Chapter 6

Results

6.1 Introduction

This chapter will discuss the test results for the system. The parts that need to be tested are the software objects and the overall functionality. The first of this chapter will show the results of the beamforming system's performance and its functionality on thread level. In the second part the digital downconverter is tested functionally. The third part describes the results for the complete system, including beamforming.

The following setup is used:
• Two function generators
• Two channels: each 521 kS/s
• ADC speed: 1042 kS/s
• IF frequency: 130250 Hz
• MSK modulated test vectors 130250 bps

A channel is one of the inputs of the quad-channel THS1206 ADC. The understanding 'channel' is also used for a baseband I/Q representation of the input signal after downconversion.

6.2 Thread test results

For a fast functional analysis on thread level, the execution diagram is used. The execution diagram displays all threads. The execution diagram does not indicate the exact time on which the threads are executed. The timeline is not linear, and only the
execution sequence is represented. This means that there is no relation between the length of a part of the execution diagram and its time. For a timing analysis in program cycles the SWI accumulators are used. The system is configured to use 2 channels. The ‘DDC’ object is called on ping and pong buffer. For stability reasons the beamform algorithm is only executed for the DDC for the ping buffer. Figure 6.1, shows the result of the thread level test.

![Execution diagram](image)

Figure 6.1: Execution diagram

In the execution diagram the white blocks indicate the posting of a software interrupt and the dark blocks indicate that a thread is running. It is not possible to view HWIs or HWI service routines in the execution diagram. Therefore the object ‘process buffer’ described in the chapter 4 is not visible in the execution diagram. The names of the object that start with ‘Swi’ are the software interrupts, which are being posted to initiate its relevant function or thread. The threads that are coupled to the SWI are visible in the execution diagram when they are executed. Objects as TSK_idle and KNL_swi are part of the operating system.

First ‘SwiStartConversion’ is posted, and its service routine programs the EDMA and ADC. Then after a while the object ‘SwiDDCping’ is posted, to start the digital downconversion of the ping buffer. Directly after posting ‘SwiDDCping’,
‘SwiBeamform’ is posted. Due to priorities the thread ‘Beamform’ is executed after the execution of the thread ‘DDCpong’. Once the interrupt service routine for the pong buffer (not visible in execution graph) is called ‘SwiDDCpong’ is posted. The program was configured to only process the thread ‘Beamform’ for the ping buffer, in case it cannot meet its time constraints. The execution graph shows that the DDCpong is called, before the ‘Beamform’ object is started. When the DDCpong object is finished, the ‘Beamform’ object continues. This indicates that the ‘Beamform’ object is not meeting its time constraints, as it continues after the ‘DDCpong’ object has been activated. The system cannot function correctly, as the ‘DDCpong’ object has disrupted the data for the ‘Beamform’ object. Continuous operation is not possible.

The system is configured to work in discontinuous mode, as visible in the execution graph in Figure 6.2. The DDC pong object is never called, and therefore the data for the beamform algorithm is not disrupted during operation. Note that this causes half of the data to be ‘dropped’ as the DDCpong object is not called anymore. From the execution graph it is clear that time constraints are now met.

Figure 6.2: Execution diagram
The program is configured to halt after processing 2 times the ‘ping’ buffer. On the fourth interrupt that indicates that the pong buffer can be processed, the HWI service routine posts two software interrupts.

1. StopEdma
2. Dummy

‘StopEdma’ has the highest priority, and will shut down the EDMA controller. The data from the ADC is no longer copied to the buffers and the Transfer Complete interrupts are no longer generated. ‘DDCpong’ is not called to perform downconversion. The ‘Beamform’ object was interrupted by the ‘StopEdma’ object, because its priority is higher. After the ‘StopEdma’ object is finished the ‘Beamform’ object continues. When this object is ready, the thread with the lowest priority is called, ‘Dummy’, which is used to halt the system on.

The execution graph indicates that the system performs stable on thread level, but doesn’t indicate the performance of the system. This can be monitored by using the statistics manager. With the statistics manager, every thread coupled to the SWIs, can be monitored. It displays the number of posts and the number of instructions of the relevant thread. Custom statistics can be monitored too, by using special statistics functions. The hardware interrupt routine, which responds to the EDMA, is monitored too. Figure 6.3 shows the statistics on all statistics objects. This threads related to the interrupt service routines are visible. The EDMA service routine, which adds statistics on the interrupts, is indicated by ping and pong.

<table>
<thead>
<tr>
<th>Object</th>
<th>Count</th>
<th>Total</th>
<th>Max</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>ping</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>pong</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>SwiStartConversion</td>
<td>1</td>
<td>5208 inst</td>
<td>5208 inst</td>
<td>5208.00 inst</td>
</tr>
<tr>
<td>SwiStopEdma</td>
<td>1</td>
<td>1472 inst</td>
<td>1472 inst</td>
<td>1472.00 inst</td>
</tr>
<tr>
<td>SwiBeamform</td>
<td>2</td>
<td>212944 inst</td>
<td>106924 inst</td>
<td>106472.00 inst</td>
</tr>
<tr>
<td>SwiDDCpong</td>
<td>2</td>
<td>27340 inst</td>
<td>14488 inst</td>
<td>13670.00 inst</td>
</tr>
<tr>
<td>SwiDDCpong</td>
<td>0</td>
<td>0 inst</td>
<td>-2.1474e+0003</td>
<td>0.00 inst</td>
</tr>
</tbody>
</table>

Figure 6.3: Statistics on the objects
The EDMA interrupt routine is executed four times, two for ping and two for pong. The object 'DDCping' is executed twice, and the averages are visible. The digital downconverter processes the buffer, which is 512 values in approximately 13700 instructions, which is around 27 instructions per sample or 18% CPU processing usage. This is based on measurement over a few conversions. Measuring over a longer period indicates that the CPU processing power is less than 14%. This is measured with the SWI accumulators running; without the SWI accumulators as debugging tool, the CPU processing power drops to less than 11%, which is measured with the CPU processing power tool.

The current beamformer uses around 106500 operations. This is too much, because the system cannot meet its time constraints. Improvement of the beamform function must be done by optimization by hand. The main problem of the beamform loop is the use of an integer divide, which causes a jump instruction to a sub routine. The jump instruction prevents that the compiler can optimize its code.

6.3 Digital downconverter

The first step in the digital downconverter is the mixing of the interleaved data from the ping or pong buffer. The next step is filtering the deinterleaved I and Q signals with a low pass filter. Two digital signal generators generate the test vector. In Figure 6.4 the result for both channels is shown.

Fig 6.4: Output of the digital downconverter for two channels, represented by I and Q components.
Both pictures indicate the I and Q component of the baseband signal. The first picture is the result of the DDC for the first channel and the second picture is the result of the DDC for the second channel. The IF signals generated for both channels are equal. It can be seen as a signal from one source, arriving at antenna boresight. The test vector was a sequence of ones followed by a sequence of zeros. The point at around 120 indicates the point where the bitsequence is changed from ones to zeros. The Q channel makes a phase shift of $\frac{1}{2}\pi$. The disruption at the start and around sample 220 are caused by the signal generators, which are synchronized in burst mode. This results in a small disruption between the repeated sequences.

To synchronize two function generators, a third function generator was used, to generate synchronization signals. Every time the third generator pulses the two other function generators, the arbitrary waveform is ‘played’. After the waveform is played, the function generators wait for the next sync pulse. This ‘wait’ causes a disruption in the generated signal.

The MSK signal can be found back in the filtered result. The overall system phase is very stable, which can be seen in the figures by the fact that the phase-shifts are $\frac{1}{2}\pi$ or $-\frac{1}{2}\pi$ at every bit instant. There is no noticeable phase difference between both channels, which indicates no severe phase lag between the function generators. This has been verified on a digital oscilloscope. If a phase lag existed between the function generators, this can be corrected by adjusting the phase of one of the generators internally.

When plotting the one of the outputs of the DDC into an X/Y plot figure 6.5 is found, where every I and Q sample pair of one of the channels in Figure 6.4 is plotted by a dot, representing the complex value. The downconverted I/Q signal has constant amplitude.
6.4 Beamform algorithm

The CM algorithm was chosen to be implemented, as it does not need synchronization compared with the LMS algorithm. For the CM algorithm a modified error is used and (3.22) is modified to:

\[ \varepsilon = \frac{y(n)}{|y(n)|^2} - y(n) \]

\[ \varepsilon \] ..........................(6.1)

The error is changed, where the modulus of \( y(n) \) is changed to a squared modulus. This removes the required square root function for the calculation of the modulus. This square root function is slow and prevents software pipelining by the compiler due to jump instructions, and should therefore not be used in fast algorithms. Both (3.22) and (6.1) have proven to be stable algorithms in MATLAB tests. The implementation of the beamform algorithm suffered from the unstable development environment, which made the implementation process slow. Therefore a successful alternative was used to implement the beamform functions. As the algorithm is not part of the real-time functionality of the system, it was first implemented on a PC in Visual Studio, using linear debugging. This type of debugging is much faster and simpler then the debugging on the DSP. For testing the functionality of the beamform function, MATLAB was used to generate test vectors.
The function was implemented using integers only, for speed purposes. For the implementation, it is very important that no overflow is caused in the algorithm. The 32 bit integers have a range of \(-2^{147483648}\) and \(2^{147483648}\). This is not much as the algorithm requires many multiplys and some divides. This means that through the algorithm, many elements need to be scaled. The scaling is done with bit shifts, as this functionality is available in the DSP's ALUs. This is faster than dividing or multiplying, but the factor for scaling is always a power of 2.

This scaling is necessary, to prevent the algorithm to get unstable due to overflow or underflow. The actual implementation of the algorithm is visible in Appendix C, in the section 'beamform.c'. This C version was compared to the MATLAB simulations and has been tested. The CM algorithm is only implemented for a system with two antennas. Therefore the algorithm can only remove one interferer. After the functional testing of the algorithm, the function has been transferred to the DSP software environment, where the algorithm was added to the real-time system.

The test setup for the system is as followed:
1. Two function generators generate the signals for the two antennas
2. The function generators are to be synchronized by a third reference signal generator
3. There is one 'desired' signal
4. There is one 'interfering' signal

The file 'arbgem2.m' generates the signals, received by the two antennas. The function is implemented for 2 antennas only. The function generators simulate the necessary IF frequencies, by using the generated signals from MATLAB. Two experiments are carried out to demonstrate the beamforming system.

6.4.1 Experiment I

The first experiment demonstrates the beamform algorithm. Two signals are generated to simulate the two antennas and receivers, which are a combination of the signals
specified in table 7. In the experiment both the IF signals that are received are a combination of the desired and interfering signal, where the signals are shifted in time (phase) to simulate the angle of arrival.

Table 7: Specification of the signals

<table>
<thead>
<tr>
<th>Desired Signal</th>
<th>Interfering signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK modulated signal</td>
<td>MSK modulated signal</td>
</tr>
<tr>
<td>150 bit test vector</td>
<td>150 bit test vector</td>
</tr>
<tr>
<td>Long lengths with the same bits</td>
<td>Frequent changes of bits</td>
</tr>
<tr>
<td>Arriving at zero degrees</td>
<td>Arriving at angle of 18 degrees</td>
</tr>
</tbody>
</table>

It has been shown that the beamform algorithm is not fast enough for continuous operation. If the beamform algorithm is started sequentially to the 'DDCping' object, 'DDCpong' is started before the beamform object is ready. Therefore the beamform algorithm is only executed once. The 'DDCping' object is never started, because it will overwrite the data for the 'Beamform' object. Therefore the 'Beamform' algorithm is not running in a continuous manner in the experiment and the 'DDCpong' object is not called. The system drops half of the data blocks, as 'DDCpong' is never started.

Two function generators are programmed with the appropriate signals. The desired signal is arriving at antenna boresight, at an angle of zero degrees. This means that the desired signal is arriving in phase at both the antennas. The interfering signal arrives at an angle of 18 degrees. After 256 samples the beamformer results in an array output as in figure 6.6, which shows the I and Q channels.

Figure 6.6: I/Q signal of array output after beamforming after over 256 samples
This figure represents the output of the array, that is the result of the weight factors found by the beamform algorithm. In Chapter 3.3.1, this signal is represented by \( y(t) \). The figure shows that the algorithm slowly converges. However, the amplitude is not constant yet, which can be seen in the complex plane in Figure 6.7.

![I/Q plot of array output after beamforming over 256 samples](image)

Figure 6.7: I/Q plot of array output after beamforming over 256 samples

The received signal is not an MSK signal, and still a combination of both received signals, the interferer and desired signal.

If the system runs for a long period Figure 6.8 and Figure 6.9 are found. The signal that is shown is clearly an MSK signal and in this case it is the signal that represents the slowly varying bit sequence. Around sample 100, the phase shift in the I channel is clear which indicates that the signal is changing its run of bits. The output shows a disruption around sample 192. This is due to the fact that at this point the signal generators are synchronized, causing a disruption in both signals.

![I/Q signal of array output after beamforming over 2560000 samples](image)

Figure 6.8: I/Q signal of array output after beamforming over 2560000 samples
The resulted weight factors are used in MATLAB to plot the antenna response pattern. Figure 6.10 shows that the found weight coefficients suppress the interferer at around 18 degrees. The suppression is around 12 dB. This makes the signal to interference ratio 18.5dB as the desired signal at zero degrees has a gain 6.5dB.

Figure 6.10: Antenna response, after beamforming with an interfering signal at 18 degrees and a desired signal at zero degrees.
The experiment is repeated for a different angle of the interferer. The angle is chosen now at $-37$ degrees. The found results are represented in Figure 6.11.

Figure 6.11: antenna response, after beamforming with an interfering signal at $-37$ degrees and a desired signal at zero degrees.

The suppression is better than the previous experiment. Now the signal at $-37$ degrees seems better suppressed, but it is not exactly at the minimum of the plot. The minimum is at $-39$ degrees and the suppression at $-37$ is $21\text{dB}$. Note that the signal at zero degrees is amplified with $6.5\text{dB}$, which makes the signal to interference ratio $27.5\text{dB}$. This indicates that even a small mismatch of the minimum of the antenna response, can have strong influence on the suppression.

6.4.2 Experiment 2

In this experiment the same signals are used as in experiment 1 only the desired signal and the interfering signal are swapped. This means that the signal that is arriving close to boresight needs to be suppressed and the desired signal arrives at an angle of $-2.5$. To select the signal at $-37$ degrees, the spatial minimum as a result of the initial weight vector is placed right from the interfering signal. The algorithm will converge to the closest signal to be suppressed, which is the interfering signal at $-2.5$ degrees.
Table 8: Specification of the signals

<table>
<thead>
<tr>
<th>Interfering Signal</th>
<th>Desired signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK modulated signal</td>
<td>MSK modulated signal</td>
</tr>
<tr>
<td>150 bit test vector</td>
<td>150 bit test vector</td>
</tr>
<tr>
<td>Long lengths with the same bits</td>
<td>Frequent changes of bits</td>
</tr>
<tr>
<td>Arriving at antenna −2.5 degrees</td>
<td>Arriving at angle of −37 degrees</td>
</tr>
</tbody>
</table>

The resulting signal with the weight coefficients found by the beamform algorithm is represented in Figure 6.12, represented by an I and Q component and as a complex vector in Figure 6.13. The figures shows a rapid changing signal that is an MSK modulated signal, with frequent bit changes. The amplitude is less constant but ‘open’ as visible in Figure 6.13.

![Figure 6.12: Resulting I and Q channels after beamforming](image)

![Figure 6.13: I/Q representation in the I/Q plane](image)
If the found weight vectors are used to plot the antenna response pattern, Figure 6.14 shows the corresponding antenna pattern. The antenna pattern has its minimum at -4, and at -2.5 the gain is around -10 dB. Signal to interference ratio is therefore 16.5 dB.

![Antenna Response Pattern](image)

Figure 6.14: antenna response, after beamforming with an interfering signal at -2.5 degrees and a desired signal at -37 degrees

Both experiments indicate proper operation of the beamforming. With these experiments the concept of beamforming is demonstrated in a real system. The system is only tested with an MSK signal; in further experiments also other interferers must considered to be tested. Only convergence of the algorithm was demonstrated. The speed of the algorithm was not tested, and the number of samples used by the algorithm was chosen very large, to be sure that the algorithm had converged. To test the quality of the system also the speed of convergence must be tested in future tests.

### 6.5 Experiences with TI tools

The experiences and achievements with the development of software on the TI DSP platform and the use of DSP/Bios have leaded to insight in the system design approach in this environment.
The general experiences can be summarized as followed:

* The learning curve for the TI tools and DSP bios is very steep
* Functional programming should be done with linear debugging
* Real-time functionality of the DSP environment is not stable

The TI tools for DSP/BIOS are interesting to work with when designing functionality on thread level. For designing algorithms on functional level the tools are less suited. This section will discuss the tools and experiences and is followed with a proposal to develop the drawbacks of software development with the real-time tools are listed below:

* **Visualization tools can only be used, when the exact effect on the software is known**
  One of the most important tools in the software environment is the so-called configuration manager. The configuration manager provides a visual interface to all system configurations. The system needs to be understood completely, before the visual interface is easy to work with. The configuration tool can be seen as an interface for changing low-level functionality. It is not a high level tool, and the smallest error in the configuration can cause serious problems.

* **Thread level debugging depends on idle thread**
  When testing the thread level program, there are many reasons why an error can be caused. Most of the errors occur when timing constraints are not reached. Instead of signaling the timing constraints the system hangs on this type of error. If the SWI objects are stacking up, a stack overflow is easily reached. TI has made its tools to take as little processing as possible. The so-called idle thread is called, when no other activity is reported. This means that debugging at task level is only working when your system actually performs satisfying! When timing constraints are not met, the system will never go into idle. This is strange as debugging the real-time errors certainly becomes impossible.
A workaround would be to develop a custom debugging interrupt service routine that is coupled to a timer function on the processor. This means that your system is having less power due to the debug overhead, but the system can be debugged better. If this is done on regular occasion the system is more reliable, as it is performed always at the same time. If it is hardware interrupt routine, coupled to one of the DSP’s timer a very reliable debugging system could be made. The interrupt routine must communicate with a host on the PC to transfer the necessary debug information. All the TI hosts’ tools work through the idle thread. If debugging information needs to be displayed on-screen, a custom host must be implemented. Threads need to be programmed now to communicate with the debugger. The Real-time Data Exchange plug-ins could provide this functionality.

Real-time system crashes are severe

When the idle thread is never started, or a different problem induces a system hang, the resulting system crash does severe damage to the stability of the tools and future performance of the DSP.

By following the next procedure, the system can be restarted:

1. Hard reset the DSP multiple times
2. Software reset the DSP (to verify basic operation)
3. Reload gel-file (initializes the necessary DSP peripherals)
4. Restart program pointer
5. Load the appropriate program
6. If the program cannot be loaded, repeat the above steps
7. If the program cannot be loaded, restart all tools, and turn DSP power off and on.

If a program is loaded, it is still not certain if it will run properly. In many cases proper operation can be derived from the fact that the system will halt on breakpoints. If the system is not halting on defined breakpoints the program is still not functioning. In this case restarting tools and DSP is the best option. A programmed EDMA controller causes also severe system crashes, if the EDMA is not reprogrammed to shut down.
Therefore it is always necessary to build an EDMA shutdown function that is called every time you halt the system by brake points or by hand. Due to these severe system crashes, debugging the software becomes a very intensive and slow task.

*Thread level design and function level design must be separated*

Designing at thread level and function level at the same time is not recommended. Function-level design requires so-called linear debugging, which means that the functions are debugged by stopping at breakpoints, to monitor if they work correctly. Combined by viewing the memory and monitoring variables a function can be debugged. When the code functions correctly the function can be included at task level. If errors occur now, it will not be because of the malfunctioning of the low level functions. Note that the thread level debugging has certain shortcomings as stated earlier.