Chapter 4

Smart Antenna Test-bed: Design and Implementation

4.1 Introduction

As there are many approaches to smart antenna implementations, it is necessary to analyze the possibilities before actually building the system. For an adaptive array there are certain aspects that need to be covered by the architecture. In general for every antenna in the array an RF front-end is necessary, performing up and down conversion and filtering. At high frequencies measurements are more subjected to feedback, noise and distortion. The length of the cables, the shape of connectors and the PCB layout are all contributing to the quality of the transceiver. Therefore the RF area can be considered a special field of work, and if it is part of a thesis a high level of expertise is required to develop a system.

A way to avoid difficulties in this area would be the use of common of the shelf (COTS) transceivers. An implementation of a front-end receiver from COTS RF components appeared to be impossible; as most of the COTS components are designed for specific operation in existing communication standards or they are expensive professional products. Another way to avoid part of the design in the RF area is to use evaluation modules from existing RF integrated circuits. The evaluation boards contain a setup with extra components and PCB board, to evaluate the product easily. The evaluation boards can also function as a reference design, for an eventual custom design.
The RF part of a wireless communication system generally provides a conversion between the radio or microwave frequency and baseband or intermediate frequencies (IF). This is called up and downconversion, which is respectively related to up and down shifting in the frequency spectrum of the signal that needs to be received or transmitted. In the past years the RF technology has improved a lot, due to the growth of the wireless industry. Many chip manufacturers develop integrated circuits for RF applications covering all analog and digital communication standards in the world. These products range from fully integrated circuits for digital standards, to flexible building blocks for custom RF products. RF designing is now a special field of expertise, where a practical approach is the common way to work. It is leaded by the mobile industry and the ICs are more and more fully integrated, providing a complete receiver, transmitter or transceiver to be implemented in a mobile communication device. The mobile communication standards DECT, GSM, IS-95, D-APMS, are just a few from the pile of mobile communication technologies that exist all over the world.

In principle, a smart antenna can be formed to work with one of these technologies. Certain systems will be easier to implement and certain radio architectures are more suitable then others. From the theory, it was seen that for the used algorithms complex valued signals are necessary. The receiver not only receives and recovers the in phase and quadrature signals, but also preserves phase and amplitude information of the RF signal. Figure 4.1 shows the representation of the receiver.

![Figure 4.1 beam-former with radio receivers](image-url)
This chapter considers the different architectures and an evaluation is given supported by practical as well as mathematical foundations. Furthermore a proposal is made for an RF front-end for a receiver with simple components. This proposal can be used in the future as a reference for the actual design and implementation. The chapter discusses the implementation of the test-bed. First the general system is given. After the system overview, the hardware will be discussed as well as its performance. In the last part the software design on task level is treated.

4.2 Receiver Fundamentals

One of the most common building blocks of a receiver is the so-called mixer. The mixer can be seen as a multiplier, which has two inputs, and one output. Generally the two inputs of the mixer are the received signal and a locally generated signal called Local Oscillator (LO). The basic function of a mixer is a translation of an input signal to a different frequency. The basic math that describes this function is formed by the trigonometric relations:

\[
\cos(\omega_1 t + \phi) \cos(\omega_2 t) = \frac{1}{2} \left[ \cos((\omega_1 - \omega_2) t + \phi) + \cos((\omega_1 + \omega_2) t + \phi) \right] \quad (4.1)
\]

\[
\cos(\omega_1 t + \phi) \sin(\omega_2 t) = \frac{1}{2} \left[ \sin((\omega_1 - \omega_2) t + \phi) + \sin((\omega_1 + \omega_2) t + \phi) \right] \quad (4.2)
\]

The angular frequencies are represented by \(\omega_1\) and \(\omega_2\) and the phase difference by \(\phi\). The function of the mixer is the translation of both frequencies to sum and difference frequencies. In receiver or transmitter architectures, only one of the two output frequencies is interesting, and therefore the mixer will be combined with an output filter, filtering either the sum or difference frequency.
To explain the basics of the receiver, an architecture is introduced, which is mathematically the simplest form of a receiver, known as the direct-conversion receiver. The concept of the direct-conversion receiver is visible in Figure 4.2.

![Direct-conversion receiver diagram]

Figure 4.2 Direct-conversion receiver

The receiver consists of an antenna, a low noise amplifier, a mixer stage and a low pass filter. The received signal at the antenna is amplified and then injected to the mixer stage. The mixer stage usually consists of two mixers, which demodulates the received signal. The LO is offered to both mixers with a 90 degree phase shift for the demodulation of the in phase and quadrature component.

Consider the received signal \( r(t) \) a quadrature modulated signal represented by:

\[
r(t) = m_i(t) \cos(2\pi f_c t + \phi) + m_q(t) \sin(2\pi f_c t + \phi)
\]

(4.3)

Where, \( m_i(t) \) and \( m_q(t) \) represent the in phase and quadrature message components respectively. The received message is multiplied with the LO with exactly the same frequency:

\[
y_i(t) = \left( m_i(t) \cos(2\pi f_c t + \phi) + m_q(t) \sin(2\pi f_c t + \phi) \right) \cdot \cos(2\pi f_c t + \phi)
\]

(4.4)

In (4.4) \( y_i(t) \) is the output of the receiver for the in phase message component. \( \phi \) and \( \hat{\phi} \) are respectively the phase of the received signal and the phase of the local oscillator.
Using the trigonometric relations (4.4) becomes:

\[
y_I(t) = \frac{1}{2} m_I(t) \cos(\phi - \ddot{\phi}) + \frac{1}{2} m_I(t) \cos(4\pi f_s t + \phi + \ddot{\phi}) + \frac{1}{2} m_Q(t) \sin(\phi - \ddot{\phi}) + \frac{1}{2} m_Q(t) \sin(4\pi f_s t + \phi + \ddot{\phi})
\]

(4.5)

If low pass filtering filters out the high frequency component then (4.5) becomes:

\[
y_I(t) = \frac{1}{2} m_I(t) \cos(\phi - \ddot{\phi}) + \frac{1}{2} m_Q(t) \sin(\phi - \ddot{\phi})
\]

(4.6)

By following the same routine to find \(y_Q(t)\), where the LO is phase shifted 90 degrees, \(y_Q(t)\) is found in (4.7).

\[
y_Q(t) = \frac{1}{2} m_Q(t) \cos(\phi - \ddot{\phi}) + \frac{1}{2} m_I(t) \sin(\phi - \ddot{\phi})
\]

(4.7)

When \(\phi\) and \(\ddot{\phi}\) are equal \(y_I(t)\) and \(y_Q(t)\) become:

\[
y_I(t) = \frac{1}{2} m_I(t)
\]

(4.8)

\[
y_Q(t) = \frac{1}{2} m_Q(t)
\]

(4.9)

The effect of a mismatch in phase is clear from (4.6) and (4.7). If there is a mismatch between the phases, the quadrature component leaks to the in phase component and visa versa. If we consider the message as a complex vector consisting of the in phase and quadrature component, the vector is rotated by the difference in phase. This effect is important for smart antennas, as the phase component of the received signal is present in the demodulated signal.

Another receiver architecture is the heterodyne receiver. The principle is to use of more than one mixer stage, to convert the radio frequency to the base-band. The concept of the heterodyne receiver is visualized in Figure 4.3.
Instead of converting the received signal directly to the base-band, the first stage converts the signal to an intermediate frequency range. After this the signal is bandpass-filtered to filter one of the resulting images and after that the signal is converted to in phase and quadrature signals. The last conversion is based on exactly the same principle as the direct conversion receiver. The use of the heterodyne receiver has certain practical advantages.

As stated before, the received signal for digital beam-forming should be an I/Q signal, to represent the phase and amplitude of the incoming signals. Certain modulation types do not need an implementation where the signal is converted to the I/Q plane to finally detect the bits. For FSK a different method can be used, which is called the FM detector. Every commercial solution for an FSK receiver uses this FM detection method.

The FM detector is a detector where frequency is translated to amplitude. This type of detector will in practice only work in a certain frequency range and can be regarded as the reversed version of the VCO. The architecture is represented in the second stage of the heterodyne receiver structure in Figure 4.4.
The first mixer stage remains the same as the previous heterodyne receiver; the second stage however is not using the in phase and quadrature detection but the FM detection method. The signal from the mixer is limited and after that, multiplied with a phase-shifted version of itself. The phase shift is done by an external network, which provides a phase shift depending on the frequency of the signal. After multiplying the signal, the result is filtered by a low pass network. If the incoming signal is represented by:

\[ r(t) = \cos\left(2\pi f(t) t + \phi \right) \]

\[(4.10)\]

In (4.10) \(f(t)\) indicates the frequency modulation, where the frequency of the signal changes in time. The actual message, which is the source for the frequency modulation is left out of the formula for simplicity.

\[ r_r(t) = \cos\left(2\pi f(t) t + \phi + \theta(f) \right) \]

\[(4.11)\]

The phase shift by the external network is frequency depending, which explains the phase shift \(\theta(f)\). Both (4.10) and (4.11) will be multiplied in the mixer:

\[ r(t) r_r(t) = \cos\left(2\pi f(t) t + \phi \right) \cos\left(2\pi f(t) t + \phi + \theta(f) \right) \]

\[ = \cos(\theta(f)) + \cos\left(4\pi f(t) t + 2\phi + \theta(f) \right) \]

\[(4.12)\]
A low pass network removes the high frequency component, which is the second term in the result of (4.12). The resulting term is depending only on frequency of the input signal. The phase delaying network will have a phase shift of 45 degrees for the carrier and 90 degrees for the two frequencies on which the signal is modulated. An important aspect is that the initial phase $\phi$ of the input signal is removed.

The result is only depending on the received frequency. The effect of the limiter on the system is that the first term in the result of (4.12) is not a cosine but a linear function. Phase preservation is necessary for a beam-forming antenna array and therefore the FM detector is not suitable for a beam-forming antenna.

4.2.1 Receiver building blocks

The amount of different RF hardware products is very large. Often, these RF products are integrated circuits. To understand the receiver it is important to understand the separate parts of the receiver. The receiver can be split up in the following building blocks.

Antenna

The antenna is the interface between the receiver and the free air. The antenna has many characteristics as gain, bandwidth, radiation efficiency, beam width, and beam efficiency. The antenna is the interface between air and receiver and therefore the signals must be transferred as good as possible. The antenna should be impedance matched between the free air and the receiver input.

Low-noise amplifier

The low-noise amplifier (LNA) is the first amplifier in the receiver chain. Its influence on the noise figure is strong compared to the subsequent amplifiers. The amplifier must have a high gain and a very low noise figure. Too much gain compresses amplifiers in the rest of the circuit. A tradeoff must be made between gain and noise figure.
**RF filters**
The RF filter is necessary to filter the desired signal from out-of-band noise. Especially the so-called image frequency, which has the same frequency difference to the LO as the desired signal, can distort the signal. This way only the desired signal is transferred to the IF frequency.

**Mixer**
The mixer is a circuit, which is injected with the received signal and a reference signal from an oscillator. The mixer converts the desired frequencies to the IF band. This requires the need for an RF filter as stated before.

**Local Oscillator**
An oscillator generates the reference signal for the mixer. The oscillator consists of a phased lock loop and a fixed reference oscillator. The fixed oscillator can be a digital circuit or a crystal oscillator. The phase locked loop will lock to a divided version of the reference oscillator. By using a low pass filter in the phase locked loop, phase noise is filtered out to get a steady oscillator. This loop filter slows down the lock-time of the loop to the divided reference oscillator. This means that the loop filter should be narrow enough to limit oscillator spurs, but wide enough to have a fast lock time.

**IF filter**
Only the IF frequency range is passed by the IF filter. The summed result of the LO and the carrier frequency is removed as well as out of band noise.

**Detector**
The final stage is a detector to convert the signal to a suitable baseband signal. The type of detector is depending on the modulation technique used. For FSK an FM detector is used, which converts the frequency shifted signal to an analog NRZ signal. The I/Q demodulator is necessary if a form of quadrature modulation is used.
4.2.2 Receiver architectures in practice

The direct-conversion receiver structure is simple in theory but difficult to realize in practice. Certain problems make the realization of this structure nearly impossible. The first problem is the leakage from the local oscillator to the antenna port. As the LO has exactly the same frequency as the carrier signal, this received signal is indistinguishable from the transmitted signal from other systems in the vicinity.

At higher frequencies, more problems arise, due to the effect of devices and circuits which act as antennas. The second problem is the leakage from the RF port to the LO’s VCO. This effect only occurs at strong signals, which can pull off the VCO’s frequency. Small phase shifts will induce the shifting of the VCO’s frequency, with strong effects on phase-modulated systems. Therefore the problem is fewer presents then the leakage from the oscillator to the RF port, but still degrading the performance of the receiver. The heterodyne receiver concept is not suffering from the above problems as the LO is not the same as the carrier. The heterodyne receiver is widely used in all types of receivers from mobile terminals to FM radio receivers. The heterodyne receiver has improved a lot in the last few years because of new technologies. RLC filters, ceramics, quartz crystals and surface acoustic wave devices have resulted in a high quality receiver structure. However, due to these costly extra devices in the heterodyne receiver, the direct conversion receiver is gaining popularity again and special on-chip architectures compensate for the previous described problems.

The integrated circuits available are a combination of different types of building blocks to get the best receiver performance. Different architectures exist, resembling one of the discussed receiver structures [10]. Many approaches are taken to design an architecture conform its design parameters. The common design parameters for a receiver vary from low cost to low power, complexity, noise performance, gain and so on.
There is one aspect however that is similar for all integrated receivers. Filters are difficult to realize on a chip. The consequence is that the signals leave the chip to be filtered and will return to the chip afterwards. This is one of the most difficult aspects of designing an RF receiver. When the high frequency signals leave the chip, the distances over which they travel are larger than on-chip. Small I/O pins on the chip and the PCB lines suddenly become small antennas that pick up noise and introduce distortion and feedback.

The filters generally consist of the following types:
* Surface Acoustic Wave filter (SAW)
* RLC filter
* RC filter

The SAW filter is one of the most common RF filter nowadays and it is actually one component. The advantage is that they are available in a wide range of frequencies or bandwidths. They are characterized by a sharp cut-off, hence being very frequency selective. The disadvantage is that they are ceramic and therefore very expensive. Furthermore SAW filters suffer from high insertion losses. Besides RF filters the SAW filter is also used as IF filter when the frequency is relatively high. The output of a local oscillator can be filtered also to eliminate noise.

The RLC filters are common types for IF filters and consist of a resistance, inductance and capacitance. They can be formed in many ways and different orders. RLC filters involve multiple components and they are therefore subjected to noise and feedback. If the IF frequency is low, even a low-pass RC filter can be used. Low pass filters are also common to filter outputs of the detector. The implementations of different systems range from application-specific single-chip solutions, to custom configured solutions with multiple chips.

The most interesting bands for experimenting with RF equipment, without the need for a special shielded chamber, are summarized in Table 1.
Table 1: Bands suitable for experiments

<table>
<thead>
<tr>
<th>System</th>
<th>Frequency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECT</td>
<td>1.800-1.900 GHz</td>
<td>DECT is a standard for personal cellular systems. The DECT system is for indoor telephone and data systems.</td>
</tr>
<tr>
<td>ISM</td>
<td>2.400-2.483 GHz</td>
<td>The ISM (Industrial, Scientific, Medical) band for indoor applications. The transmit power is limited. The band will be occupied with hiperlan and bluetooth devices</td>
</tr>
<tr>
<td>ISM</td>
<td>868 MHz</td>
<td>Another ISM band</td>
</tr>
</tbody>
</table>

It should be able to do experiments in the DECT band without disturbing too much other DECT systems. The range of a DECT system is limited and the number of channels is quite high. When there is not a data DECT system in the area, which can use multiple systems, but only phone systems, the chance is limited that the disturbance of one channel would be a problem.

The 2.4GHz ISM band is one of the most interesting bands, as the ISM band is totally free to use. The wide bandwidth is interesting as well, it give a lot of space for the hardware to operate in. The lower ISM is another option. Though quite narrow, the frequency is low, easing the constraint for the hardware. However a larger wavelength will require larger antennas and a larger array, which can be a disadvantage.

4.3 RF design proposal

Due to the lack of experience in the field of RF design, the actual implementation of an RF front-end was not possible in this thesis. An effort has been made to design the RF front-end for a beam-forming system and the result is a general RF design proposal. The design can be regarded as a high-level reference design, which fulfills the demands for a beam-forming network. Before starting with the design, the following guidelines are followed to form the design proposal:
The building blocks will be non-single chip solutions

It is necessary to be able to build a custom and flexible design. The single chip solutions are not suited for a beamforming system. Most of the single chip solutions are designed for a non-suitable frequency range e.g. GSM or D-AMPS etc. The single chip solutions in a suitable frequency range (for DECT or upbanded DECT) are using an FM detector as described earlier in this chapter. This type of solutions is not suitable for a digital beamforming system.

The building blocks are available in the form of evaluation modules.

To be partly depending on the actual implementation issues concerned with RF design a evaluation module is a good starting point for understanding and experimenting with the RF hardware. Most of the evaluation modules are properly designed PCB’s with impedance matched inputs and outputs. A disadvantage could be the fact that the combining of multiple evaluation modules involves signals traveling over long distances when the evaluation modules are connected to each other.

The receiver will be able to preserve phase and amplitude information

The downconversion method must be able to preserve the phase and amplitude of the signal. Most of the time the so-called mixers are sufficient for this function.

The receiver elements can be synchronized

To be sure that the total receiver is not suffering from independent phase drift for the different radios, it is important that the receiver modules or building blocks can be synchronized in a way. This normally involves synchronization of the LOs, or the use of one LO for different radio-chips.

IF sampling is used, which eliminates the use of a frequency detector

IF sampling involves the conversion of IF frequencies to the digital domain. This means that the second analog stage in a receiver is not required anymore, which eliminates the use of analog filters and another local oscillator. The usage of IF sampling involves fast AD converters which can sample rather high IF frequencies. As the current AD converters are very fast ranging up to one gigasamples per second, the
IF frequency is not so much of a problem. The digital hardware following the AD converters to perform downconversion and demodulation could be a problem, as the total throughput is subjected to the processors limited memory bandwidth and processing power. In that case, additional hardware is necessary to fulfill digital downconversion using for example an FPGA or ASIC digital downconverter. Figure 4.5 shows the design concept following the earlier mentioned guidelines.

The design uses a MAXIM IC, which contains an LNA and a mixer per chip. The LO signal may be generated with a VCO module which are also available from MAXIM. LO needs to be divided 4 inputs. This involves power-splitting to be sure that all impedances are matched. For filtering a Surface Acoustic Wave device (SAW) could be used. These types of filters are known for its narrow filter characteristics. For IF frequencies with lower constraints a RLC bandpass filter can be used or just a lowpass filter to remove the high frequency component. After that the signals are filtered. Impedance matching is less problematic at IF frequencies.

If enough RF expertise is available a custom PCB could be designed containing an appropriate number of RF downconverters combined with micro-strips for powersplitting and impedance matching. The actual implementation of the design
concept can be found in the next section of this chapter. In the implementation programmable function generators replace the RF hardware.

4.4 Digital receiver

The digital receiver is part of the test-bed in the form of a digital downconverter. The sampled input signal is downconverted digitally to an in phase and quadrature stream. The type of modulation that is selected is minimum shift keying (MSK) or if desired Gaussian minimum shift keying (GMSK). Furthermore the signal needs to be demodulated. The signal is converted from the I/Q signals to an actual bit sequence.

Instead of performing the conversion from radio frequency to the baseband using analog integrated circuits, it is possible to do the conversion partly analog and partly digital. The current state-of-the-art digital hardware is able to process digital signals up to one gigahertz. For certain receivers an all-digital approach is possible. In that case, the antenna output is amplified, and sampled by a high speed AD converter. Digital algorithms perform the complete transformation from the RF domain to the baseband.

When the digital hardware is less powerful, the RF frequency might be too high to be sampled and processed. It is possible to sample the analog signals after downconversion to the IF. The translation of the IF domain to the baseband can then be done in digital hardware. This is called IF sampling, also known as a form of direct sampling.

The digital downconversion can be done by configurable hardware or by software on a general-purpose processor. This functional unit, which performs the downconversion, is called a digital downconverter (DDC). For digital communications, the generalized digital receiver is comparable to the analog version. Instead of using an analog multiplier and a local oscillator, the software or digital hardware fulfils these functions. Fig. 4.6 shows the generalized digital receiver.
If the resulting I/Q signals are oversampled with a large factor, the I/Q should be converted to a lower sample rate, which is called sample rate conversion.

When the IF frequency is one fourth of the sample frequency, both the digital oscillators are reduced to a repeating vector of [1 0 1 0] for the ‘cosine’ oscillator and [0 1 0 1] for the ‘sine’ oscillator. The digital multiplier can perform multiplication at an even lower rate, as half of the numbers to multiply with is zero. This is interesting as it reduces the load on the processor, or it simplifies the architecture for a digital downconverter. A FIR filter is sufficient to filter the output of the multipliers to remove the summed component. The result is that the input signal is converted to a baseband signal.

To implement an MSK receiver on a DSP, several functional blocks are necessary. If the I and Q signals are available, the receiver can be build using a differential detector, a frequency compensation loop, and a bit clock recovery loop, followed by hard decision as represented in Figure 4.7.

Figure 4.7, digital receiver with frequency compensation and bit-clock recovery
4.5 The Test-bed: System overview

The system will be built following the heterodyne receiver concept, which is shown in Figure 4.8.

![Diagram](image)

Figure 4.8, basic concept of the smart antenna receiver

The RF part consists of one mixer stage. The mixers are synchronized with one local oscillator and the output IF signal is filtered to reject images and to reduce noise. The IF signal is then sampled, and the digital downconverters convert the signal to in-phase and quadrature baseband signals. Beamforming is performed digitally.

Because there is no RF hardware available, the thesis will focus on the implementation of the digital part. To implement a test-bed, without the proper RF hardware a solution is found in programmable function generators or in arbitrary function synthesizers. The generators simulate the RF hardware, as they can be programmed with the appropriate signals. Besides that, the generators can be synchronized, which will prevent drift of the local oscillators.

The AD converters in figure 4.8 are implemented by a quad channel AD converter of Texas Instruments, the THS1206. The digital downconverter and beamform algorithm can be implemented in software, running on an evaluation module for the Texas...
instruments TMS320C6711. Both the selected AD converter and the DSP board are compatible with each other. The configuration is visible in Figure 4.9.

![Configuration Diagram](image)

Figure 4.9, set-up of the test-bed

The working of the system is as followed. All inputs are sampled by the sample and hold units simultaneously. A multiplexer feeds the signals to the AD converter sequentially. The AD converter fills its FIFO until the buffer is almost full, depending on its speed and configuration. At 8 or 12 samples in the FIFO it will indicate to the EDMA controller that it is ready to send over the data. This is done by using an external interrupt. The EDMA controller is triggered by the external interrupt and it will copy the data from the AD converter's FIFO to an assigned memory block (large buffer). When this large buffer is full, the EDMA controller generates an interrupt, which indicates that the data in the buffer is ready to be processed by the digital signal processing algorithms. This EDMA controller takes the data acquisition load completely to reduce processor usage. The processing power is now available for the digital downconversion and the beamforming algorithm.

The programmable function generators (MFG206) can be programmed with a range of 16000 values between +2047 and 2047. The signal type that is chosen as modulation type is minimum shift keying (MSK) or, if desired, Gaussian minimum shift keying (GMSK). The sample frequency is 4 times the IF frequency to be sure that downconversion can be done as discussed in section 4.4 of the Chapter. The system uses digital beamforming based on the LMS or CM algorithm.
4.6 Hardware implementation

The total hardware of the system consists of a multi-channel AD converter, a DSP board and an Arbitrary Function Synthesizer. The DSP is available on an evaluation board and the AD converter on an evaluation daughterboard. This set-up has certain advantages. The system is widely usable, because of the general-purpose hardware. The AD converter is not exceptionally fast, but provides enough bandwidth to start with. The DSP board and its software environment offer enough flexibility for fast implementation of the smart antenna system. The hardware is fast enough for system development, but must be considered too slow as a complete platform for software-defined radio with or without beamforming. For simplified communication systems and custom defined radio systems, the system is satisfying. The next sections discuss the details on the hardware.

4.6.1 Texas Instruments TMS320C6711 development board

The development board has the following facilities:
- 150MHz TI floating-point DSP
- 16 Mb RAM
- Onboard AD/DA converter
- Parallel port interface
- Emulation JTAG controller
- 2 line I/O
- EVM compatible Daughter card Interface
- 128K Flash ROM

The C6711 processor can be split into three parts, the CPU core, the peripherals and the memory. Eight functional units operate in parallel split up into two equal sets. This is because of the dual data path that is available in the processor. These units use two register files of 16 32-bit registers. Figure 4.10 shows the block diagram for the CPU.
Algorithms that need to be executed fast must be optimized for the functional units and its data paths. The optimization can be done by hand in assembly, or partly in C by the compiler. The compilers ability to optimize code is limited and the result is normally not as good as when optimizing assembly by hand. A way to build C code, called 'software pipelining', enables the compiler to optimize the code more easily. The functional units are split up in two multipliers and six ALUs. There is no dependency check in the processor, which means that all of the instructions are checked for dependencies and paralleled by the compiler.

Furthermore the DSP contains a lot of peripheral hardware. Among others the following important peripherals are available:

- Enhanced Direct Memory Access (EDMA) Controller
- Enhanced memory interface (EMIF)
- Multi-channel Buffered Serial Port (MeBSP)
- Timers
These features are important for building communication systems. The EDMA controller can be used to efficiently transfer the data from an ADC to the DSP's memory, without any processing power. The EMIF enables most memory types to be supported including the asynchronous memory interface of the ADC. The McBSP enables multiple serial devices to communicate over the same port. Multiple DSPs can be connected, if desired, and communicate through the McBSP [70]. The timers are necessary for periodic functionality, such as generation of a clock for an AD converter.

4.6.2 THS1206 AD converter evaluation module

The system contains a unit that provides sampling possibilities for the CPU board. On the EVM expansion interface it is possible to connect a daughterboard. In this case a daughterboard is chosen which contains a high speed AD converter THS1206. The AD converter is capable of sampling at 6MS/s and has four inputs, which can be sampled simultaneously. Figure 4.11 shows a block diagram of the THS1206.
Figure 4.11, Block diagram THS1206 AD converter

The four inputs can be configured to run in single ended or differential mode or a combination of both. The resolution is 12 bit and a 16-word FIFO buffer will take the load from the processor, because the processor can receive the data in burst mode. Note that, when using multiple inputs, the sample rate will be divided over the number of used channels. When the AD converter is configured for 3 single ended inputs, for example, the sample rate per channel will be reduced from 6 MS/s to 2 MS/s. The maximum input frequency is 54 MHz, which enables IF bandpass sampling. IF bandpass sampling cases the constraints on IF sampling, as signals higher than the Nyquist rate can be sampled, provided that the bandwidth is small enough for bandpass sampling [65].
4.6.3 Arbitrary Function Synthesizer MFG206

Features of MFG206:
1. 1mHz to 20MHz / 30Vp-p (at open load)
2. High accuracy and stability of frequency due to synthesizing method
3. Arbitrary waveform generator function (10Mwords / s, 12bits, 128Kwords)
4. BURST / GATE functions
5. Dual functions of linear and logarithmic sweep
6. RS-232C (standard) and GP-IB (optional)

MFG206 synthesizer is a function generator equipped with a phase-locked loop (PLL) circuit that is capable of generating a highly accurate and stable frequency. The arbitrary function synthesizer can be used for the following applications:

1. A function synthesizer that can generate sine, triangle, or square wave in the wide bandwidth of 1mHz to 20MHz with a wide amplitude, as high as 30Vp-p at open load.
2. A 12-bit and 128K-word arbitrary waveform generator that has a sampling frequency of 10MHz maximum.

The arbitrary waveform generator creates and edits a given waveform using the Windows 98 as shown in figure 4.12.

Figure 4.12 Windows 98 screen for forming waveform
The waveform is then transmitted to the MFG206 synthesizer. Any waveform can be easily formed and edited using the Windows 98.

**Waveform:**
Horizontal axis: 8 to 131072 (128K) words
Vertical axis: 12 bits

**Sampling frequency**
Setting range: 100Hz to 10MHz / 4 digits setting
Accuracy: less than ±50ppm (0 to 40°C)

**Forming of waveform**

**Standard waveforms:**
Sine, Triangle, Square, Ramp, sinX / X, (1-e^{-ax}), e^{-ax}, White noise and DC

Straight-line: Link two points by a line
Calculation: +, -, ×, Clipping, Absolute, Mirror, Smoothing, Resize, Offset One-point Input: Input each point
Editing of waveform: Cut, Copy, Paste, Undo, Delete
File: New, Open, Close, Save, Save as, Data import, Data export, Print, Printer setup, Transmit, Exit [70].

### 4.7 Hardware performance

The set-up of the beamforming system can be found in Figure 4.9. The performance specifications are summarized in Table 2. The AD converter has a maximum sample rate of 6 MS/s, but this sample rate is divided over the number of channels. The FIFO buffer is circular, which will cause the first memory location in the FIFO to be overwritten when the FIFO is full. For continuous operation, the FIFO should be copied to the DSP's memory, when 8 or 12 samples are in the FIFO. This is necessary as it takes some time to copy the data to the processor. During this time the AD converter will fill the remaining empty memory locations of the FIFO.
The EDMA controller needs a certain time to send over the 8 or 12 samples and during this time the processor can not use the EDMA controller for other purposes. As the EDMA controller is also used to copy data from the external RAM to the cache, the systems performance can be decreased. Therefore the part of the data of the program that is time sensitive should be kept in the cache or the DSP's internal memory as much as possible.

Table 2: Performance specifications of the hardware

<table>
<thead>
<tr>
<th>Performance specification</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum bandwidth 1 channel (Nyquist)</td>
<td>3 MHz</td>
</tr>
<tr>
<td>Maximum bandwidth 2 channels (Nyquist)</td>
<td>1.5 MHz</td>
</tr>
<tr>
<td>Maximum bandwidth 3 channels (Nyquist)</td>
<td>1 MHz</td>
</tr>
<tr>
<td>Maximum bandwidth 4 channels (Nyquist)</td>
<td>750 kHz</td>
</tr>
<tr>
<td>Resolution</td>
<td>12 bit</td>
</tr>
<tr>
<td>System clock</td>
<td>150 MHz</td>
</tr>
<tr>
<td>Maximum input frequency</td>
<td>54 MHz</td>
</tr>
<tr>
<td>Input signal level</td>
<td>-1V to 1V</td>
</tr>
<tr>
<td>Floating point operations per second</td>
<td>900 MFLOPS</td>
</tr>
</tbody>
</table>

Note that the interrupt speed should be kept as low as possible. With a FIFO buffer size of 12 samples the interrupt frequency is reaching 500 kHz, when sampling at 6 MS/s. According to the documentation of Texas instruments this is too high. The interrupt speed should be kept lower then 200 kHz. Though the system works all right at that frequency, it may reduce the bandwidth of the complete system. The AD converters resolution is 12 bit. This means that the dynamic range is 84 dB. For systems on short range or systems using power control this is enough.

If four channels of the system are used the maximum sample rate per channel is 1.5 Ms/s. This indicates a signal bandwidth based on the Nyquist frequency of 750kHz. If the system needs a form of downconversion, the minimum bandwidth is half of 750
kHz, as is necessary in the current setup. Wide band communication systems cannot be sampled, as they require more bandwidth. Also FSK for certain systems could be a problem. DECT uses 2 MHz per band, and the bandwidth is too high for this system. The DECT channel can be sampled with at least 4 Ms/s, which means that no more than 1 channel can be used.

The use of the system as a beamforming system requires too much bandwidth for this communication standard. Simply increasing the sample rate with faster AD converters will not be sufficient as the system is balanced using the DSP, memory, and AD converter. If the system needs more bandwidth, the processing power must be increased, by using more processors or by adding configurable hardware as an FPGA between the AD converters and the DSP.

It is clear that the system cannot function as a system to implement a complete working platform for one of the existing communication system as DECT or GSM. The current set-up is fast enough to test algorithms, and design real-time systems for custom data communications. Software objects can be developed, and are compatible with other general-purpose solutions. Commercial options for a system with more processing power are available as multiprocessor systems. Custom design of a system is not possible without special expertise on high-speed electronics and system design.

4.8 Software implementation

This section will discuss the code composer studio (CCS DSK2) and the software tools, which consist of the real-time operating system DSP/BIOS, its libraries and plug-ins. Section 4.9, will discuss the actual software implementation on task level.

4.8.1 Using the CCS DSK2

Steps to connect the DSK board and the PC host:
1 Turn off the PC power.
2 Connect the male end of the DB25 printer cable to the PC parallel port.
3 Connect the female end of the DB25 printer cable to the DSK board.
4 Plug the power supply into a wall outlet.
5 Connect any other cables (headphones, microphone, etc)
6 Connect the 2.5mm connector of the power supply cable to the power supply connector (J4) on the DSK board.
7 Turn on the PC power.

To develop and run a simple program:

Create a project from scratch, add files to it and review the code. After build and run the program, one will learn how to change build options using the build options dialog, and fix syntax errors using the editor. Finally, one can learn how to utilize basic debug techniques like breakpoints, watch windows and file I/O.

Creating a New Project

1. If you installed Code Composer Studio in c:\ti, create a folder called volume1 in the c:\ti\myprojects folder. (If you installed elsewhere, create a folder within the myprojects folder in the location where you installed.)
2. Copy the contents of c:\ti\tutorial\target\volume1 folder to this new folder.
3. If you have not already done so, from the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio →Code Composer Studio. (Or, double-click the Code Composer Studio icon on your desktop.)
4. From the Project menu, choose New.
5. In the Project Name field, type volume1.
6. In the Location field, browse to the working folder you created in step 1.
7. In the Project Type field, select Executable (.out).
8. In the Target field, select the target you have CCS configured for and click Finish.
9. Code Composer Studio creates a project file called volume1.pjt. This file stores your project settings and references the various files used by your project.
Adding Files to a Project

1. Choose Project→Add Files to Project. Select volume.c and click Open. You can also add files to the project by right clicking on the Project View icon and choosing Add Files to Project or by dragging and dropping files into folders in the Project View window.

2. Choose Project→Add Files to Project. Select Asm Source Files (*.a*, *.s*) in the Files of type box. Select vectors.asm and load.asm, and click Open. These files contain assembly instructions needed to set the RESET interrupt to branch to the program's C entry point, c_int00. (For more complex programs, you can define additional interrupt vectors in vectors.asm, or you can use DSP/BIOS to define all the interrupt vectors automatically.)

3. Choose Project→Add Files to Project. Select Linker Command File (*.cmd) in the Files of type box. Select volume.cmd and click Open. This file maps sections to memory.

4. Choose Project→Add Files to Project. Go to the compiler library folder (C:\6000\gctools\lib). Select Object and Library Files (*.o*, *.lib) in the Files of type box. Select the rt6200.lib file for the target you are configured for and click Open. This library provides run-time support for the target DSP. For some targets, the runtime library may have a more specific file name, for example, rt6200.lib.

5. In the Project View window, right-click on volume1.pjt and select Scan All Dependencies. volume.h should appear under the Include folder in the Project View window.

6. Expand the Project list by clicking the + signs next to Project, volume1.pjt, Libraries, and Source. This list is called the Project View.
Opening Project View

If you do not see the Project View, choose View→Project. Click the File icon at the bottom of the Project View if the Bookmarks icon is selected. You do not need to manually add include files to your project, because Code Composer Studio finds them automatically when it scans for dependencies as part of the build process.

After you build your project, the include files appear in the Project View. If you need to remove a file from the project, right click on the file in the Project View and choose Remove from project in the pop-up menu. When building the program, Code Composer Studio finds files by searching for project files in the following path order:

- The folder that contains the source file.
- The folders listed in the Include Search Path for the compiler or assembler options (from left to right).

Then review the source code and lastly build and run the program [71] [73].

4.8.2 DSP/BIOS

DSP/BIOS is a set of APIs, tools and plug-ins for the development of real-time applications for the TI DSPs. The tools facilitate program generation, and testing. The API standardizes DSP programming for TI chips. The plug-ins varies from real-time analysis tools to data exchange and debugs features.

DSP/BIOS real-time library and API

A small firmware DSP/BIOS is available for basic runtime services. This means that a lot of features in a real-time system do not need to be developed. The DSP/BIOS and API can capture information from the target program. It includes a software interrupts manager, a clock manager, I/O modules and more.

1895
**Code generation tool**

The parameters for the DSP/BIOS can be configured with a special configuration tool. The interface is graphical and can be used to configure for example the software interrupts, hardware interrupts, and real-time data exchange options.

**The DSP/BIOS plug-ins**

For probing, tracing and monitoring DSP applications special plug-ins can be used. The plug-ins has minimal impact on the real-time application. The host takes care of formatting, analyzing and displaying the data, to unload the DSP from these tasks.

**4.8.3 Real-time analysis**

With the real-time analysis components it is possible to acquire data on the fly and determine the application’s behavior. The conventional method for debugging is running the target on the DSP until an error occurs. By adding breakpoints and re-executing, information is gained on the error. This type of debugging is not very efficient when developing real-time systems, as the system suffers from timing constraints, and its behavior is non-deterministic.

The continuous operation of a real-time system is important. The instrumentation API of DSP/BIOS is designed for this type of continuous debugging. The instrumentation APIs includes different modules, for instrumentation. The two most used modules are the message log manager (LOG) and a statistics manager (STS).

**Log (message Log Manager)**

This module gives information about events. The API can display system events, but also programmed messages can be send to the LOG system. The processor stores the messages in LOG buffers, and will only be formatted after it is transferred to and stored on the host, where the data is available for displaying and analysis. The host will regularly collect the buffer with messages from the DSP, which will keep the necessary amount of memory low. Figure 4.13 shows the window, which displays the messages to the object 'trace' from the application. A log manager can be compared
with the standard I/O functionality of printf, only now the formatting is done on the
host and the messages can be send to different log objects.

Figure 4.13, Log window for the trace object

STS (Statistics Manager)
These objects can capture the count and the maximum, average and totals for objects
in real-time. If the object is a task, it will indicate the number of instructions required
to execute the task. Both software, and hardware interrupts can be monitored. It is
possible for the program to generate its own statistics objects. The statistics are
accumulated on the target, but the host performs the actual calculations on the
statistics. The host polls regularly for data and resets the statistics on the target, to save
memory on the target. Figure 4.14 shows the window, which displays the statistics for
various software interrupts and variables.

Figure 4.14, the statistics window
Another instrumentation plug-in is the 'execution graph', which can be used to view the execution of the different parts of the program as software interrupts, periodic timers and clocks. The clock and periodic timers are necessary to provide measure of time intervals. The system does not timestamp each event, as this is a processing and memory bandwidth-consuming job. Figure 4.15 shows the execution graph.

![Execution Graph](image)

Figure 4.15, the execution graph

### 4.8.4 Real-time program structure

The DSP/BIOS program consists of the different threads, which have different typical properties. Threads with high priorities will be executed before threads with lower priorities. A thread can even be disrupted to let a higher priority thread precede. In a typical real-time program the important threads are activated by events, and only if an event occurs they are scheduled. Certain threads with very low priority are only scheduled if no events occur. The operating system keeps track of all the threads and when they need to be executed. The threads can be split up in three types of threads.

**Background thread**

This type of thread has very low priority. The so-called idle thread will be started if no other execution is necessary. When the host wants data from the processor for statistics or real-time information this can only be done in the idle thread. This means that when the system never enters the idle thread none of the previous mentioned
instrumentation plug-ins can receive their data. Other threads with higher priority will only be executed when no software or hardware interrupt service routine is executed.

**Software interrupts**
The software thread is an interrupt service routine (ISR), which is called by software functions. The SWI service routines add priority levels between the hardware interrupts and the background threads. Due to the response time of SWIs they should only be used for events with deadlines of 100 microseconds or more.

**Hardware interrupts**
Hardware events that occur in the DSP's environment can trigger the ISR for the event. The hardware interrupts are used for time critical events, and the ISR will be scheduled directly. Hardware interrupt service routines can be processed quickly, but should be completed within 2 to 100 microseconds.

The software ISR is called when an SWI is posted. The sender can set some additional options in the mailbox of the ISR. These options are checked before actually posting the SWI. This makes the software ISR more flexible in its response to an SWI. The software ISR can, for example, configured to be started after a number of posts have been made. It can also configured to be started when two other objects have reported to be ready. If an SWI is posted twice, the choice can be made whether the ISR should be executed twice or once. The SWI manager enables all these properties to be set and followed, which makes the SWIs very flexible.

**4.9 Software design**
The approach to the beamforming system is closely related to software radio and therefore a large part of the receiver and beamformer is implemented in software objects. The system uses the real-time operating system DSP BIOS, as described in the previous section. This section will discuss the aspects of the system design on software level and the different objects that can be found in the system. The system can be described on two levels:
1. Thread level
2. Function level

The thread level is related to objects as real-time threads, which are part of the operating system. The function level is related to low-level functions, which perform mathematical computations. This section discusses the thread level design only, which consists of software objects for real-time purposes, and for this a set of properties is described:

1. Name; specifies the name of the object
2. Event; specifies the event type that triggers the object.
3. Input variables; input of the object
4. Output variables; output of the object
5. SWI posts; the SWIs that can be posted by the object
6. Description; describes the functionality of the object
7. Function calls; low-level function calls;

Figure 4.16 shows the objects included in the system. All objects are software interrupt service routines, except the 'Process buffer' object.

![Diagram](image)

Figure 4.16: Software objects as part of the software-defined DDC

And beamform algorithm
Start AD conversion

This object configures the AD converter and performs the programming of the EDMA controller. The EDMA controller needs to be programmed to copy data from the AD converter and generate and interrupt when ready.

Process buffer

This object is triggered by an external interrupt, generated by the EDMA controller. The EDMA controller is configured to use two buffers, called a ping and a pong buffer. The process buffer object is a hardware interrupt service routine, and its functionality is restricted. The routine checks whether the ping or pong buffer is ready and depending on this it will start the DDC and beamforming objects.

DDC ping/pong

This object performs the actual downconversion of the channels. It operates either on the ping or the pong buffer. The downconversion consists of mixers and filters. The input is either the ping or pong buffer, and its output consists of the I/Q signals for different channels.

Beamform

This object will perform the actual beamforming algorithm. For this the CM algorithm is used. The input is consists of the I/Q signals for different channels. Its output is one I/Q signal representing the modulated received signal. The following sections describe the thread level functionality, according to its properties.

Debug

The debug object is a thread that can be used for various debug functions. It can be used to halt the system, by placing a breakpoint in the thread. The other objects can call the debug object when an error occurs, and the system is halted. Special debug threads as this, with custom defined priorities, are useful, as the system's debugger only executes when the system is in idle mode. In the next section the threads are discussed in more detail.
4.9.1 Start AD conversion

The EDMA controller is programmed to react on the external interrupt 4, which is connected to the AD converter. If the interrupt occurs, the EDMA controller will copy the data from the ADC’s FIFO buffer to a buffer in the internal memory of the DSP. The EDMA is programmed to copy the data subsequently to a so-called ping and pong buffer. If the buffer reaches a defined value (for example, 512 values), a Transfer Complete interrupt is generated. This will enable the software to react on the data in the buffer, which is ready to be processed.

The EDMA controller will reload its parameter table, which contains a different destination address, to start copying to the other buffer. The buffer is switched every time the Transfer Complete interrupt is generated. The EDMA controller is programmed to use the global variables ad_buffer_ping and ad_buffer_pong as ping and pong buffer. The buffers are interleaved, as the interleaving is done inside the ADC’s FIFO buffer. No de-interleaving is performed and the data is copied directly from the FIFO to the buffers by the EDMA controller. The properties of the object start AD conversion are given in the next table.

Table 3: Properties of the object start AD conversion

<table>
<thead>
<tr>
<th>SWI object</th>
<th>StartConversion()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>None, called in main();</td>
</tr>
<tr>
<td>Input variables</td>
<td>None</td>
</tr>
<tr>
<td>Output variables</td>
<td>ad_buffer_ping(global)</td>
</tr>
<tr>
<td></td>
<td>ad_buffer_pong(global)</td>
</tr>
<tr>
<td>Description</td>
<td>Part of the initialization. Programs the EDMA channel and starts the conversion.</td>
</tr>
<tr>
<td>SWI posts</td>
<td>None</td>
</tr>
<tr>
<td>Function calls</td>
<td>edma_pingpong</td>
</tr>
<tr>
<td>Status</td>
<td>Implemented, tested.</td>
</tr>
</tbody>
</table>
4.9.2 Process ping or pong

When the EDMA controller completes its task, it will initiate an interrupt, called as the Transfer Complete interrupt. Additionally, it will set a number in the interrupt pending register. This number indicates whether the ping or pong buffer is ready to be processed. The object for 'process ping or pong' is an HWI service routine, responding to the Transfer Complete interrupt, and it will post the SWI for DDCping or -pong to start the processing of respectively the ping or pong buffer. The function also posts the SWI for the object 'Beamform', which initiates the beamform algorithm. As both SWIs are defined by a priority, the system will first execute the 'DDC' object (higher priority) followed by the 'Beamform' object.

Table 4: Properties of HWI object EDMA_HWI

<table>
<thead>
<tr>
<th>HWI object</th>
<th>Edma_HWI()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>Interrupt 8 (EDMA Transfer Complete interrupt)</td>
</tr>
<tr>
<td>Input variables</td>
<td>None</td>
</tr>
<tr>
<td>Output variables</td>
<td>None</td>
</tr>
</tbody>
</table>
| SWI posts | &SwiDDCping or &SwiDDCpong  
&SwiBeamform |
| Description | Reads the interrupt pending register,  
Determines ping or pong buffer ready,  
Clears interrupt pending register,  
Clears interrupt  
Starts the digital downconverter  
Starts the beamform |
| Function calls | None |
| Status      | Implemented, tested. |
4.9.3 DDCping/pong

The 'DDCping' and 'DDCpong' objects perform the downconversion. First the function *mixer2channel* is called. Note that the official system would need 4 channels, this can be implemented if the system is ready to perform beamforming for 4 channels. This function de-interleaves the data (*ad_buffer_ping* or *ad_buffer_pong*) and multiplies them with cosine and sine functions. The result is stored in intermediate buffers, which is subsequently processed by the FIR filter function *fir_r8*. To enable continuous operation of the filter without disrupting the data as a consequence of the block input, the state of the filter is stored and recalled when a block data is ready to be filtered. This is done by *fireconfig* and *firrestore*.

**Table 5:** Properties of the SWI object DDCping or DDCpong

<table>
<thead>
<tr>
<th>SWI object</th>
<th>DDCping() or DDCpong()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>&amp;SwiDDCping, &amp;SwiDDCpong</td>
</tr>
<tr>
<td>Input</td>
<td><em>ad_buffer_ping</em> or <em>ad_buffer_pong</em> (global)</td>
</tr>
<tr>
<td>Output</td>
<td><em>I1out, Q1out, I2out, Q2out</em> (global)</td>
</tr>
<tr>
<td>Description</td>
<td>Reads interleaved data from the <em>ad_buffer_ping</em>, Downconverts and de-interleaves data to I/Q signals per channel</td>
</tr>
</tbody>
</table>
| Function calls | *Mixer*  
*Fireconfig*  
*Fir_r8*  
*Firrestore* |
| Status     | Implemented for 2 channels, tested for 2 channels. |

4.9.4 Beamform

This object performs the actual beamforming. Its input is the I/Q signals from different channels, and its output is the beamformed I/Q signal. The resulting weight vectors (W) are stored for the next time that the object is called. The low-level function that is
called is the actual algorithm that performs the beamforming. The beamforming is based on the Constant Modulus Algorithm and can be used to demonstrate suppression of one interferer.

Table 6: Properties of the HWI object Beamform

<table>
<thead>
<tr>
<th>HWI object</th>
<th>Beamform()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>&amp;Swi/Beamform</td>
</tr>
<tr>
<td>Input variables</td>
<td>I1out, Q1out, Q2out, Q2out, W (global)</td>
</tr>
<tr>
<td>Output variables</td>
<td>beamres1, beamres2</td>
</tr>
<tr>
<td>SWI posts</td>
<td>None</td>
</tr>
<tr>
<td>Description</td>
<td>Performs beamforming</td>
</tr>
<tr>
<td>Function calls</td>
<td>Beamform</td>
</tr>
<tr>
<td>Status</td>
<td>Implemented, tested.</td>
</tr>
</tbody>
</table>

4.10 Software to be implemented

The receiver is implemented partly. For example, demodulation is not implemented. The demodulation is necessary to convert the output of the beamformer to a real bit-stream. In the case of the MSK signal, this can be done, by using a differential detector and a bit-clock recovery algorithm to detect the optimum bit instant. When this part is implemented the system can be tested by performing bit error rate (BER) calculations.

The use of the LMS algorithm requires a system where the reference signal is synchronized with the system. This can only be done by using a known bit sequence, and after demodulation, a correlator is necessary to find the sequence. When the correlator has found the start of the bit-sequence, the reference vector could be aligned with received signal from the antenna array. When the reference vector is aligned the system initiate the object for the LMS algorithm.

The current software structure is suitable for a demonstration of simple software-defined radio architecture. When a system gets more complex, the current
implementation is not satisfying anymore. A special framework is necessary to work within. This framework should specify how the software objects are defined and how they should operate. Also the usage of inter-object communication and the usage of data objects should be specified. In the current implementation there is no inter-object communication and the data objects are global variables. If a time constraint is not met within the current system an object may overwrite the data of another object, leading to disruption of the data. These are typical issues that should be covered by a quality real-time framework.