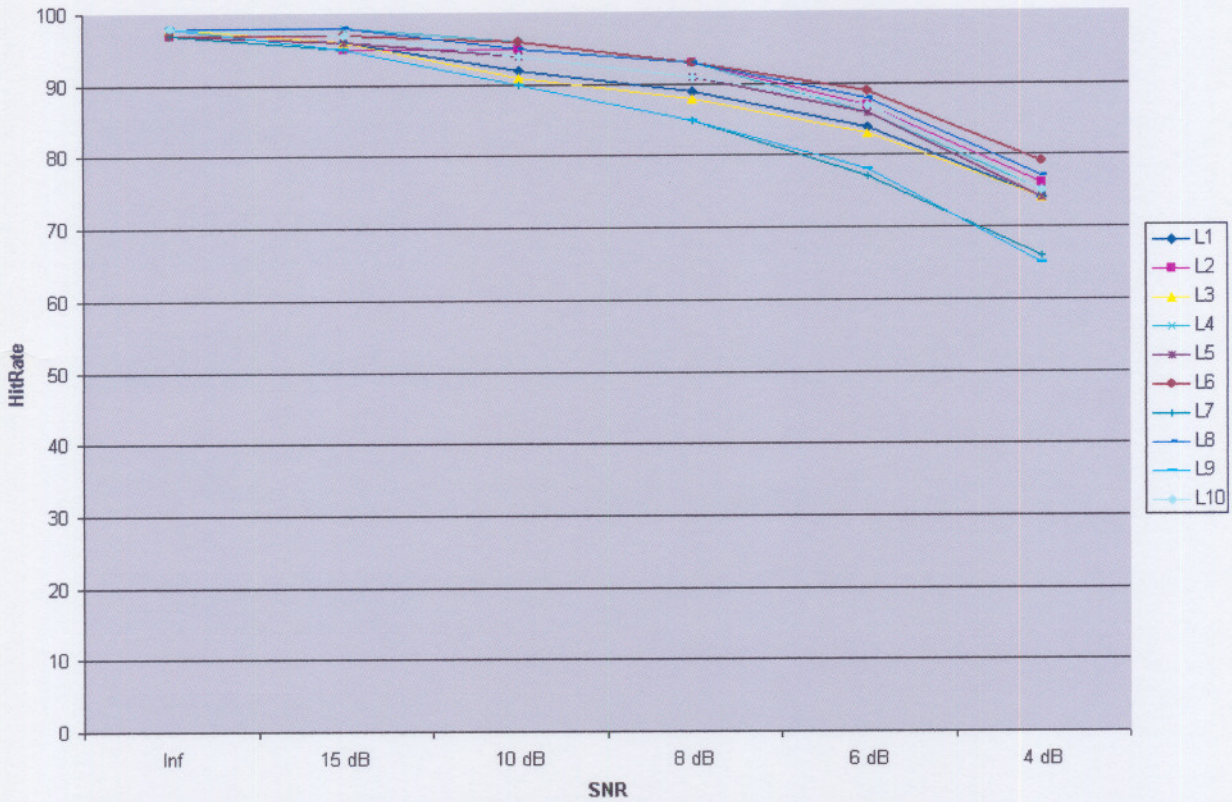

Figure 5.2.5: Detection percentages in noisy situation.

From table 5.2.2 and figure 5.2.5 it can be seen that the detection algorithm performs quite well, even in noisy situations where the signal to noise ratios are down to as low as 4 dB.

5.2.2. ID's using 7 MHz waves

An example of an input signal with 5 IDs is shown in figures 5.2.6 and 5.2.7.

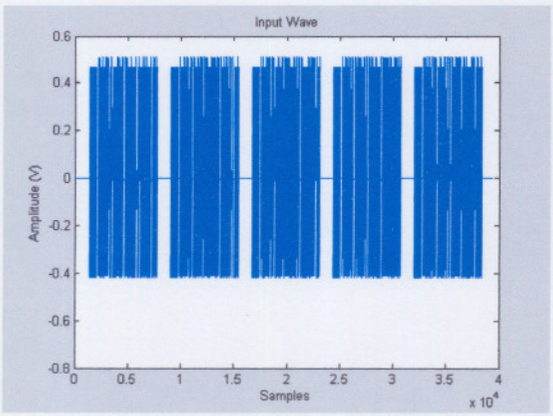

Figure 5.2.6: Five 500 mV ID's in incoming signal

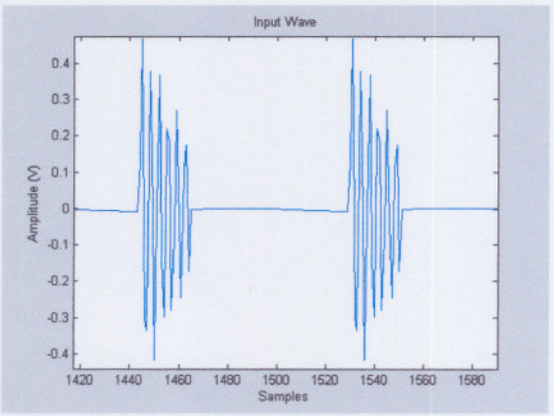

Figure 5.2.7: Zoomed view of five ID's in incoming signal

In an analysis, 5 different input signals, each with 1000 ID's, were used. For each of the signals, the ID's were different and generated randomly. The resulting detection ratios were as shown in table 5.2.3:

Signal	**Detection ratio**
1	100%
2	100%
3	100%
4	100%
5	100%

Table 5.2.3: Results of the first analysis

Noisy Detection:

To evaluate the operation of the implemented detection algorithm under various signal to noise ratios, the same 5 input signals, each with 1000 ID's, as in the previous section were used, but in this situation white Gaussian noise was added to the signals. An example is shown in figure 5.2.8; the blue (dark) signal is the original signal and the red (light) signal is the signal with noise added.

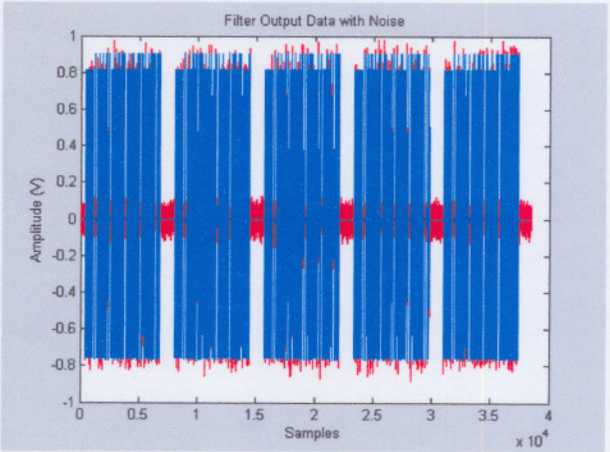

Figure 5.2.8: Noisy input data

Table 5.2.4 summarizes the results of the detection of the IDs in the five signals under different signal to noise ratios:

SNR	Infinite	15 dB	10 dB	8 dB	6 dB	4 dB
L1	100	99	97	92	80	61
L2	100	99	97	92	80	61
L3	100	99	97	92	80	61
L4	100	99	97	92	80	61
L5	100	99	97	92	80	61

Table 5.2.4: Detection percentages in noisy situation

From the table it can be seen that the detection algorithm performs quite well, even in noisy situations down to signal to noise ratios as low as 4 dB. The only strange thing is that all the values are exactly the same for the different ID series. Because of this unexpected behaviour, an extra test is done.

In this test, a certain amount of noise is added to the signal at the moment the ID's are generated. If there would be a mistake in the program that makes it possible to have exactly the same results in different ranges, these results also have to be exactly the same. The amount of noise added in this test, is not calculated using a specific signal to noise ratio, but instead, it is done in percentage of the original signal. To get an idea of the amount of noise used in this test, figures 5.2.9 and 5.2.10 are two of the input signals that are used in the tests.

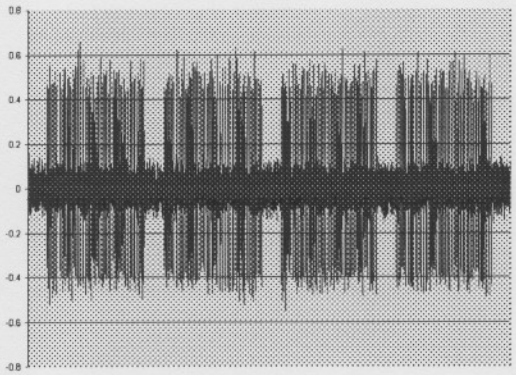

Figure 5.2.9: Input signal with 20 percent white Gaussian noise

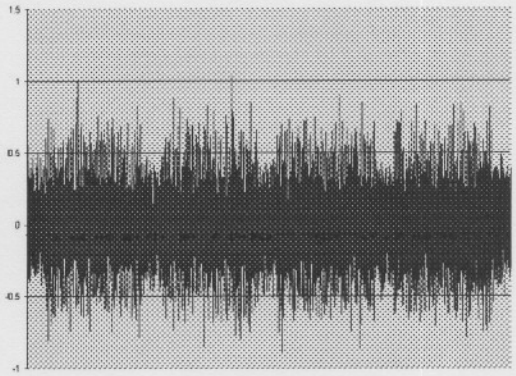

Figure 5.2.10: Input signal with 70 percent white Gaussian noise

These tests are done over an amount of 100 ID's per test. The ID's of one data series are again exactly the same in all the different noise levels. Please note that because of the different approach of adding the noise, the noise levels do not have to be exactly the same in both situations. Due to this, it can happen that although two measurements have the same amount of noise in the earlier test and this test, the results vary.

Noise Level	L1	L2	L3	L4	L5
0 %	100	100	100	100	100
10%	100	100	100	99	100
15%	98	96	99	96	100
20%	88	88	95	87	91
30%	62	66	62	58	69
40%	50	48	51	41	51
50%	38	31	34	35	29
60%	19	21	21	13	11
70%	9	11	4	7	3

Table 5.2.5: Hit rates with different levels of noise

In table 5.2.5 it can be seen that in some cases the outputs are indeed exactly the same, but that they are overall different. It can now be expected that the measurements of the last two tests are reliable enough.

In figure 5.2.11, the values of the last test are plotted. A nice addition to these results is that although the noise level is raised to a maximum of seventy percent, a level in which it becomes very difficult to even see an ID, there are still a few ID's that are detected without any errors.

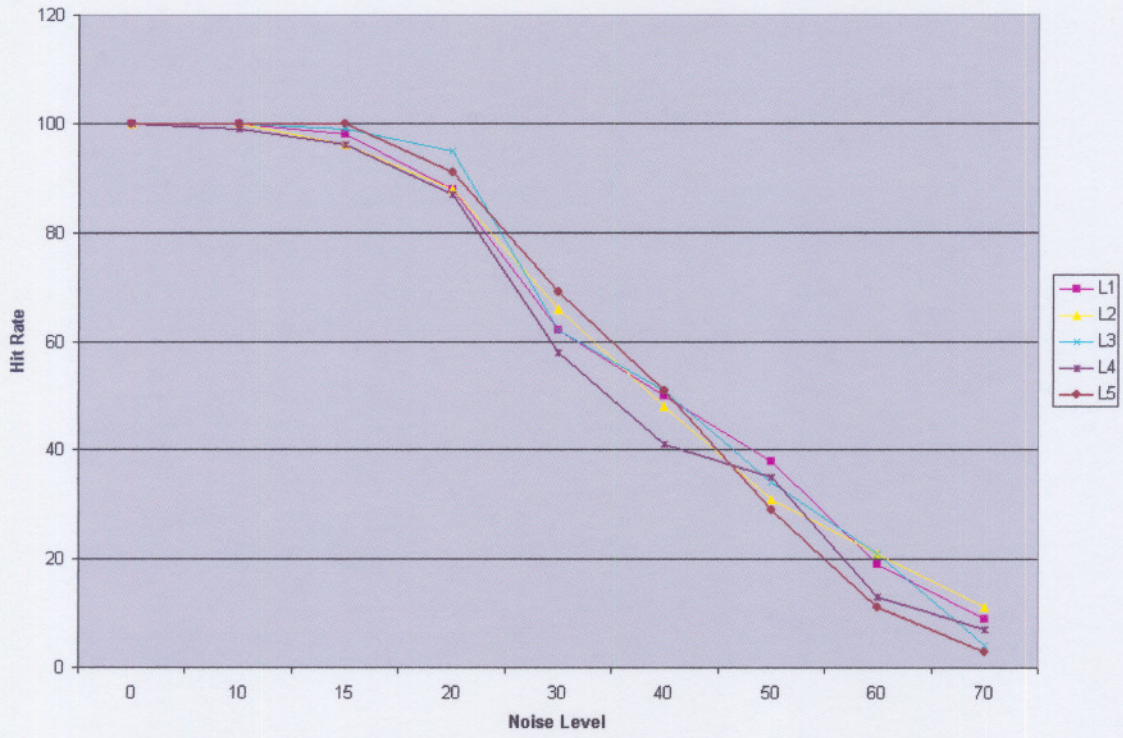

Figure 5.2.11: Results of an additional test with high noise levels

5.3. Testing the algorithms with actual signals

When the main software was tested and proved to be capable of doing its job on generated signals, it became more and more of a request to see if it would also be possible to use this detection system on actual data signals. Therefore, a two day test period was arranged at iPico to capture data of signals that are known to give problems and signals that are expected to be detected correctly by the analogue reader.

As a result of this test period, a large number of sample files were available with data signals for the Dual frequency as well as for the UHF system. The samples of the UHF system were handled by Christo Vorster, who has joined the project to look more specifically at the UHF problems. The dual frequency samples are discussed here.

5.3.1 Testing Parameters

Below there are six captured signals from all types of environments with specific problems. Based on the fact that it is only guessing if this specific ID would have been detected by the analogue reader, because of the random effects and the quickly changing testing environment, only a prediction of the performance of the analogue reader is given. This has however been proven to be a close match of the actual performance. To calculate the output for the digital reader, a file has been compiled where one hundred of the same ID's are put after each other, which is then used as input for the simulation software.

The term "close by" means that the tags are on a distance of a few centimetres from the reader antenna, the term "far away" means that the tag is situated on the edge of the power field. Would the tag have been shifted a few more centimetres from the reader, it would not broadcast a complete ID signal anymore. For all the measurements the input and the output of the filter are drawn. For the input, the complete ID is given as a reference for the quality of the incoming signal. For the output, a few bits of the ID are given as a reference for the strength of the output signal.

5.3.2. Tag with little noise, close by

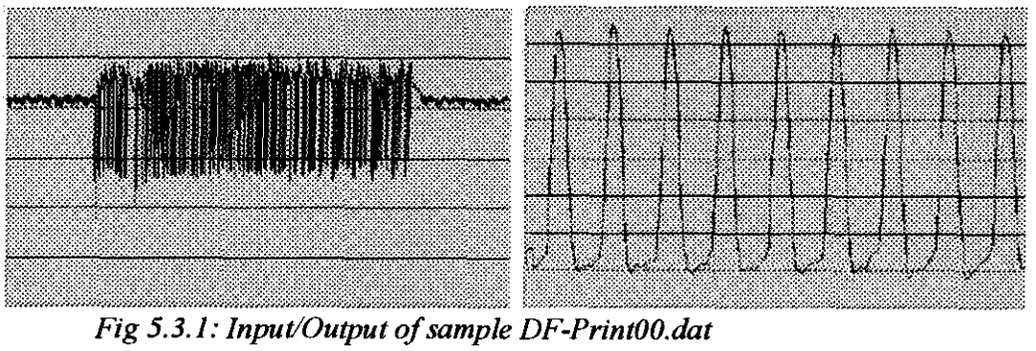

Fig 5.3.1: Input/Output of sample DF-Print00.dat

Analogue expectation: Good detection.
Digital results: ID's detected: 100%
ID's detected without errors: 99%

As this is the most basic form of an ID signal, the results should be, and are in both cases, correct.

5.3.3. Tag with little noise, far away

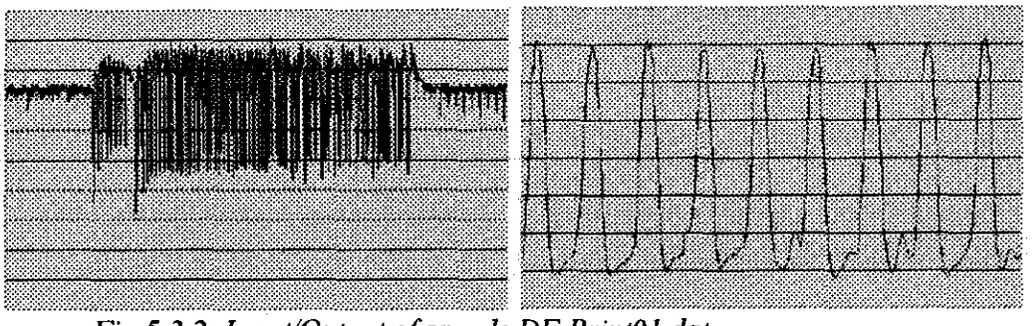

Fig 5.3.2: Input/Output of sample DF-Print01.dat

Analogue expectation: Weaker detection
Digital detection: ID's detected: 100%
ID's detected without errors: 1%

The reason for the unsatisfactory digital detection can probably be found in the unexpected strength drop of the signal during the start bits. Together with this strength-drop, the bit speed of the ID signal also drops. Strangely, this strength and bit speed drop only occurs during the start bits, and very little during the rest of the ID signal. There is a difference of about 12-15% in the bit speed between the last few start bits and the first few ID bits.

In the detection algorithm the start bits are used for synchronizing the data stream and determining the bit speed. As an effect of this extreme bit speed change, which is not expected according to the specifications from iPico, the synchronisation of the signal fails and therefore the complete ID gets discarded. When the synchronisation of the ID-signal is corrected manually after the start bits, ID detection is close to 100%, showing that the rest of the ID detection proceeded without any problem.

5.3.4. Tag close by, with noise from electric drill

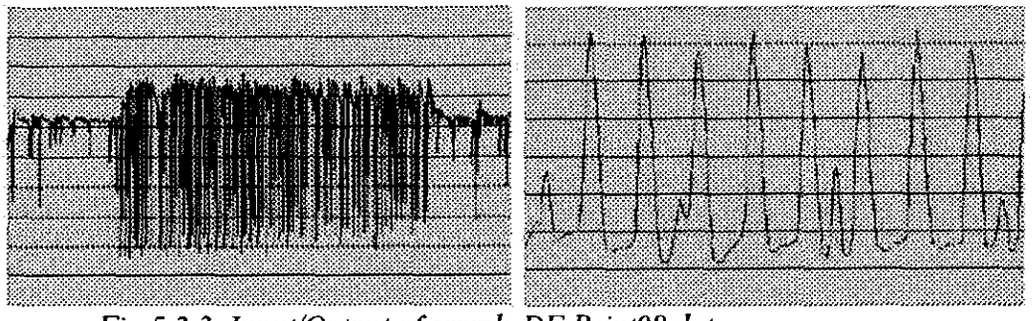

Fig 5.3.3: Input/Output of sample DF-Print08.dat

Analogue expectation: Weak to poor detection
Digital results: ID's detected: 100%
ID's detected without errors: 99%

Noise spikes up to the same strength as the ID signal are reduced to a maximum of 50% of the ID signal in the output signal (see right figure). Therefore, they are minimised too small to interfere with the main part of the ID detection. If a spike turns up on the wrong place at the wrong time however, it can still have a negative influence on the ID detection.

5.3.5. Tag far away, with noise from electric drill

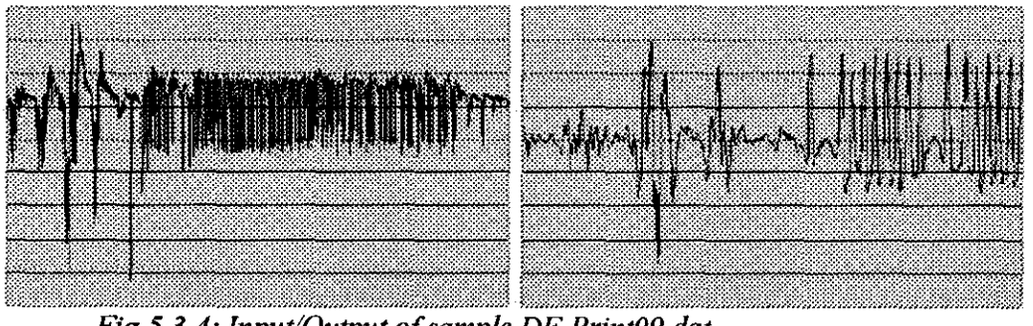

Fig 5.3.4: Input/Output of sample DF-Print09.dat

Analogue expectation: Poor to no detection
Digital results: ID's detected: 100%
ID's detected without errors: 98%

As mentioned before, when a spike turns up at the wrong place at the wrong time, disturbance is still possible. In this case, a spike turned up just before the first start bits. Fortunately, using strong digital filtering, it is still possible to detect the ID's without errors. It has to be admitted that this is on the edge of the possible.

5.3.6. Tag far away, with noise from switching power supply

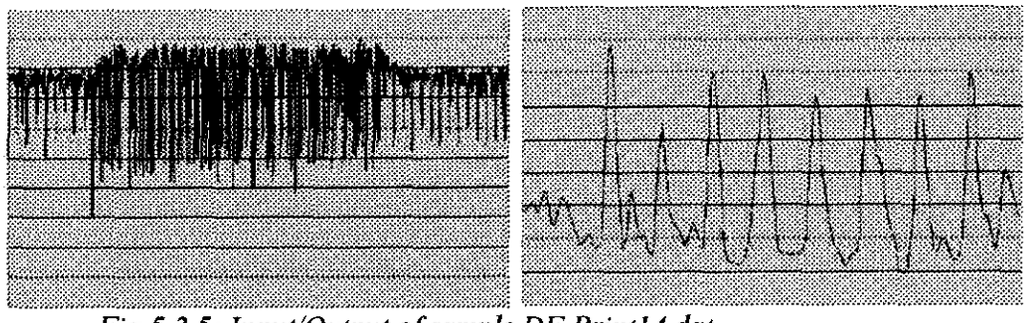

Fig 5.3.5: Input/Output of sample DF-Print14.dat

Analogue expectation: Poor to no detection
Digital results: ID's detected: 100%
ID's detected without errors: 0%

This is probably one of the worst scenarios that can be found in Dual Frequency reader systems; a weak ID signal with huge noise spikes. Both systems are very likely to fail in these situations. But now that these extreme situations are identified, more attention can be put onto this type of signals at a later stage, which can increase the digital system's strength to such an extent that it might enable the system to cope with this type of noise.

5.3.7. Collision between strong and weak tag

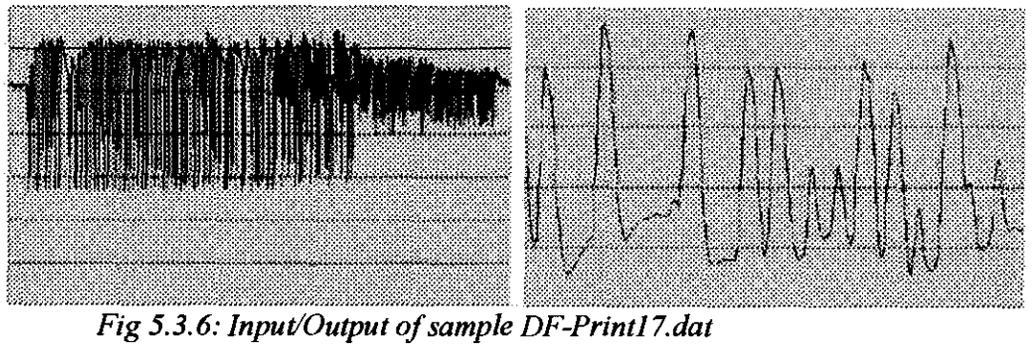

Fig 5.3.6: Input/Output of sample DF-Print17.dat

Analogue expectation: Weak to no detection
Digital results: ID's detected: 100%
ID's detected without errors: 99%

Note: Only the strong ID's are detected, the weak ID's are determined as noise.

As understood from iPico, the analogue reader experienced major problems dealing with colliding tag signals; in most cases both ID signals are discarded. As the results from the digital reader shows, some of these ID's can actually still be detected. When these ID

signals can still be used, it will lead to a major improvement on the statistical time needed to detect more tags in a reader beam.

5.4. Checking the input of the Prototype

To see if the incoming information is roughly the same as can be expected with the simulation programs, some examples of data that is captured using the AD converter onboard is given below. For every sample there is some explanation given about what exactly is captured and how it relates to the simulation expectations.

Detection results are not given here, because those results would roughly be the same as the results generated with the last test of the simulation software. Therefore, this test is only done to demonstrate that the board can actually capture data with the same quality as expected from the simulations.

5.4.1. Signal near the antenna

One of the objectives of the entire project was to see if it would be possible to detect signals in a range further from the antenna than the analogue system was capable of, and also in environments with more noise.

Because of this aim, very strong signals with little noise are not so interesting; these should be detected without problems. Although this is indeed true, there is one obstacle that deserves some attention; the input domain of the AD converter.

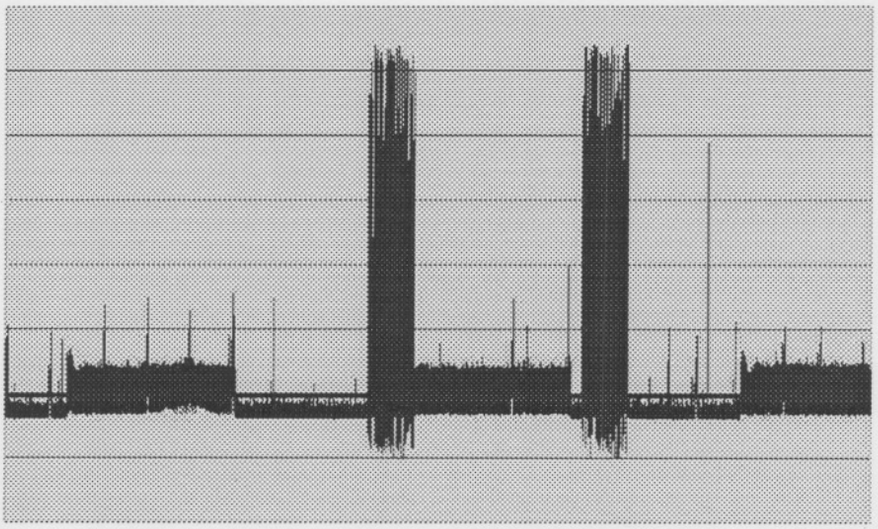

Fig 5.4.1: Two ID signals broadcasted near the antenna

As already explained in paragraph 3.1.1, the quantization of the signals when they are sampled can form a strong disturbing noise if the size of the signals per bit is made too large. To avoid this, there is chosen to let the incoming signal peak outside of the input range. In some cases, the AD converter might recognise this as the highest value possible, but, in other cases; the AD converter will output a value at the bottom of the range. This is also the case with this converter.

In figure 5.4.1, there is an input sample of an antenna signal with two ID broadcasts. As it can be seen, most of the signal is clipped at the bottom, as it has been done by the buffer circuit between the AD converter and the analogue reader. (See paragraph 4.5) It seems however, as if the ID signal is not clipped. These are some of the values that are shifted to the bottom of the AD converter's range.

In this stage of the development, the influence of this problem is not too big, and therefore not investigated. It has to be kept in mind that if the domain of the AD converter would be calibrated on the largest signal, the quality of smaller signals will dramatically decrease. When a final setup is made, this problem can be handled quite easily with a small addition to the buffer circuit, or even better; by implementing a different AD converter configuration that does not give this problem.

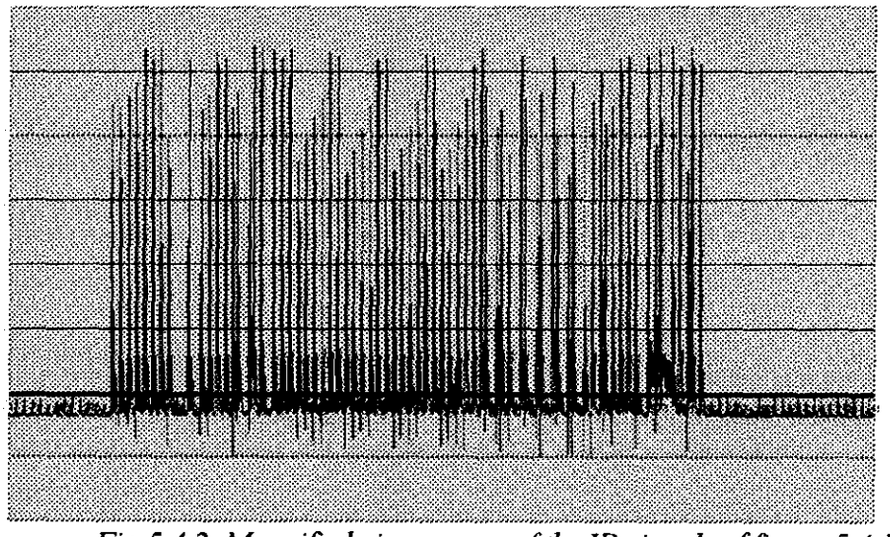

Fig 5.4.2: Magnified view on one of the ID signals of figure 5.4.1

In the magnified figure, figure 5.4.2, it can be seen that this irregularity also affects some areas of the top of the signal. This is however no problem, because the signal/noise ratio is so big, some loss of quality will not cause any difficulties.

5.4.2. Signal far from the antenna

The more important input signal is the one where the signal to noise ratio is very weak. An example of this is illustrated in figures 5.4.3 and 5.4.4. This input signal was taken when a tag was on the edge of the power field; a few centimetres more and the ID signal would not have been broadcasted anymore.

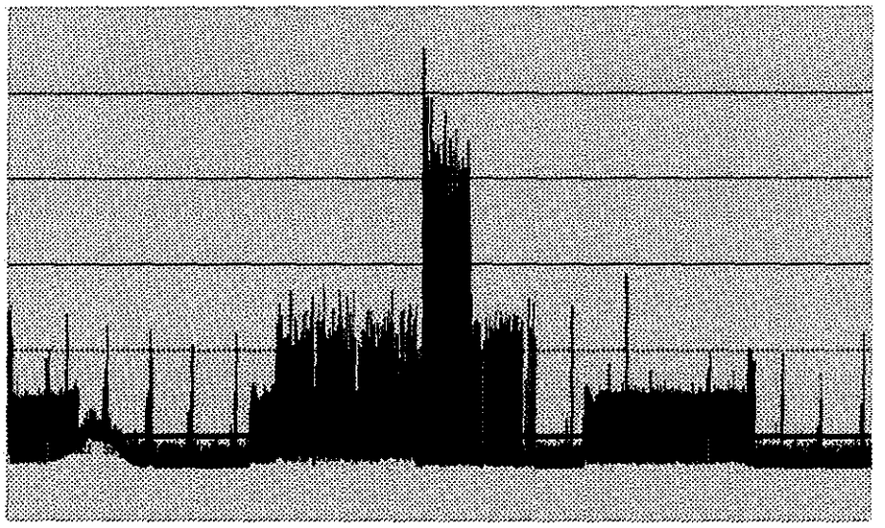

Fig 5.4.3: ID tag at the edge of the power field

As it can be seen, the signal to noise ratio in this example even falls to 3 dB. This can be described as one of the worst circumstances in which a tag has to be detected. Please also note that the problem discussed in the previous paragraph can not be found on this signal anymore.

Although there is a very big chance that the analogue reader will not detect this ID, the digital reader will have a good chance of finding it, even without errors. This is, as long as the bit rates of the tag remain within the specifications, as discussed in paragraph 5.3.3.

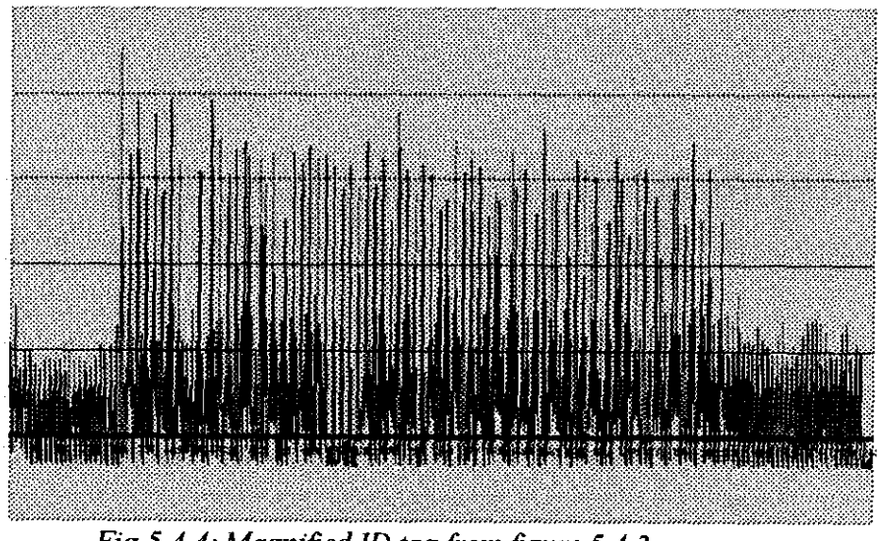

Fig 5.4.4: Magnified ID tag from figure 5.4.3

5.5. Testing of the Prototype

As a final test, a simple measurement is done to check if the prototype is capable of detecting any tags and showing these on a host computer. This is done to calculate the difference in performance between the analogue and the digital system.

The values given here were taken on Tuesday 22nd of June 2004. Although these values will not be up-to-date, it will give an impression of the power of the system and the method in which it is calculated.

5.5.1. Test Comparison

Every time a batch of samples is calculated, the program starts sampling again and sends out a message to the host computer. This batch has at the time of the measurement a size of 40000 samples.

The sample rate in this example is 1.5625 MHz, what means that every 640 nano seconds a sample is taken. This gives a time of 25.6 milli seconds per batch. (40000 samples times 640 nano seconds) In other words, every time a batch of samples is captured, the system listens to the antennas for 25.6 milli seconds.

When the program was run for just below two minutes (117 seconds), 506 batches of samples are captured. This means that the complete time that the system listened to the antenna is 506 batches times 25.6 milli seconds; 12.95 seconds.

The time the program ran was 117 seconds; the captured time is approximately 13 seconds. So, the amount of times the program is too slow can be calculates as: 117 / 12.95 = 9.03 times. This value can now give an indication of how long the analogue system should listen to the antenna to be able to make a good comparison.

5.5.2. Test Set-up

For this set-up, three measurements are done;

1. five tags on the floor (within the antenna coil)
2. five tags on a height of 40 centimetres above the antenna
3. five tags on a height of 60 centimetres above the antenna (edge of the field)

All three measurements were done with five tags, and the position of the tags was not changed during a test.

5.5.3. Test Results

Firstly, test 1 is done, the five ID-tags have been place within the antenna, and the signal strength is as strong as possible. The analogue system picks up all five tags and roughly in the same quantity.

The digital system does not perform so well in this test; although all tags were identified at least once, the amounts are quite different. This can be explained when the problem from paragraph 5.4.1 is kept in mind; some of the signals are too big and will form a distortion that makes it impossible to detect the ID correctly.

Tables 5.5.1 and 5.5.2 give the results of all three tests and the amount of times unique ID-tags are captured.

Analogue Detection		
Test 1	**Test 2**	**Test 3**
12	10	6
13	12	7
12		
14		
13		

Table 5.5.1: Test Results of the Analogue System

The second test gives a better performance for the digital system. While the analogue system struggles a bit to detect the ID's, the digital system is able to detect one ID more than the analogue one. Furthermore, the total amount of tags is much more than the analogue system.

Digital Detection		
Test 1	**Test 2**	**Test 3**
14	36	12
6	25	13
3	1	
4		
1		

Table 5.5.2: Test Results of the Digital System

The last test, test 3, shows that both systems now have problems detecting signals. Still, the digital system seems to be able to detect more tags.

	Test 1	**Test 2**	**Test 3**
Analogue	60	25	13
Digital	28 (53%)	62 (248%)	25 (192%)

Table 5.5.3: Amount of times an ID-tag is detected correctly

When all three tests are compared, it is clear that the digital system achieves a higher detection rate, especially when it is kept in mind that the detection rate for the digital system in the first test can be pushed higher (see paragraph 5.4).

It is now shown that all the goals of the project have been achieved.

6. Conclusions

After evaluating the different phases of the development and the results of different types of input signals, it can be said that the aims as they are stated in chapter one have been achieved, although with some remarks.

Using the simulation software, it has been shown that it is possible to use digital techniques for the detection of wireless identification signals and that the algorithms used can compete with the quality of the analogue reader. After that, it has also been shown that these algorithms can be successfully implemented in hardware whereby it can produce a complete system capable of performing the detection.

It is important to know that this system can run on present-day technology. With still further improving DSP technology, it will soon be possible to implement the algorithms on an even faster or smaller device whereby the possibilities and the costs-effectiveness will improve. Next to that, also the problems that are still present in the prototype can be fixed.

Concluding, the project is successful and the prototype developed can form a strong base for further development with the definite potential of manufacturing a commercial product.

7. References

Steven W. Smith, "The Scientist and Engineer's Guide to Digital Signal Processing"
California Technical Publishing, 1997, http://www.dspguide.com

"TMS320C6713 DSK, Technical Reference"
Spectrum Digital, 2003

Texas Instruments
"TMS320C6000 Peripherals Reference Guide"
Literature number: SPRU190D
Texas Instruments, February 2001

"12-Bit, 4 analogue input, 6MSPS, Simultaneous sampling analogue-to-digital converters"
Literature number: SLAS217H
Texas Instruments, July 2003

"5-6K Interface board EVM, Users guide"
Literature number: SLAU104A
Texas Instruments, May 2004

"THS1206, THS12082, THS10064, THS10082 Evaluation module, Users Guide"
Literature number: SLAU042B
Texas Instruments, January 2001

Heinz-Peter Beckemeyer, "Designing with the THS1206 High-speed data converter"
Literature number: SLAA094
Texas Instruments, April 2000

"Modular THS1206EVM, Users Guide"
Literature number: SLVU098
Texas Instruments, December 2003

iPico
Marius van Dyk, "DF Single Channel Long Range Reader, Product specification"
iPico Identification, December 2002

Luther Erasmus, "Dual Frequency Tag Specification"
iPico Identification, July 2002

"EM4022, Multi frequency contact less identification device"
EM microelectronic – Marin SA, 2002

"EM4322, Read-only dual frequency identification device"
EM microelectronics – Marin SA, 2002

North-West University
Prof W.C. Venter, "Digital RF-ID Reader, Phase I"
North-West University, 2003

Prof W.C. Venter, "Digital RF-ID Reader, Phase II"
North-West University, 2003

Prof W.C. Venter, "Digital RF-ID Reader, Phase III"
North-West University, 2003

M.C. van de Pol, "Digital reader software"
North-West University – Saxion Hogeschool Enschede, 2004

Special thanks to:
Google, http://www.google.com