Development of Ultrasound Transducer Characterization Software

Tero Hiltunen

Bachelor's Thesis

Bachelor's Degree in Electronic Engineering

Savonia
University of Applied Sciences

SAVONIA UNIVERSITY OF APPLIED SCIENCES	**THESIS Abstract**

Field of Study Technology, communication and transport			
Degree Programme Degree programme in Electronic Engineering			
Author(s) Tero Hiltunen			
Title of Thesis Development of Ultrasound Transducer Characterization Software			
Date	11 August 2011	Pages/Appendices	41/15
Supervisor(s) Mr. Michael Zapf Mr. Arto Toppinen, Principal Lecturer			
Client Organisation/Partners Karlsruhe Institute of Technology, Karlsruhe, Germany.			

Abstract

The aim of this thesis was to redesign and generalize the program structure of existing ultrasound transducer characterization software. The secondary aim was to develop the functionality of the software and to make the graphical user interface easier and faster for users to use.

This thesis was done for Karlsruhe Institute of Technology (KIT) in Karlsruhe, Germany. KIT makes its own ultrasound transducers which need to be tested and characterized with the measurement system for further development. Ultrasound transducers are built and developed to be used in a project at KIT called USCT (Ultrasound Computer Tomography). USCT focuses on early breast cancer detection.

The ultrasound transducer characterization software was first created in 2005 and has since been extended and modified several times by several persons. Over time the program code got more and more complex and disorganized, partly because of everyone's own kind of style to program. Also the GUI of the software became very complex, due to the amount of settings and parameters able to be defined for the measurement.

To make the software structure generalized and more easily extendable, it was necessary to have a proper concept. For this project, a concept that was used in another project at KIT was selected to be implemented. The concept is called Table driven design and its basic principle is to separate execution of the program from the control logic by placing the program control variables in an external table. Table driven design is intended to generalize and simplify programs.

The result of this thesis was a simplified and generalized program structure by implementing the Table driven design concept. A control table was created to control the flow of the program's execution, and separate functions were made for the logic and execution parts of the software. Along with other upgrades, a help window was created to the GUI to make it easier to use. The GUI became also faster to use, mainly because of a new function for redoing measurements.

Keywords
Ultrasound imaging, tomography, refactoring, table driven design

Confidentiality
Public

SAVONIA-AMMATTIKORKEAKOULU **OPINNÄYTETYÖ Tiivistelmä**

Koulutusala Tekniikan ja liikenteen ala			
Koulutusohjelma Elektroniikan koulutusohjelma			
Työn tekijä(t) Tero Hiltunen			
Työn nimi Ultraäänimuuntimien mittauslaitteiston ohjelmiston kehittäminen			
Päiväys	11.8.2011	Sivumäärä/Liitteet	41/15
Ohjaaja(t) Michael Zapf. Yliopettaja Arto Toppinen.			
Toimeksiantaja/Yhteistyökumppani(t) Karlsruhen Teknillinen Instituutti, Karlsruhe, Saksa			

Tiivistelmä

Tämän opinnäytetyön tavoitteena oli uudelleensuunnitella ja yhtenäistää olemassa olevan ultraäänimuuntimien mittauslaitteiston ohjelmiston rakenne. Tavoitteena oli myös parantaa ohjelmiston toiminnallisuutta ja tehdä graafisesta käyttöliittymästä helppokäyttöisempi.

Opinnäytetyö tehtiin Karlsruhen Teknilliselle Instituutille (KIT) Kalrsruhessa Saksassa. Ultraäänimuuntimien mittauslaitteisto on luotu KITillä tehtyjen ultraäänimuuntimien mittausta ja karakterisointia varten. KITillä tehtyjä ultraäänimuuntimia käytetään rintasyövän havaitsemiseen rintasyövän ollessa vielä varhaisessa kehitysvaiheessa.

Ohjelmisto luotiin alunperin vuonna 2005, jonka jälkeen sitä on paranneltu ja muokattu useaan kertaan useiden henkilöiden toimesta. Ajan myötä ohjelmakoodista tuli todella monimutkainen, osittain myös siitä syystä että jokaisella on omanlainen “käsiala” luodessaan ohjelmakoodia. Myöskin ohjelmiston graafisesta käyttöliittymästä tuli erittäin monimutkainen, johtuen erilaisten asetusten ja parametrien suuresta määrästä.

Että ohjelmiston rakenteesta saatiin yhtenäinen ja helpommin laajennettava, oli löydettävä tarkoitukseen sopiva konsepti. Tätä projektia varten valittiin konsepti, jota käytettiin eräässä toisessa projektissa Karlsruhen Teknillisessä Instituutissa. Konseptin nimi on Table driven design. Sen perusperiaate on erottaa ohjelmiston logiikka ja suoritusosat toisistaan, sekä ohjata ohjelmiston suoritusta taulukolla. Tämän konseptin tavoitteena on yhtenäistää ja yksinkertaistaa ohjelmiston rakennetta.

Tämän opinnäytetyön tuloksena syntyi yhtenäinen ohjelmistorakenne käyttämällä konseptia, jossa ohjelmiston suoritusta ohjataan ns. Ohjaustaulukko. Valittu konsepti täytti sille asetetut kriteerit. Sen avulla ohjelmiston logiikka saatiin erotettua itse suoritusosasta. Graafisesta käyttöliittymästä tuli helpompi ja nopeampi käyttää siihen tehtyjen parannuksien ansiosta.

Avainsanat
Ultraääni, tomografia, ohjelmiston rakenne, taulukkopohjainen

Luottamuksellisuus
Julkinen

Acknowledgements

This thesis was carried out in Germany during spring term 2011 in Karlsruhe Institute of Technology. I would like to thank my supervisors, Mr. Michael Zapf from Karlsruhe Institute of Technology and Mr. Arto Toppinen from Savonia University of Applied Sciences for the opportunity to do this thesis and guiding me through the whole process. I would also like to thank everyone else from KIT, who were somehow involved with the pleasant six months I spent in Karlsruhe doing my thesis.

Tero Hiltunen

11 August 2011
Karlsruhe, Germany

CONTENTS

ABBREVIATIONS

AWG	Arbitrary waveform generator
CVI	C for virtual instrumentation
DAQ	Data acquisition
ETA	Estimated time of arrival
GPIB	General Purpose Interface Bus
GUI	Graphical user interface
IDE	Integrated development environment
IPE	Institute for Data Processing and Electronics
KIT	Karlsruhe Institute of Technology
NI	National Instruments
RS232	Recommended standard 232
TAS	Transducer array system
USCT	Ultrasound computer tomography

1 INTRODUCTION AND MOTIVATION

The goal of this thesis was to redesign and generalize the structure of existing ultrasound transducer characterization software. The software controls a measuring station (Figure 1), which was created in 2005 at Karlsruhe Institute of Technology (KIT) as a diploma thesis by Lars Petzold [1]. The measuring station was created to test and to prove the quality of the ultrasound transducer made at KIT for Ultrasound Computer Tomography (USCT) project. With the measuring station it is possible to evaluate and characterize ultrasound transducers for further development.

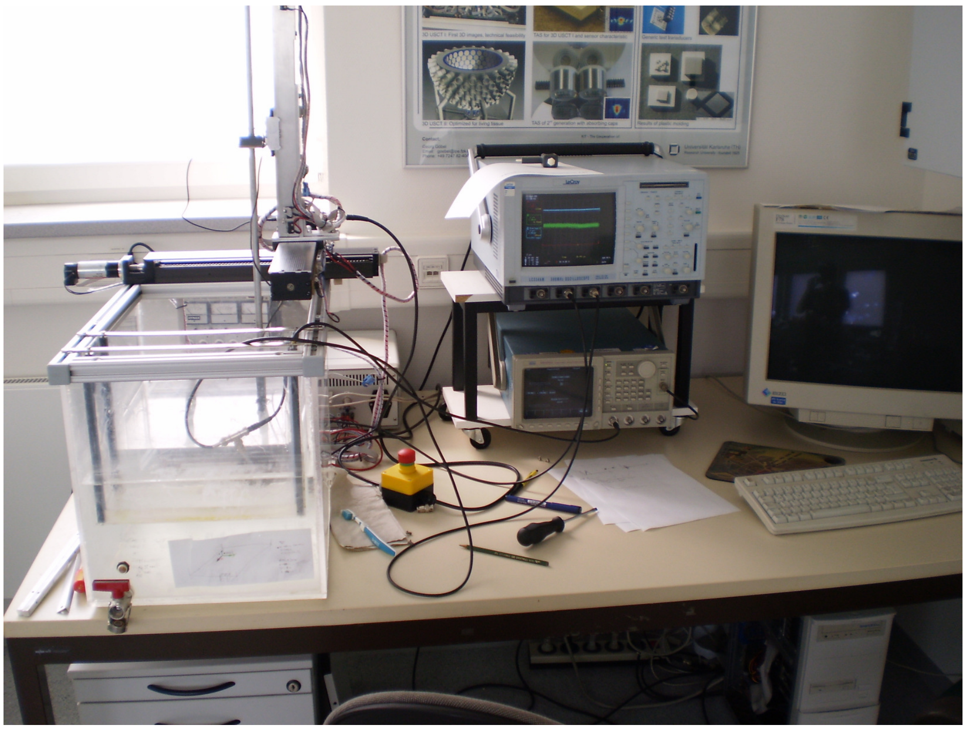


Figure 1. Measurement water tank on the left, oscilloscope and arbitrary waveform generator in the middle-top and control PC on the right.

1.1 USCT

USCT is a project at KIT focused on early breast cancer detection. The aim of the project is to detect tumors in a woman's breast at early stage, when the average diameter of tumor is less than 5mm. The USCT promises high resolution three dimensional pictures of the breast and it is based on thousands of transducers in a water filled examination reservoir (Figure 2). [2]

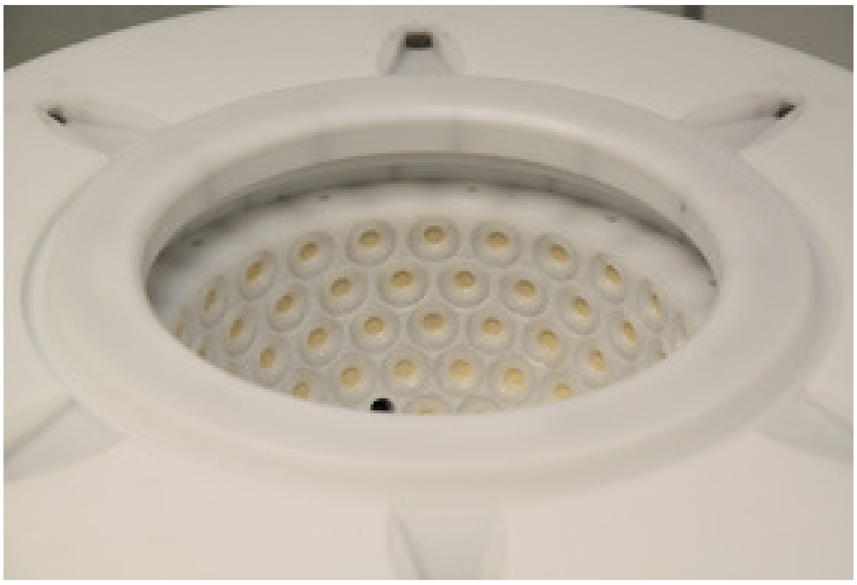


Figure 2. Examination reservoir with mounted transducer arrays filled with water as medium. [3]

The idea of the USCT is that all the transducers in the measurement container emit ultrasound one at the time, while other transducers are receiving the ultrasound waves. [2] Tumor tissue is typically denser than the fatty and glandular breast tissue. When the ultrasound waves interact with the dense tumor tissue, they are reflected and scattered. [4] The ultrasound emitted by one transducer is then received by all the other transducers and converted into electrical signals. [8] These signals are captured with data acquisition (DAQ) hardware into measurement data. From the data gathered with transducers and DAQ hardware, it is possible to locate possible tumors inside women's breast. The data is displayed and analyzed as 3D images on a PC (Figure 3).

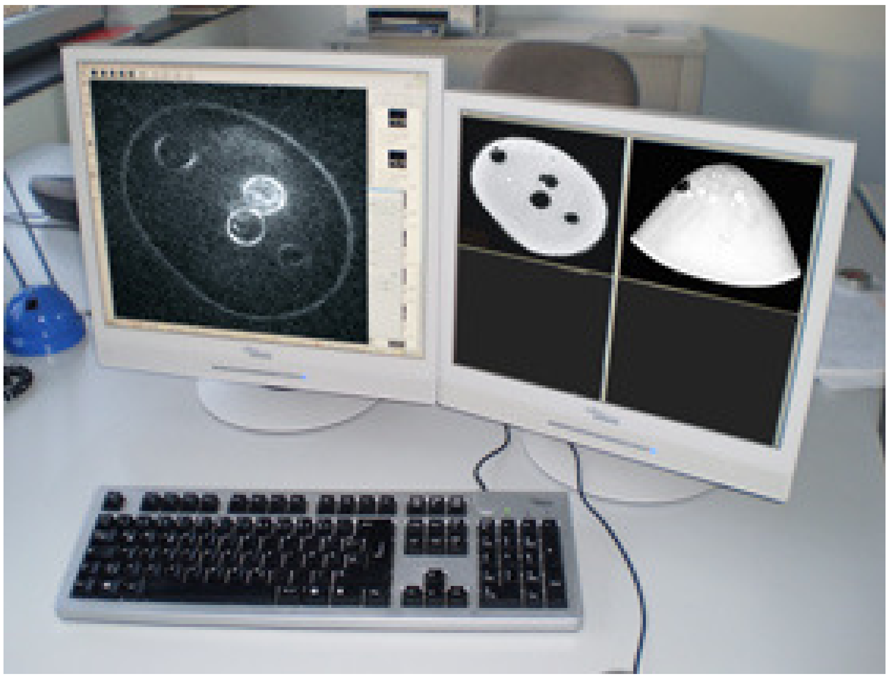


Figure 3. USCT reflection image on the left screen and on right screen a MRT image of the same breast phantom for comparison. [3]

The examination reservoir is mounted in a patient bed (Figure 4), which also has DAQ hardware installed. The whole process of creating and developing transducers for USCT project is done at KIT.

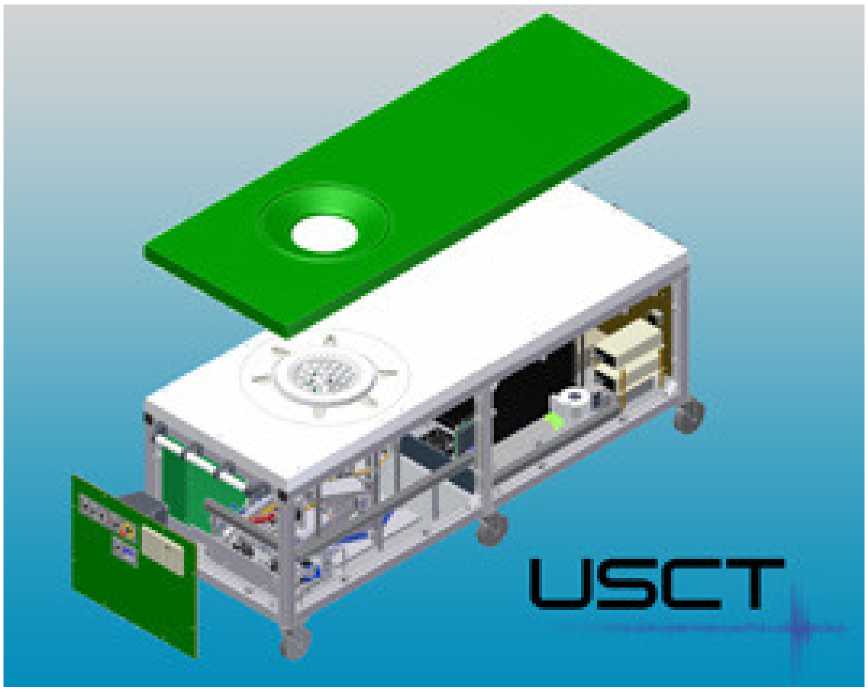


Figure 4. Patient bed with built-in examination reservoir and DAQ hardware. [3]

1.2 Measuring Station Hardware and Interfaces

The ultrasound transducer measuring station is a complex system able to characterize ultrasound transducers and hydrophones over several parameters. The ultrasound transducers and hydrophones can be moved around the measurement water tank with a three axis hydrophone/TAS holder. The excitation pulse from arbitrary waveform generator (AWG) (figure 5) for transducers can be parameterized by bandwidth, frequency, voltage, form, type, code and length.

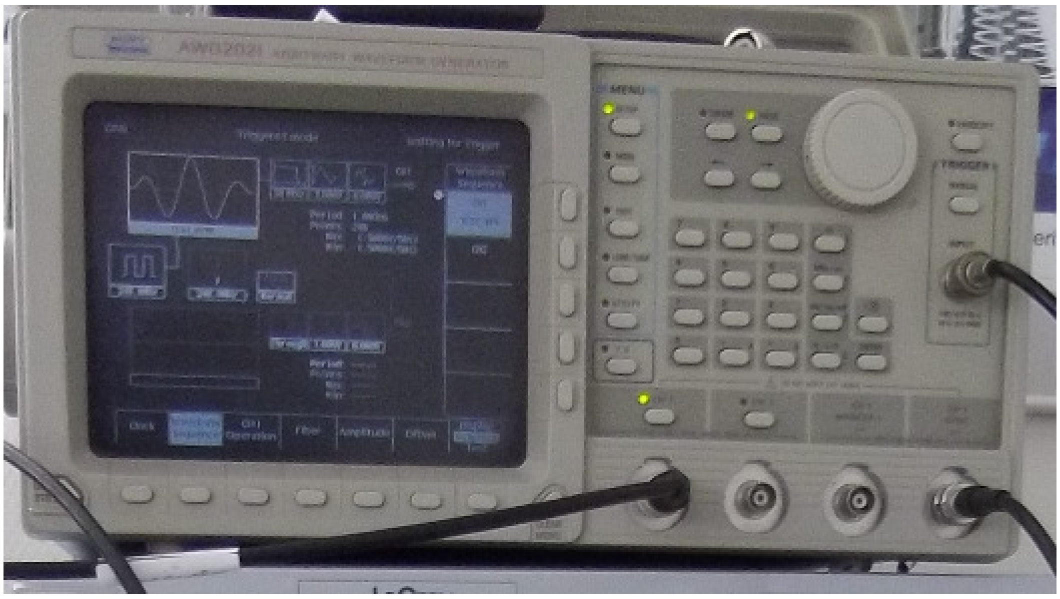


Figure 5. Tektronix AWG2021 arbitrary waveform generator. Information of the parameterized signal on the screen.

The excitation pulse and output signal of the hydrophone/TAS can be observed for debugging purposes with an oscilloscope. Figure 6 shows an example of a CE excitation pulse. On the y axis is the amplitude of the signal and on the x axis is the bandwidth.

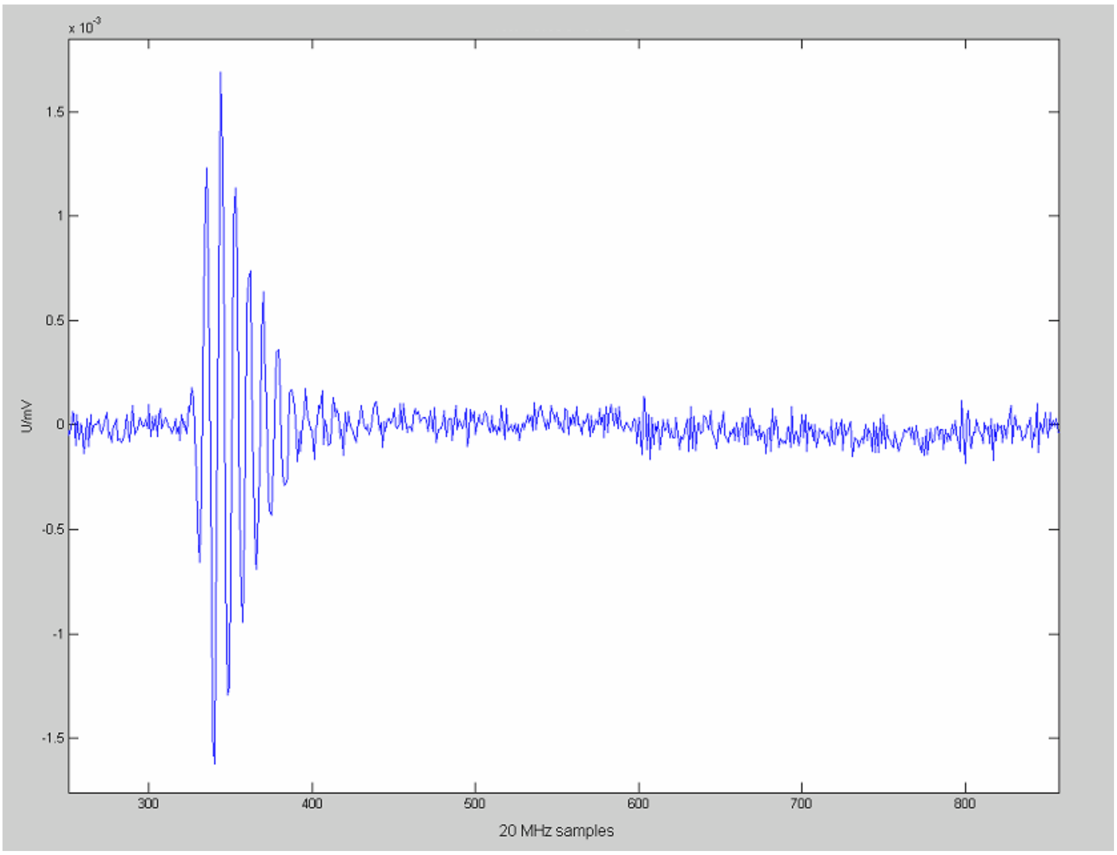


Figure 6. CE signal used for excitation. The figure shows amplitude over time.

Data is acquired with a PC DAQ card. The used receivers and emitters can be selected if the transducer is an array consisting of multiple sub transducers (Figure 7). It is also possible to do sweeps over some parameter, e.g. voltage sweep so that the measurement is done over certain voltage range and increment selected by the user.

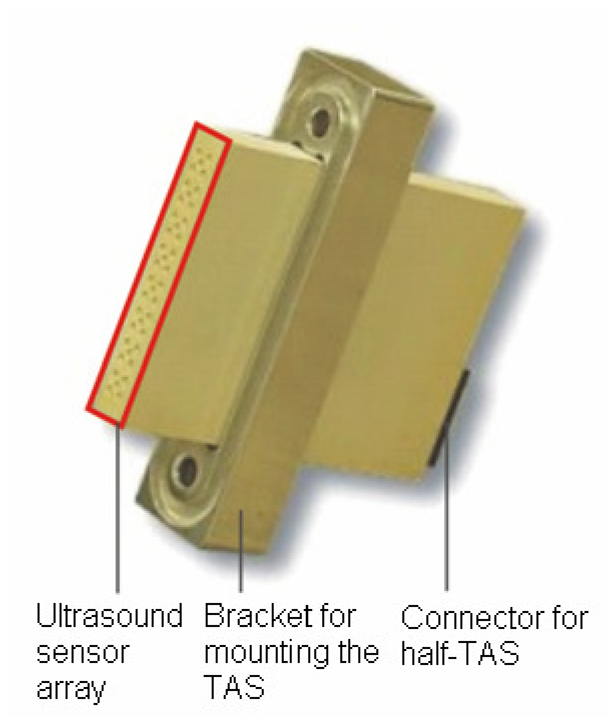


Figure 7. Transducer array system. [1]

The ultrasound measuring station is controlled by a PC, which is also used for developing the software. The PC controls the three axis hydrophone/TAS holder via two RS232 connections. The AWG is connected to the PC via one GPIB connection. In addition, there is one RS232 for accessing the embedded microcontroller in the transducer. In figure 8 is a block diagram of the measurement station hardware.

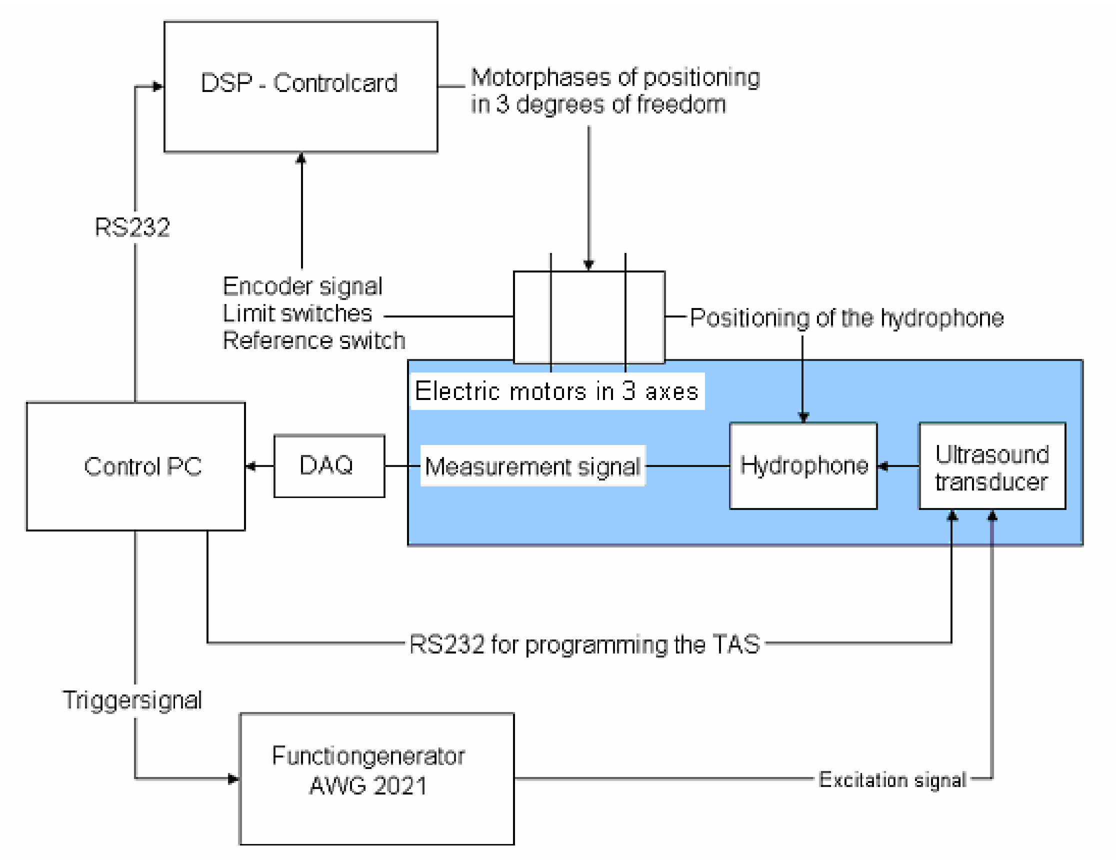

DSP - Controlcard
Motorphases of positioning in 3 degrees of freedom
RS232
Encoder signal
Limit switches
Reference switch
Positioning of the hydrophone
Electric motors in 3 axes
Control PC
DAQ
Measurement signal
Hydrophone
Ultrasound transducer
RS232 for programming the TAS
Triggersignal
Functiongenerator AWG 2021
Excitation signal

Figure 8. Block diagram of the measurement station hardware. [1]

The list below shows the list of parts of the measuring station, which can also be seen in Figure 9:

- Control PC for DAQ and developing the software
- Tektronix AWG 2021 for generating the excitation signal and triggering
- LeCroy LC334AM oscilloscope for monitoring input and output signals, also for debugging
- Three axis hydrophone/TAS holder with a mounted on transducer and ONDA HNC 0400 hydrophone, resolution 1mm.
- Three different transducer sockets with three different kind of transducers
- Water tank, dimensions: 51.6cm x 35.3cm x 38.5cm.

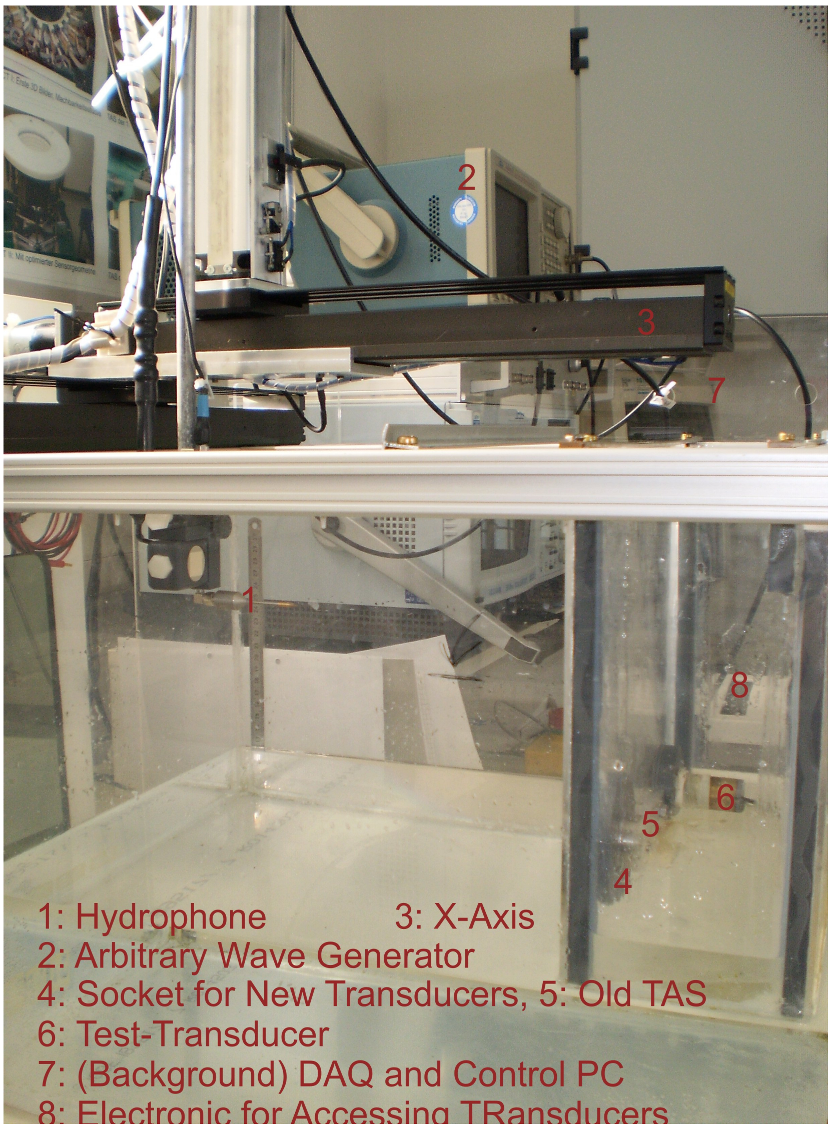

2
3
7
1
8
6
5
4
1: Hydrophone
3: X-Axis
2: Arbitrary Wave Generator
4: Socket for New Transducers, 5: Old TAS
6: Test-Transducer
7: (Background) DAQ and Control PC
8: Electronic for Accessing TRansducers

Figure 9. Parts of the measurement station.

1.2.1 Ultrasound Transducer

A transducer, in general, is a device that converts one type of energy to another. The transducers used in the USCT project are ultrasound transducers. Ultrasound transducers convert sound pressure into electrical signals. [4] Figure 10 shows a picture of an ultrasound transducer built at KIT for USCT project.

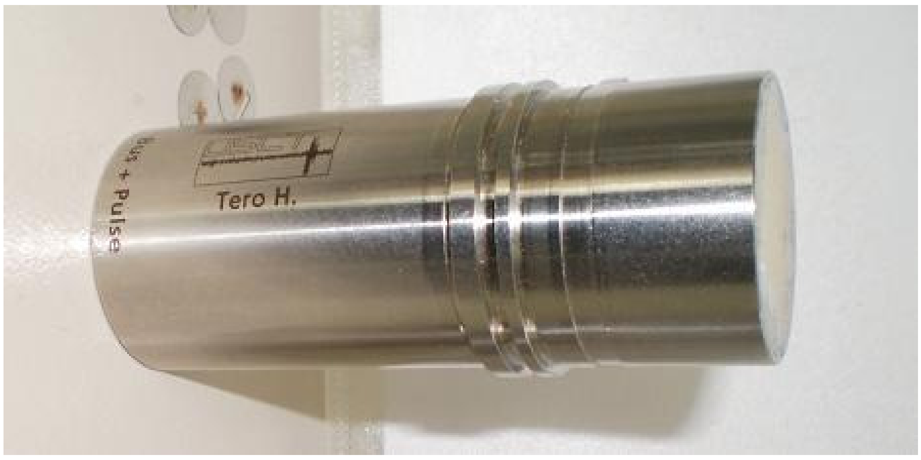


Figure 10. An ultrasound transducer built at KIT.

1.2.2 Hydrophone

A hydrophone is a device for detecting changes in pressure underwater. A hydrophone converts the underwater sound i.e. acoustic energy into electrical energy. A hydrophone can only be used for listening unlike a transducer, which can also be used for sending. Figure 11 shows a picture of the Onda HNC 0400 hydrophone used for testing the transducers built for the USCT project. In addition to the hydrophone there is also a transducer in the picture. They both are attached to the XYZ holder. [5] The XYZ holder has three axes to move the hydrophones and a transducer around the measurement water tank. With the XYZ holder it is possible to do measurements with various kinds of moving patterns, e.g. semicircle.

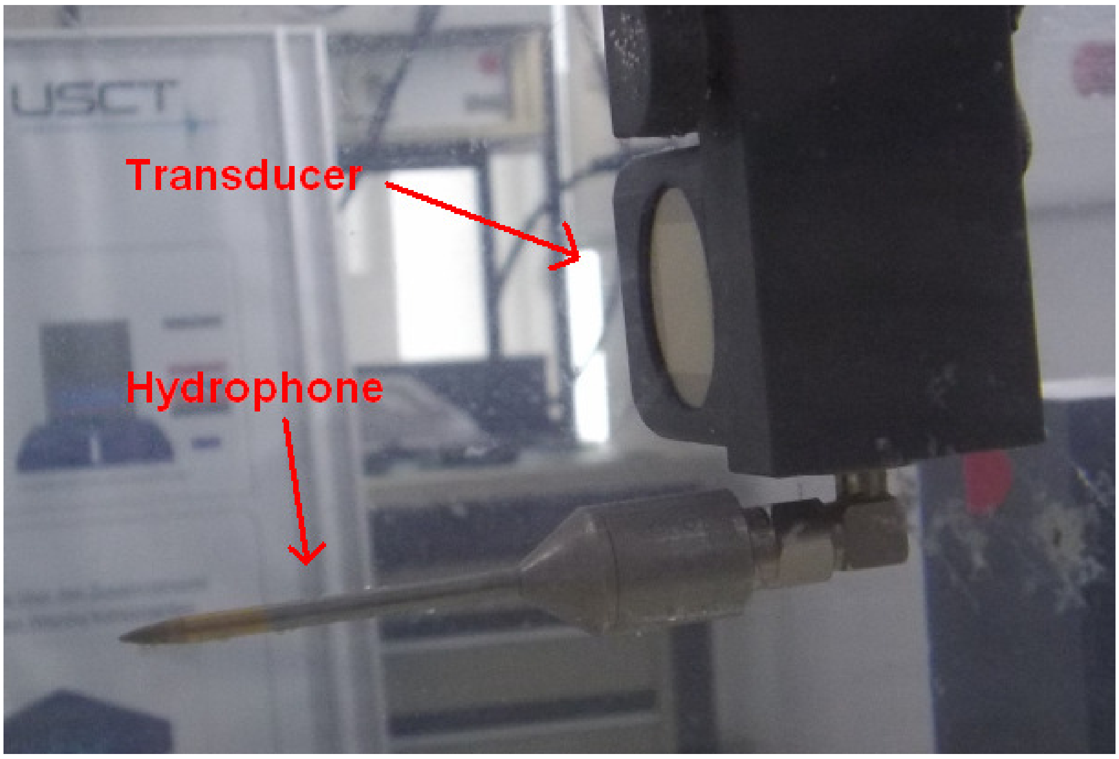


Figure 11. Onda HNC 400 hydrophone and a test transducer attached to the XYZ holder (not completely visible) inside the water tank.

1.3 Measuring Station Software

Over time the measuring station and its software has been extended several times by several persons. Therefore, the measuring station and especially its software became very complex and complicated. Part of the reason that the software structure became so complicated is that everyone has their own kind of way to program and there was not any particular concept defined to follow. For further development of the system, it was necessary to redesign and generalize the structure of the program. To do that, it was important to find a fitting concept for the program structure.

A big part of doing this thesis was to get acquainted with the existing program code. Not only that the program was so vast and complex, it also was mostly commented in German, which I, unfortunately, haven't studied at all. But with the help of multilingual colleagues and dictionaries provided by internet, there were no problems in translations. Also to ease the way through the introduction phase, some smaller upgrades were made to the program code and to GUI to get acquainted with the program code and its structure.

TortoiseSVN was used in this project for software versioning and revision control. These kinds of subversion software are used to maintain current and old versions of source files and documentation. TortoiseSVN is free software released under the GNU General Public Licence (GNU GPL). [10]

1.3.1 LabWindows/CVI 6.0

The measuring station software is written in ANSI C in *LabWindows/CVI 6.0*, which is an IDE developed by *National Instruments*. *LabWindows/CVI* provides a comprehensive set of tools for creating test and control applications. It also provides fast and easy to use tools for creating graphical user interfaces.

LabWindows/CVI has a user interface editor to create graphical user interfaces. A GUI can consist of panels, command buttons, pull-down menus, graphs, strip charts, knobs, meters, and other controls and indicators. To create a GUI with the user interface editor, it is not necessary to write a single line of code. But it is also possible to create a GUI programmatically by using function calls. The created GUIs are stored in user interface resource files (_*.uir_). [6]

The connection between GUI and software code is done with *Callback* functions. Different kinds of actions by the user lead into events, which lead into actions that are determined in the corresponding *Callback* function. For example, if the user clicks the “Start Measurement” button, the software starts to execute the code inside the corresponding *Callback* function, in this case *CVICALLBACK StartAutoMeasurement();* function. [6]

1.3.2 Software Structure

At the beginning of the work the software structure was very complex and consisted of many convoluted loops inside each other. It was an outcome of years of development by several persons and of the lack of an appointed structure to follow. The worst problems were at the main measurement loop, where the flow of the program was controlled by many functions and if –else constructs inside each other.

Figure 12 illustrates the structure of the programs main measurement loop at the stage it was at the beginning of this thesis. The sequence diagram was made with free online software called WebSequenceDiagrams [9]. The aim was to keep the diagram as simple as possible, but still bring out the convoluted loop structure. From the diagram we can see that there where different functions for different measurement cases, e.g. whole TAS measurement for measuring all the senders and receivers in a TAS or individual sender/receiver selection measurement, where single sender or receiver can be selected for measurement. There is also some code duplication perceptible from the diagram, e.g. "*Read values from GUI*".

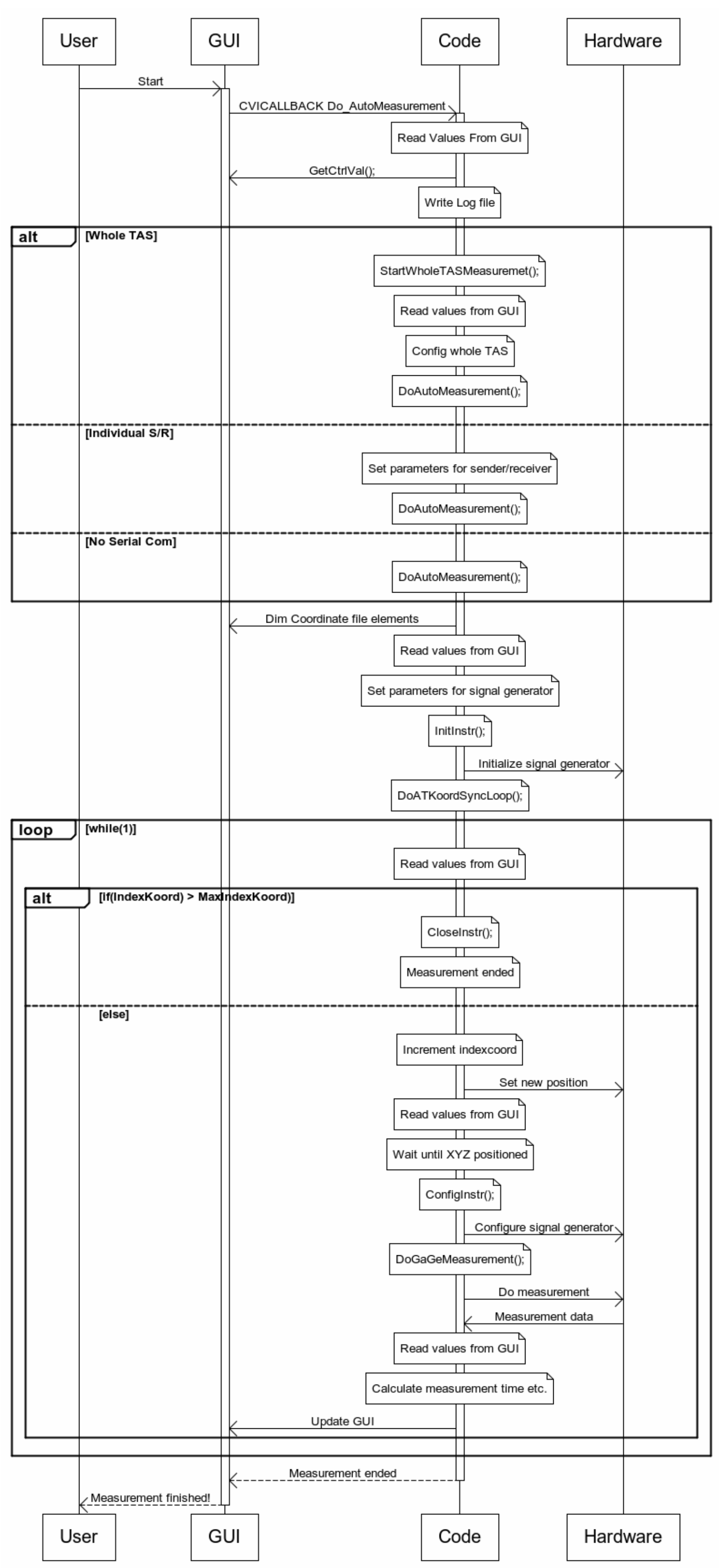
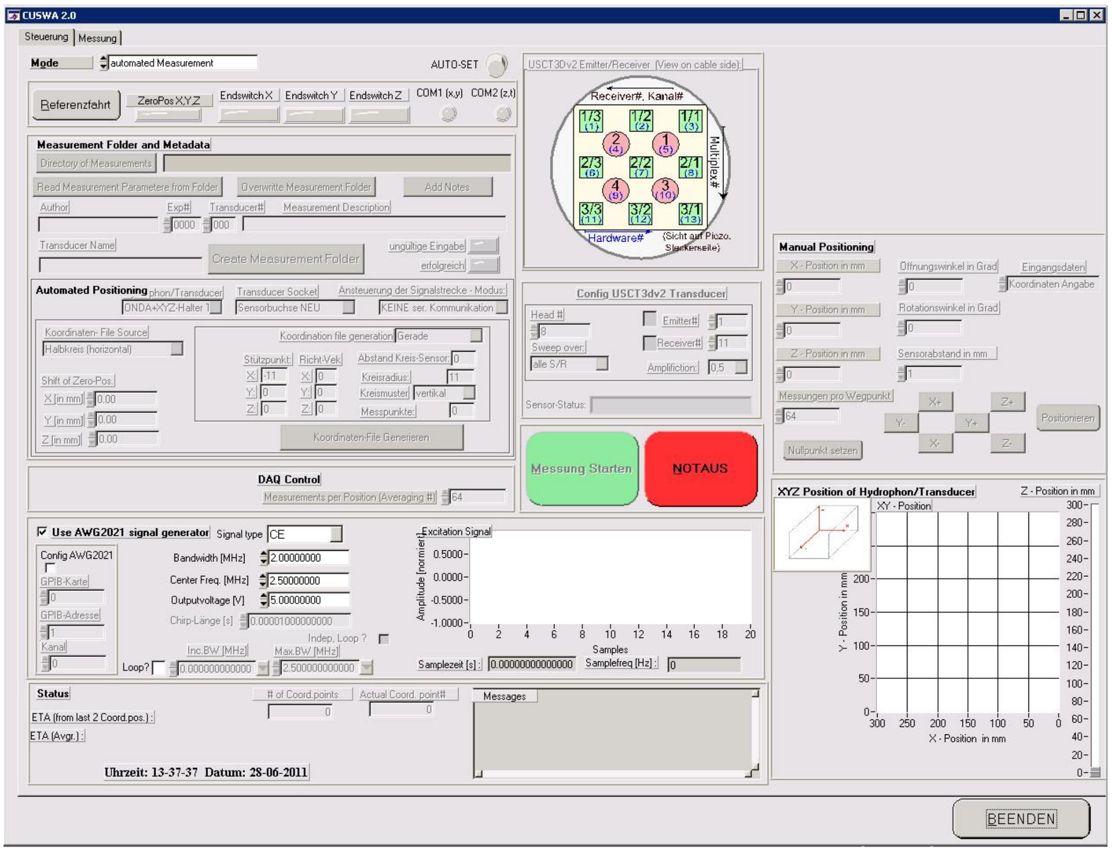


Figure 12. Sequence diagram of the program structure in the main measurement loop as it was in the beginning of the work. Y axis represents time and X axis parts of the system.

1.3.3 GUI

In software to control this kind of complex measuring system, the GUI (Figure 13) is often very complex too. One goal of this thesis was to make the GUI easier and faster to use. At the beginning of this work, it was difficult for new users to use the software without instructions on a separate paper. The settings were placed quite systematically on the GUI, so that the settings were supposed to be set in order from up to down. But the user could set the settings in almost any order or leave some important settings unset, which caused problems. Also some functions were unfinished, e.g. the "*Read measurement Parameter from folder*" button, which would be very helpful for redoing measurements.


CUSWA 2.0

Steuerung | Messung

Mode: automated Measurement

Measurement Folder and Metadata
Exp#: 0000
Transducer#: 000

Automated Positioning
Hydrophon/Transducer: ONDA+XYZ-Halter 1
Transducer Socket: Sensorbuchse NEU
Ansteuerung der Signalstrecke - Modus: KEINE ser. Kommunikation
Koordinaten- File Source: Halbkreis (horizontal)
Shift of Zero-Pos.: X [in mm] 0.00; Y [in mm] 0.00; Z [in mm] 0.00
Koordination file generation: Gerade
Stützpunkt: X: -11; Y: 0; Z: 0
Richt-Vek: X: 0; Y: 0; Z: 0
Abstand Kreis-Sensor: 0
Kreisradius: 11
Kreismuster: vertikal
Messpunkte: 0

DAQ Control
Measurements per Position (Averaging #): 64

☑ Use AWG2021 signal generator
Signal type: CE
Config AWG2021: ☐
GPIB-Karte: 0
GPIB-Adresse: 1
Kanal: 0
Bandwidth [MHz]: 2.00000000
Center Freq. [MHz]: 2.50000000
Outputvoltage [V]: 5.00000000
Chirp-Länge [s]: 0.00001000000000
Indep. Loop ? ☐
Loop? ☐
Inc.BW [MHz]: 0.000000000000
Max.BW [MHz]: 2.500000000000
Samplezeit [s]: 0.00000000000000
Samplefreq [Hz]: 0

Status
ETA (from last 2 Coord.pos.):
ETA (Avgr.):
# of Coord.points: 0
Actual Coord. point#: 0
Messages
Uhrzeit: 13-37-37 Datum: 28-06-2011

Config USCT3dv2 Transducer
Head #: 8
Sweep over: alle S/R
Emitter#: 1
Receiver#: 11
Amplifiction: 0,5
Sensor-Status:

Manual Positioning
X - Position in mm: 0
Y - Position in mm: 0
Z - Position in mm: 0
Öffnungswinkel in Grad: 0
Rotationswinkel in Grad: 0
Sensorabstand in mm: 1
Eingangsdaten: Koordinaten Angabe
Messungen pro Wegpunkt: 64

Figure 13. GUI of the measurement station software in the beginning of the work.

2 CONCEPT DEFINITION

To redesign and generalize the program structure of existing software, it was necessary to find a fitting concept. Requirement for the concept was to be simple and easily extendable. When selecting the concept, a concept used in another project at KIT rose up. The idea was to control the flow of the program with tables. Based on that idea, I started to do further research of table controlled designs.

Table driven design itself is a very old concept. Originally it was created and implemented over 50 years ago. The basic idea of table drive design is that the flow of the program is encoded in a table, so that the control logic is kept separate from the execution of the program. The control table contains the variables which somehow affect to the flow of the program. Each row of the table is then interpreted with a given routine, in a processing loop. Figure 14 illustrates the use of control table in an application. [7]

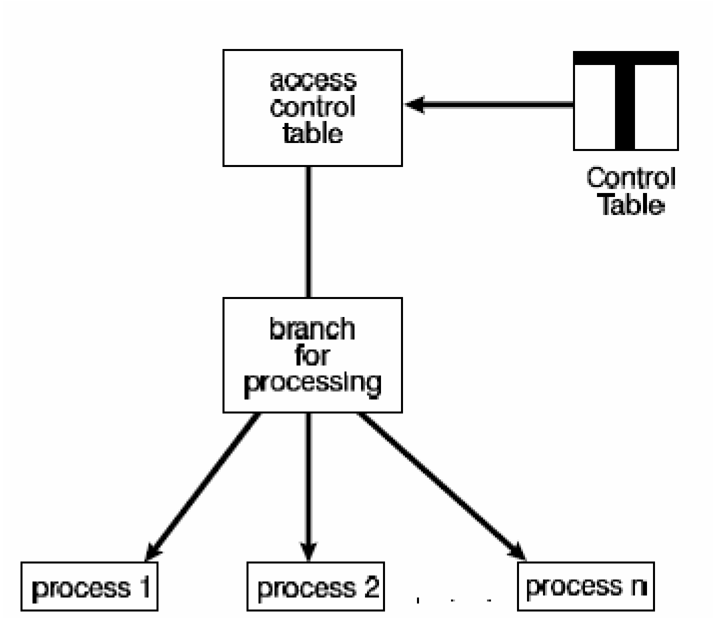


Figure 14. Generalized expression of usage of a control table in an application. [7]

3 IMPLEMENTATION

This chapter discusses the implementation of the selected concept, improvements to the software and the upgrades done to the GUI of the measurement station software.

3.1 Implementing the Selected Concept

The implementation of the defined concept started by radically modifying the program structure. The idea of the table driven design is to separate the control logic from the execution of the program. This was implemented by creating *measurementPreparation();* function for control logic and *Do_Measurement_Loop();* function for execution. Some parts of the control logic are located also in other parts of the program code, when the user sets the parameters and settings for the measurement. The *measurementPreparation();* function contains the actions that needs to and can be done before the main measurement function. These actions are initializations and definitions for the measurement.

The *measurementPreparation();* function is executed after the user has pressed the “Start measurement” button and the execution of corresponding *CALLBACK* function has started. The CALLBACK function for “Start measurement” button is called *StartAutoMeasurement();*. The *StartAutoMeasurement();* function contains also the main measurement loop function, the dim input function, read functions for the user interface variables and writing of log files. The program code of the *StartAutoMeasurement(); CALLBACK* function is presented in Appendix 6 and the program code of measurement preparation is presented in Appendix 2.

The whole execution of the software is done in the *Do_Measurement_Loop();* fuction. The program code of the main measurement loop is presented in Appendix 1. *Do_Measurement_Loop();* function contains three major function in addition to doing the measurement itself. These three functions are listed and explained in a list below.

- ConfigWholeTAS();
    - Configurations for the whole TAS measurement inside the main measurement loop. Program code presented in Appendix 5.
- SetNewPosition();
    - Function for setting the new position of the XYZ holder. Program code is presented in Appendix 4.

- CalcMeasTime();
    - Function for calculating measurement times. Program code presented in Appendix 3.

The control table was included to the software code after the structure was simple, generalized and partly functional. The full functionality of the program was tested and fixed in the results evaluation phase of this thesis.

The control table is modified as user sets setting and parameters for measurement, before the main measurement loop. When the measurement is running, the only variables which changes their values are the index number for the measurement point (*glob.CT[INDEXCOORD]*) and the variable which determines the sweep over senders and receivers (*glob.CT[MEASSR]*). The index variable for the actual measurement point goes from 0 to the maximum number measurement point, e.g. 0 to 50. The variable for sweeping over senders and receivers changes its value if the measurement is over all senders and receivers. Then its value is two at the beginning, and after all senders are measured its value is changed into one, to get all the receivers measured. All other variables remain their values throughout the measurement loop.

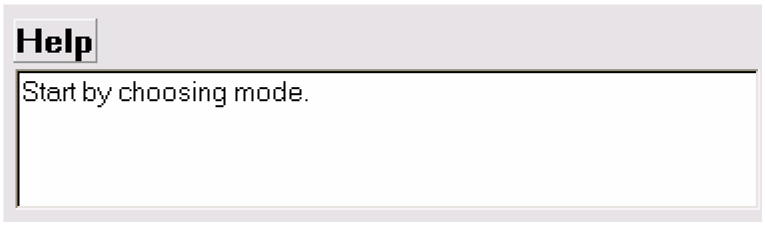
The contents of the control table are shown below as a citation of the program code. The variables are defined as constants, which are then used to indicate the index number for the corresponding variable. Below is also explained the possible values of the control table variables.

```
// Index numbers for the control table CT[] -Tero
//------Constant--Index#---Values-------------------------------------
#define INDEXCOORD 0 // actual coord. point during measurement
#define MEASMODE   1 // 0=whole TAS, 1=individual S/R, 2=no ser. Com.
#define MOVEMENT   2 // 0= /automeander, 1=drive by file/auto koord
#define SIGNALGEN  3 // 0=off, 1=on
#define LOOPTYPE   4 // 0=BW, 1=MF, 2=chirplength, 3=voltage
#define MINILOOP   5 // 0=off, 1=on
#define MEASSR     6 // sweep over 0=sender, 1=receiver, 2=all s/r
// End of control table constants -----------------------------------
```

3.2 Upgrades to the Software and GUI

During the introduction phase, many upgrades were made to the software and its GUI to make it easier and faster to use. Also during implementation phase the functionality of the GUI was improved to achieve a user interface which is fast and easy to use.

To get rid of the instructions paper for new users, a help window (Figure 15) was created. The help window is located in the upper right corner of the GUI. It tells the user what to do next in order to get a measurement running. The help window has 14 different instructions to help the user in different situations. Figure 14 shows the default value of the help window, which is shown at the startup of the program.



Start by choosing mode.

Figure 15. Help window gives instructions to the user.

In addition to help window, dim and undim functions were created to make it easier to set all settings and parameters in right order. Earlier the user could basically set the settings in any order he wanted or it was possible to run the measurement without setting all required settings, and that caused some problems in execution of the measurement loop. With the dim and undim functions it was possible to control which parts of the user interface are dimmed and which are not. Usually, only the section of the GUI is undimmed that the user needs to be able to modify. During measurement, all the sections are dimmed, so that the user can't modify any settings. Only the STOP button is undimmed, so that the user can stop the measurement in case of emergency.

One crucial upgrade to the GUI was the "*Read Measurement Parameter from Folder*" button. It did already exist when starting this thesis, but the functionality was not yet implemented. The idea of that button is that the user can redo any measurement he/she has earlier parameterized with just few clicks. The user only needs to select the measurement he/she wants to redo, and all the settings and parameters are filled automatically just like they were at the first time. The functionality of the "*Read Measurement Parameter from Folder*" button is implemented by writing all the settings and parameters into **.txt* file when the user has set all the settings and parameters. Those settings are then read from the file and set into the GUI with *SetCtrlVal();* functions.

It is important for user to know how the measurement is progressing and how long the measurement is going to take. For this purpose is the "Status" section (Figure 16) in the lower left corner of the GUI. The "Status" section has progress bar, time per last coordinate point, average time per coordinate point and estimated time left to tell the user how the measurement is progressing. The total number of measurement points and the current measurement point are also shown.

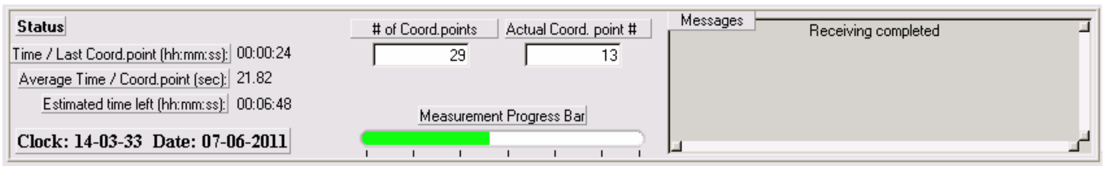
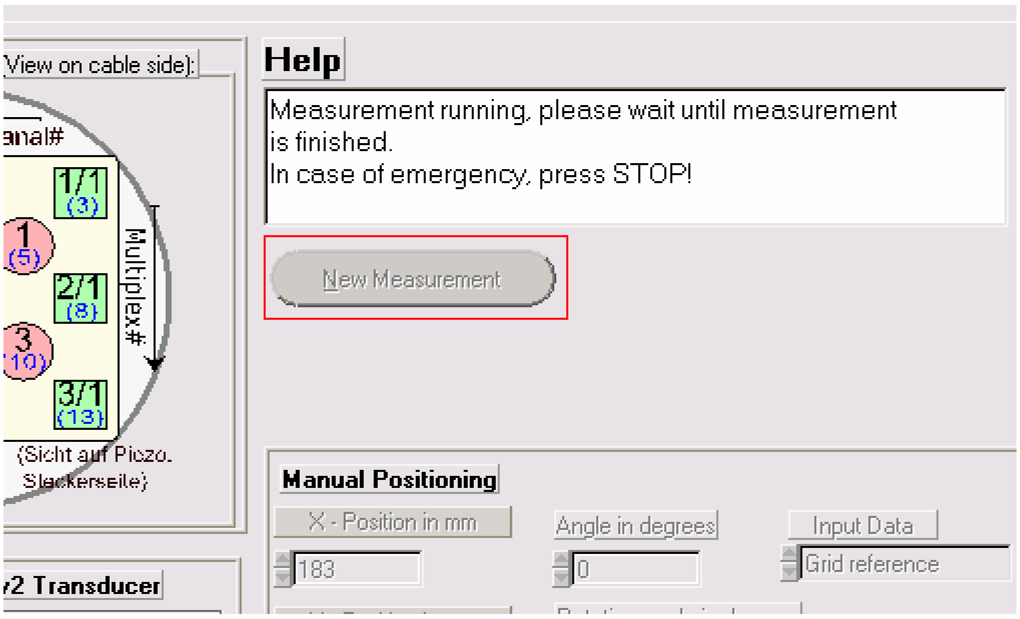

Status
Time / Last Coord.point (hh:mm:ss): 00:00:24
Average Time / Coord.point (sec): 21.82
Estimated time left (hh:mm:ss): 00:06:48
Clock: 14-03-33 Date: 07-06-2011
# of Coord.points: 29
Actual Coord. point #: 13
Measurement Progress Bar
Messages: Receiving completed

Figure 16. Status section in the GUI.

In the upper right corner of the GUI right under the help window is the "New measurement" button (Figure 17). The "*New measurement*" button becomes handy if the user wants to do many measurements in a row. Earlier it was not possible to do many measurements in a row without shutting down the software between measurements, but with the "*New Measurement*" button it is now possible. The "New measurement" button basically only initializes the GUI into the default values it had when running the software for the first time. Also when doing multiple measurements in a row, it is not necessary to do the reference positioning for the XYZ holder because the position of the holder is still in the memory of the microcontrollers.



Help
Measurement running, please wait until measurement is finished.
In case of emergency, press STOP!

Manual Positioning
X - Position in mm: 183
Angle in degrees: 0
Input Data: Grid reference

Figure 17. New measurement button below the help window.

At the beginning of the work the GUI and the software were partially in English and partially in German. To make the software generalized by language, the GUI was fully translated into English and most parts of the program code also. The translation was done with the help of dictionaries provided by internet and with the help of multilingual colleagues.

One way to make the software to be less error prone was to minimize the usage of GUI variables. The idea of minimizing the usage of GUI variables was to remove code duplication and get rid of heavy usage of *GetCtrlVal()* functions. This means that reading values from GUI into variables was reduced to its minimum, so that every value is read only once from GUI. To make this kind of change it was necessary to define global variables for the values. The global variables were placed in a struct named “glob”. This way the naming was proper and it was easy to notice that they are global variables, e.g. *glob.ReceiverNr*, which is a variable for the number of selected receiver.

4 RESULTS AND EVALUATION

This chapter discusses the results and evaluation of this thesis after the software structure was generalized and the GUI improved.

4.1 Software Structure

The main measurement loop was the part of the program which was most in need of redesigning. The old structure had multiple separate loops and functions for different measurement cases. There were separate functions for whole TAS measurement, individual sender/receiver selection and also for a measurement without serial communication. But in the new one, there is only one “main” measurement loop to cover all measurement cases. The number of hierarchical levels was reduced from five in the old to two in the new. Also, quite a lot of code duplication was removed, by minimizing the reading of the GUI variables to its minimum in a consistent and conflict free manner.

Figure 18 shows a sequence diagram of the old main measurement loop software structure, and Figure 19 shows a sequence diagram of the new structure. By comparing these two pretty simple diagrams, it is possible to see how much simpler the new structure is.

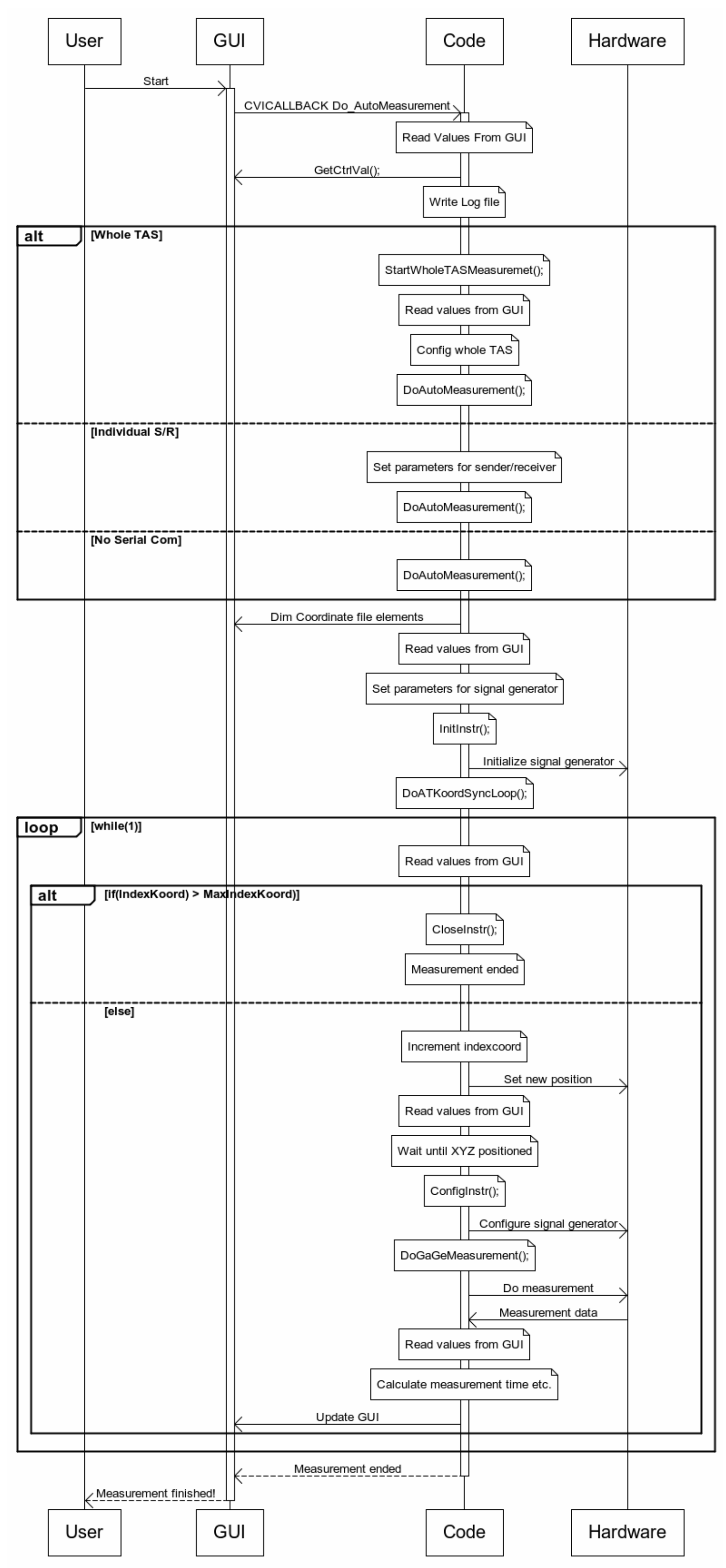


Figure 18. Sequence diagram of the old main measurement loop structure. Y axis represents time and X axis parts of the system.

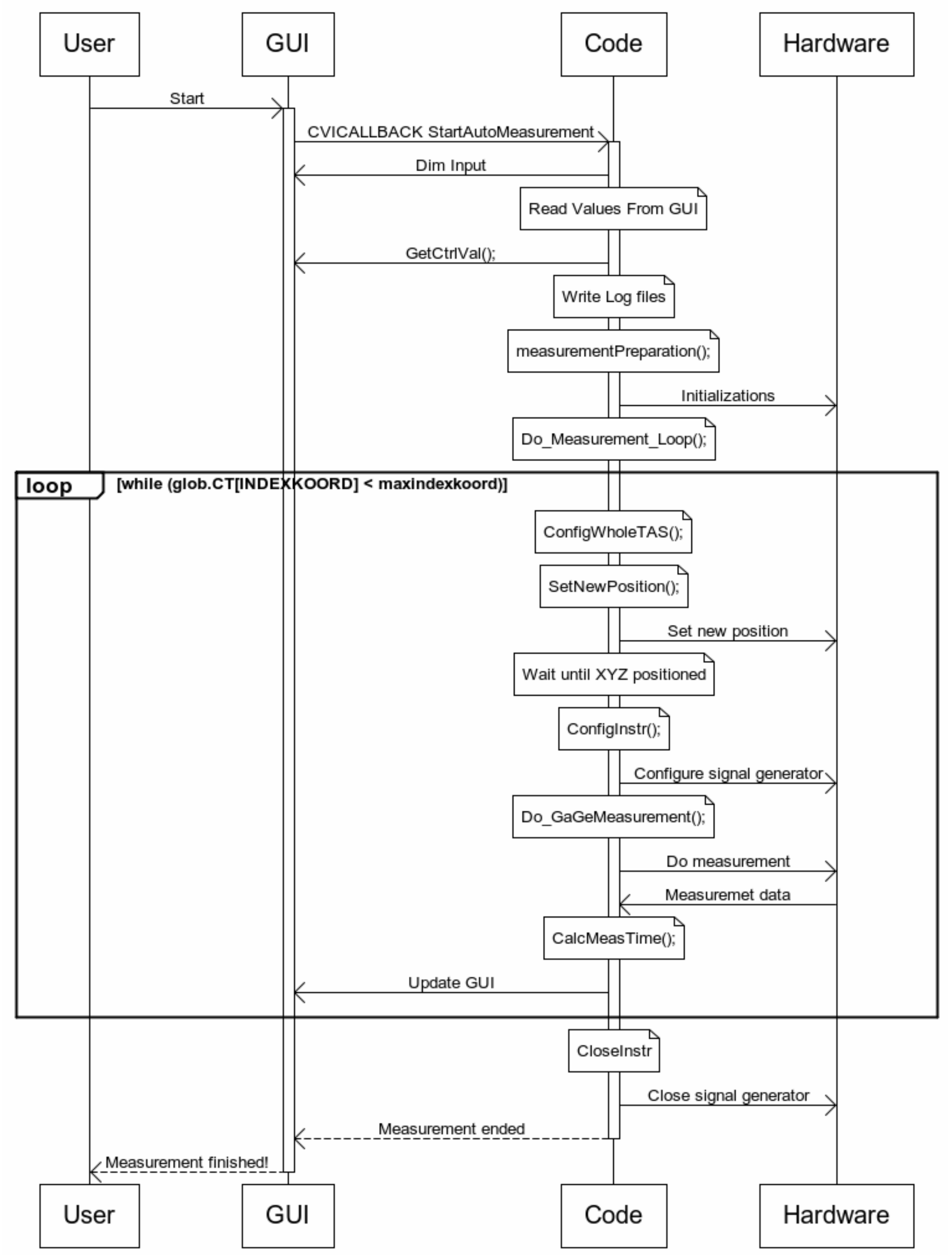


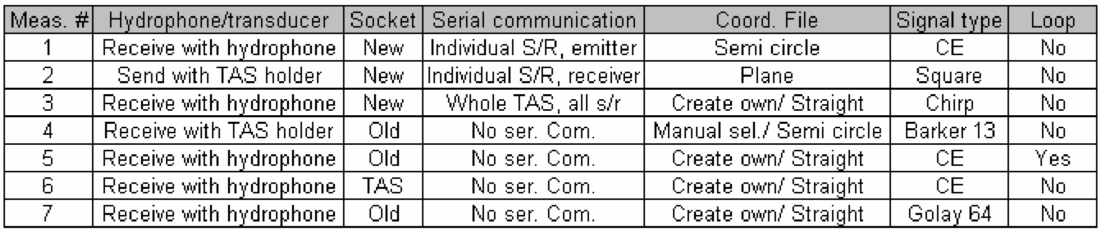
Figure 19. Sequence diagram of the new, generalized main measurement loop structure. Y axis represents time and X axis parts of the system.

4.2 Test Measurements

To be sure that the measurement system and its software still worked properly after such a big modification in the software structure, it was necessary to do test measurements. Test measurements were done at least once with every possible setting and parameter. To cover all different settings and parameters, total of seven measurements were done.

From the data of the first measurement, it was noticeable that the positioning of the XYZ holder was not correct. The holder position was few millimeters off in all x, y and z direction. First the positioning was tried to fix by checking the calculations of the positioning from the program code, but everything seemed to be correct. After this, the only way to get the positioning correct was to re-measure and calibrate the water tank, the positions of the transducer sockets, dimensions of the hydrophone holder and safety margins. After all measures and dimensions were re-measured, some differences were found when comparing to the old measures. From the data of next test measurement, the positioning was found to be correct.

All measurements and the used settings are listed in Figure 20. First column in the table is the number of the measurement, second is the used hydrophone or transducer. In column 3 is the socket which was used. In the fourth column is determined whether serial communications was used or not. Fifth column is the selected coordinate file for the measurement, and in the next column is shown the used signal type. The last column tells whether the measurement was looping over some parameter or not. While doing different measurements, some modifications were done to the program code to get the measurements running properly. This was an anticipated procedure, because some measurement cases were done for the first time by me. After all, everything went better than expected.



Meas. #	Hydrophone/transducer	Socket	Serial communication	Coord. File	Signal type	Loop
1	Receive with hydrophone	New	Individual S/R, emitter	Semi circle	CE	No
2	Send with TAS holder	New	Individual S/R, receiver	Plane	Square	No
3	Receive with hydrophone	New	Whole TAS, all s/r	Create own/ Straight	Chirp	No
4	Receive with TAS holder	Old	No ser. Com.	Manual sel./ Semi circle	Barker 13	No
5	Receive with hydrophone	Old	No ser. Com.	Create own/ Straight	CE	Yes
6	Receive with hydrophone	TAS	No ser. Com.	Create own/ Straight	CE	No
7	Receive with hydrophone	Old	No ser. Com.	Create own/ Straight	Golay 64	No

Figure 20. Settings and parameters used in test measurements.

In figure 21 is data of measurement one, which was a semicircle measurement with individual emitter selected. The radius of the semicircle was set to 2 cm in the coordi2nation file, which also can be calculated from the figure. In the figure the red color indicates that the amplitude of the received signal was strong. Other colors indicate weaker amplitudes, dark blue being the weakest. On the X axis are the numbers of measurements, in this case from 0 to 29. Y axis is time.

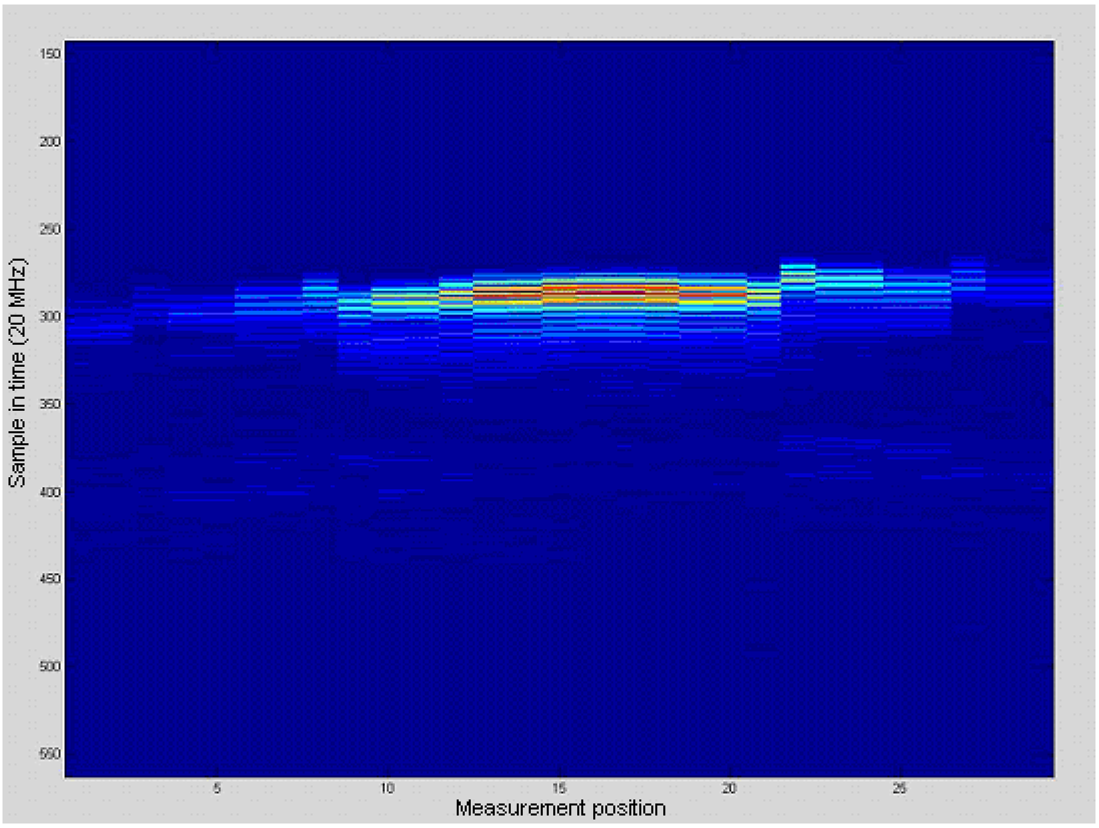


Figure 21. Data from a semicircle measurement. X axis is the measurement number and Y axis is time.

The figure tells how long it took from the signal to travel from transducer to hydrophone for each measurement point. The distance can be calculated with formula

$$s = t * c \tag{1}$$

where *s* is length, *t* is time and *c* is speed of sound.

When calculated this way, we get distance 0.0196m when hydrophone is facing the transducer at measurement point 17. The idea of a semicircle measurement is that the distance between sender and receiver is same through the measurement. Only the angle between transducer and hydrophone changes during the measurement. With this kind of measurement, it is possible to characterize the angular divergence of a TAS. From the figure we can see that the distance remains almost the same through the measurement. The small difference in the distance is caused by the 1mm resolution in the positioning hardware.

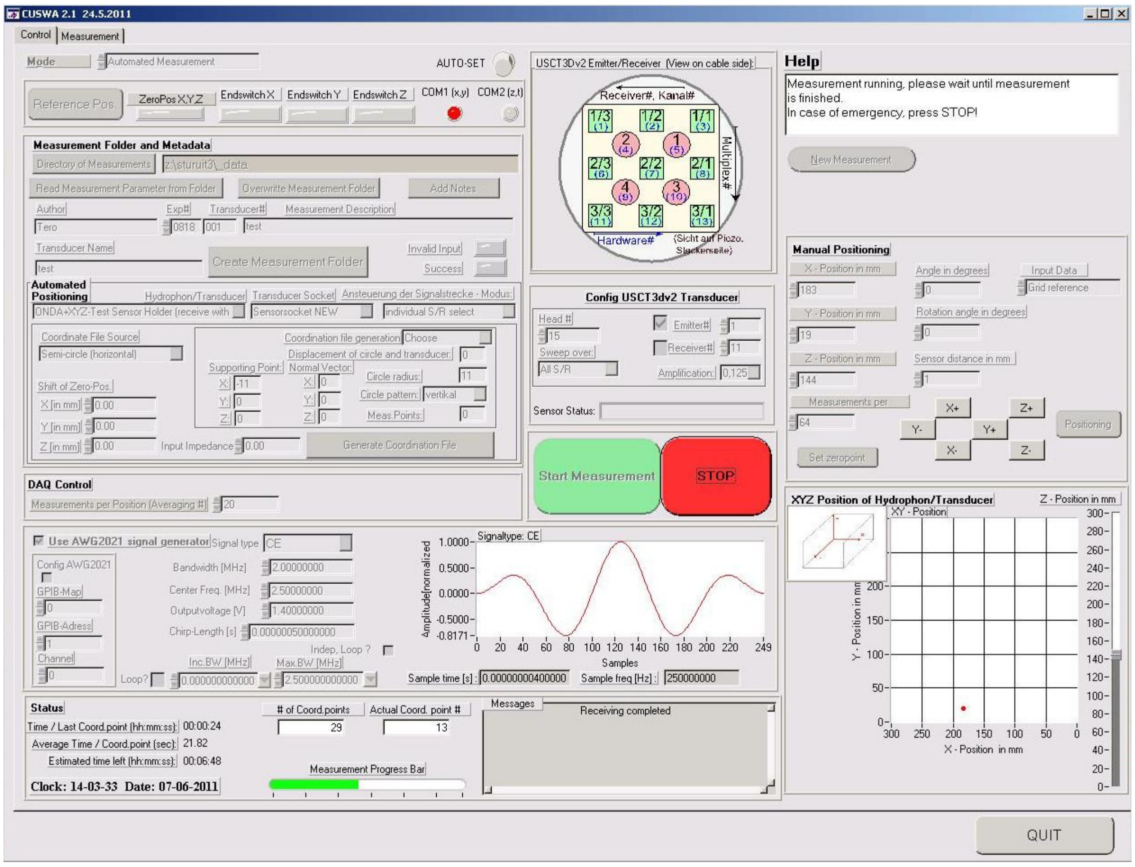
Figure 6 on page 15 shows a graph of the CE signal used in test measurement five. The graph is taken from the measurement data which was received with hydrophone.

In addition to the CE signal, there is also some noise visible in the graph. X – axis is the bandwidth and Y – axis is amplitude.

4.3 GUI Upgrades

The most visible upgrade to the GUI is the help window located in the upper right corner. With the help window users did not need separate instructions on a paper, because the help window tells the user what to do next in order to get a measurement running. With the help window and by dimming all unnecessary parts of the GUI, setting all the settings and parameters was much easier and less time consuming. Other upgrades visible to the user are located in the lower left corner in the status section. The status section contains information about the measurement progress in form of progress bar, time spent per position (average and last position) and estimated time left. Also the total number of measurement points and the actual measurement point are displayed.

In addition to these visible upgrades, the functionality of some buttons was either finished or modified. The “Read Measurement Parameter from Folder” buttons functionality was finished, in order to make it easier and faster to redo a measurement. The users can now redo any measurement they have done earlier, with just few clicks.


CUSWA 2.1 24.5.2011

Control | Measurement

Mode: Automated Measurement

Measurement Folder and Metadata

Directory of Measurements: z:\sturuit3_data

Author: Tero
Exp#: 0818
Transducer#: 001
Measurement Description: test

Transducer Name: test

Automated Positioning

Hydrophon/Transducer: ONDA+XYZ-Test Sensor Holder (receive with
Transducer Socket: Sensorsocket NEW
Ansteuerung der Signalstrecke - Modus: individual S/R select

Coordinate File Source: Semi-circle (horizontal)
Coordination file generation: Choose
Displacement of circle and transducer: 0

Supporting Point: X: -11, Y: 0, Z: 0
Normal Vector: X: 0, Y: 0, Z: 0
Circle radius: 11
Circle pattern: vertikal
Meas.Points: 0

Shift of Zero-Pos.: X [in mm]: 0.00, Y [in mm]: 0.00, Z [in mm]: 0.00
Input Impedance: 0.00

DAQ Control

Measurements per Position (Averaging #): 20

Use AWG2021 signal generator ✓
Signal type: CE
Bandwidth [MHz]: 2.00000000
Center Freq. [MHz]: 2.50000000
Outputvoltage [V]: 1.40000000
Chirp-Length [s]: 0.00000050000000
GPIB-Map: 0
GPIB-Adress: 1
Channel: 0
Inc.BW [MHz]: 0.000000000000
Max.BW [MHz]: 2.500000000000

Sample time [s]: 0.00000000400000
Sample freq [Hz]: 250000000

Status

Time / Last Coord.point (hh:mm:ss): 00:00:24
Average Time / Coord.point (sec): 21.82
Estimated time left (hh:mm:ss): 00:06:48
Clock: 14-03-33 Date: 07-06-2011

# of Coord.points: 29
Actual Coord. point #: 13

Messages: Receiving completed

Config USCT3dv2 Transducer

Head #: 15
Sweep over: All S/R
Emitter# ✓: 1
Receiver#: 11
Amplification: 0,125

Help

Measurement running, please wait until measurement
is finished.
In case of emergency, press STOP!

Manual Positioning

X - Position in mm: 183
Y - Position in mm: 19
Z - Position in mm: 144
Measurements per: 64
Angle in degrees: 0
Rotation angle in degrees: 0
Sensor distance in mm: 1
Input Data: Grid reference

Figure 22. Enhanced GUI with Help window in the upper right corner.

5 CONCLUSION

The aim of this thesis was to redesign and generalize the structure of existing ultrasound transducer measurement station software. The measurement station was first created in 2005 and has since been extended several times. The software structure became very complex and it was very hard for anyone to do further development for the software. The software code was commented partly in German and partly in English and there was no appointed coding style or policies to follow. That is why a pretty long introduction phase was required to get into the software.

The outcome of this thesis was a generalized program structure by using Table driven design. The created structure is simple and easily extendable for further development, as the structure follows a certain concept. The table driven design was well fitting for this project because of the program’s iterative nature of executing the same code several times in a row.

The graphical user interface of the software became easier and faster to use as a result of the upgrades made to it. The settings are easy to set with the guidance of the help window and also because the order to set settings is systematic, going from up to down. The GUI became faster to use because of the automated filling of settings when redoing a certain measurement.

After implementing the selected concept and the upgrades to the GUI, the functionality of the software was tested with several test measurements. Test measurements were done at least once with every possible setting and parameter to make sure that the software was fully functional. The amount of test measurements was sufficient to ensure that the funcionality of the software was appropriate.

The structuring of my work and schedule were planned successfully. There was no lack of time while doing my thesis, except while doing the test measurements. The finishing of writing this thesis was done after the 6 months’ period. If I had to do the same work again, I would do more documentation during the whole work, not just at the end. It would then be a lot easier to do the documentation, when there were some material ready to start with.

As a personal goal I had improving my skills at C programming and also improving my linguistic skills in written and spoken English. I feel that I succeeded in both of my personal goals quite well. Also this was my first work as a part of a group primarily focused on software development, and it was very educational for me.

REFERENCES

1. Lars Petzold, Diplomarbeit. Aufbau eines Messplatz zur Ermittlung der Schallfeldcharakteristik eines Ultraschallwandlers. [Reference made 07.06.2011] http://www.ipe.fzk.de/~ruiter/DPA/Diplom.pdf

2. Nicole V. Ruiter, Gregor F. Schwarzenberg, Michael Zapf and Hartmut Gemmeke. Conclusion from an Experimental 3D Ultrasound Computer Tomography. [Reference made 07.06.2011]
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4774292&tag=1

3. Karlsruhe Institute of Technology, Institute for Data Processing and Electronics. Early Breast Cancer Detection with Ultrasound Computer Tomography [Reference made 07.06.2011]
http://www.ipe.kit.edu/english/167.php

4. Nicole V. Ruiter, Gregor F. Schwarzenberg, Michael Zapf, R. Liu, Rainer Stotzka, Hartmut Gemmeke. 3D ultrasound computer tomography: Results with a clinical breast phantom. [Reference made 5.7.2011]
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4152113

5. Gerald R. Harris. Hydrophone measurements of medical ultrasound devices. [Reference made 4.7.2011]
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1535750

6. National Instruments Corporation (1998). Getting Started with *LabWindows/CVI* [reference made 06.06.2011].
http://www.ni.com/pdf/manuals/320680d.pdf.

7. Cunneyworth, W. (1994). Table driven design. Data Kinetics Ltd. [reference made 08.06.2011]
http://www.dkl.com/html/Files/CMFiles/53table_driven_design.pdf

8. Rainer Stotzka, Helmut Widmann, Tim Muller, Klaus Schlote Holubek, Hartmut Gemmeke, Nicole Ruiter, Georg Gobel. Prototype of a new 3D ultrasound computer tomography system: transducer design and data recording. Forschungszentrum Karlsruhe. 2004. August 4, 2005.
http://www.stotzka.de/Publications/stotzka2004.1.pdf

9. Hanov Solutions Inc. WebSequenceDiagrams
http://www.websequencediagrams.com/index.php

10. TortoiseSVN Subversion Client
http://tortoisesvn.tigris.org/

Program code of the main measurement loop

```
void Do_Measurement_Loop()
{

    int error = 0;

    char *msg = malloc(512);
    char buffer[512];
    char buffer2[512];
    char buffer3[512];

    memset(msg, 0, 512*sizeof(char));   // initialize msg

    // update logfile
    strcat(msg, "Measurement started:\n");
    strcat(msg, currentDateAndTime);
    writeLogFile(msg);
    memset(msg, 0, 512*sizeof(char));

    FileNameExtension = TRUE;

    // Set flag measurement started
    measuringStartet = 1;

    // initialize tas_index
    glob.tas_index = 0;

    //Coord loop start
    while (glob.CT[INDEXCOORD]<=MaxIndexKoord)
    {

        ProcessSystemEvents();

        if(glob.CT[MEASMODE] == 0) // if whole TAS measurement
            ConfigWholeTAS(glob.CT[MEASSR], 1, glob.CT[INDEXCOORD]);
// configurations for whole TAS measurement

        // Set coordinates
        SollPos.lX_Koord = Koords[glob.CT[INDEXCOORD]];
        SollPos.lY_Koord = Koords[(glob.CT[INDEXCOORD] +1)];
        SollPos.lZ_Koord = Koords[(glob.CT[INDEXCOORD] +2)];

        // Set coordinates into GUI
        SetCtrlVal(PANEL_2,PANEL_2_MP_NUM_1,
(Koords[glob.CT[INDEXCOORD]]));
        SetCtrlVal(PANEL_2,PANEL_2_MP_NUM_2,
(Koords[(glob.CT[INDEXCOORD] +1)]));
        SetCtrlVal(PANEL_2,PANEL_2_MP_NUM_3,
(Koords[(glob.CT[INDEXCOORD] +2)]));

        // Incerement of coord points
        glob.ActualCoordPoint = 1+glob.CT[INDEXCOORD]/3;
        glob.CT[INDEXCOORD] +=3;    // Point on next one

        // Update actual coord. # and progress bar
        SetCtrlVal(PANEL_2,PANEL_2_AT_NUM_4, glob.ActualCoordPoint);
        SetCtrlVal(PANEL_2,PANEL_2_Progress_Bar,
glob.ActualCoordPoint);

        // set new position
        SetNewPosition(glob.CT[INDEXCOORD]);
        Delay(0.1); // small delay to calm the water after
positioning

        if (glob.CT[SIGNALGEN]) // if signal generator in use
        {
            // if miniloop measure only over first pos.
            if ((glob.CT[MINILOOP]) && (glob.inc>0))
            {
                glob.CT[LOOPTYPE]=1;
                glob.fC=100e3;
            }

            // Loop over fB, fC, chirplen or outputvoltage. Run once
even if not looping
            for(glob.loop_index = glob.loop_val; glob.loop_index <
glob.max; glob.loop_index += glob.inc)
            {

configInstr(glob.CT[LOOPTYPE],glob.loop_index,glob.fB,glob.fC,glob.ch
irplen,glob.voltage);
                ProcessSystemEvents();
                Do_GaGeMeasurement1(FALSE);
                if (glob.inc == 0)
                    break; // to ensure run once break after one run
            }

            if (glob.CT[MINILOOP])
            {
                // ready the program to run normally with the
selected values
                glob.fC = glob.fC *1000000;

                // -> deactivate later freq. loops
                glob.max = glob.fC; glob.inc=0;

                // reset by init instrument
                configInstr(4,0,glob.fB, glob.fC, glob.chirplen,
glob.voltage);
            }

        }
        else //for NOT Signalgenerator
            Do_GaGeMeasurement1(FALSE);

        CalcMeasTime(glob.CT[INDEXCOORD],glob.numMeasurements);
// Calculate measurement time etc. -Tero

        if(glob.CT[MEASMODE] == 0) // if whole TAS measurement
            error = ConfigWholeTAS(glob.CT[MEASSR], 0,
glob.CT[INDEXCOORD]); // configurations for whole TAS measurement

        if(error == 1)
            return;

    }   // end of coordloop

    glob.CT[INDEXCOORD] = 0;
    AT_Koord_Funk = 0;

    // Flag measurement finished
    glob.koordinatenLoopEnd = 1; // For loop over all
senders/receivers
    closeInstr();
    // Update log file
    strcat(msg, "Measurement ended:\n");
    strcat(msg, currentDateAndTime);
    writeLogFile(msg);
    memset(msg, 0, 512*sizeof(char));

    // Undimm New measurement button
    SetCtrlAttribute(PANEL_2, PANEL_2_New_Meas, ATTR_DIMMED, FALSE);

    // Update help textbox
    ResetTextBox (PANEL_2, PANEL_2_HELP_TEXTBOX, MEAS_FINISHED_HELP);

    // Beep -sound after measurement is finished
    Beep();

    // Dim STOP -button after measurement
    SetCtrlAttribute(PANEL_2,PANEL_2_NA_COMMAND,ATTR_DIMMED,TRUE);

    if(glob.CT[MEASMODE] == 0) // if whole TAS measurement
        ConfigWholeTAS(glob.CT[MEASSR],2, glob.CT[INDEXCOORD]); //
last configurations for whole TAS measurement

    return;

}
```

Program code of the measurement preparation function

```
int measurementPreparation(int mode)
{
    double chirp_Length;
    int j;

    switch(mode)
    {

        case 0: // whole TAS

            //reset loopcounter
            glob.currentSensorInLoop = 0;

            // Set the number of measurements i.e. number of loop
passes
            switch(glob.CT[MEASSR])
            {
                case 0:
                    glob.numMeasurements = 4;
                    break;
                case 1:
                    glob.numMeasurements = 9;
                    break;
                case 2:
                    glob.numMeasurements = 13;
                    break;
            }


        break;

        case 1: // individual sender/receiver selection

            if(glob.scheck == 0 && glob.rcheck == 0)
            {
                MessagePopup("Error", "Measurement cannot be started!
Sender or receiver must be selected!");
                return 0;
            }

            if(glob.scheck == 1)     // transmitter selected
            {

                if(glob.snumber == 1 || glob.snumber == 2
||glob.snumber == 3 || glob.snumber == 4)
                {
                    //ko.isSending=1;
                    modifyKoordFile(glob.snumber);
                    initSender(glob.snumber, glob.tasnumber);  //1-->
Verst□ung // 1--> amplification
                    glob.koordinatenLoopEnd = 0;
                    glob.choosenSenderReceiver = glob.snumber;
                }
                else
                {
                    MessagePopup("Error", "Sender number is not
entered correctly!!");
                    return 0;
                }
            }

            if(glob.rcheck == 1)     // receiver selected
            {
                // Check that receiver number is valid!
                if (glob.rnumber == 11 || glob.rnumber == 12 ||
glob.rnumber == 13 || glob.rnumber == 21
                || glob.rnumber == 22 ||glob.rnumber == 23 ||
glob.rnumber == 31 || glob.rnumber == 32
                || glob.rnumber == 33)
                {
                    modifyKoordFile(glob.rnumber);
                    initReceiver(glob.rnumber, glob.tasnumber,
glob.whichGain); //1--> Verst□ung // 1--> amplification
                    glob.koordinatenLoopEnd = 0;
                    glob.choosenSenderReceiver = glob.rnumber;
                }
                else
                {
                    MessagePopup("Error", "Receiver number is not
entered correctly!");
                    return 0;
                }
            }

            glob.numMeasurements = 1;

        break;

        case 2: // no serial communication

            glob.koordinatenLoopEnd = 0;
            glob.numMeasurements = 1;

        break;

    }


    if (glob.CT[SIGNALGEN])//signalgenerator)
    {

        GetCtrlVal(PANEL_2, PANEL_2_file, glob.file);
        StringUpperCase(glob.file);

        Fmt(glob.efile, "%s.EQU", glob.file);
        Fmt(glob.wfile, "%s.WFM", glob.file);

        glob.loop_val = 0; // initialize

        // check if some loop type is selected
        GetCtrlVal(PANEL_2, PANEL_2_fB_on, &glob.loop);
        if (glob.loop) glob.CT[LOOPTYPE] = 0;
        else {
            GetCtrlVal(PANEL_2, PANEL_2_fC_on, &glob.loop);
            if (glob.loop) glob.CT[LOOPTYPE] = 1;
            else {
                GetCtrlVal(PANEL_2, PANEL_2_fO_on, &glob.loop);
                if (glob.loop) glob.CT[LOOPTYPE] = 3;
                else {
                    GetCtrlVal(PANEL_2,PANEL_2_CHIRP_LEN_on,
&glob.loop);
                    if (glob.loop) glob.CT[LOOPTYPE] = 2;
        }}}

        switch (glob.CT[LOOPTYPE])
        {
            case  0 :

                glob.inc *= 1000000;        // MHz/Hz-Conversion
                glob.max *= 1000000;
                glob.loop_val = glob.fB*1000000;

            break;

            case  1 :

                glob.inc *= 1000000;        // MHz/Hz-Conversion
                glob.max *= 1000000;
                glob.loop_val = glob.fC*1000000;

            break;

            case  2 :
                glob.loop_val = glob.chirplen;
            break;

            case  3 :
                glob.loop_val = glob.voltage;
            break;


        }

        initInstr();
        if (instrHandel == NULL)
        return 0;

    } //end of signal generator code

    for(j=0;j<4;j++)     // = {1,2,3,4}
        glob.SenderNr[j] = j+1;

    for(j=0;j<3;j++)     // {11,12,13,21,22,23,31,32,33}
    {
        glob.ReceiverNr[j]   = 11+j;
        glob.ReceiverNr[j+3] = 21+j;
        glob.ReceiverNr[j+6] = 31+j;
    }

    glob.nSender    = 4;
    glob.nReceiver  = 9;

    // Calculate chirp length
    chirp_Length = 1 / (glob.fB*1000000);

    // Set chirp length into GUI
    SetCtrlVal(PANEL_2,PANEL_2_CHIRP_LEN, chirp_Length);

    // Start signalpreview
    showType() ;

    glob.XYState = 1;
    glob.ZTState = 1;

    SetCtrlVal(PANEL_2,PANEL_2_AT_NUM_4,0);
    glob.ActualCoordPoint = 0;

    // Read the coordinates
    Read_Koords();

    // update coorpoints into GUI
    SetCtrlVal(PANEL_2,PANEL_2_AT_NUM_3,glob.CoordPoints);
    SetCtrlVal(PANEL_2,PANEL_2_AT_NUM_4,0);

    // Set progress bar maximum value

SetCtrlAttribute(PANEL_2,PANEL_2_Progress_Bar,ATTR_MAX_VALUE,glob.Coo
rdPoints);

    return 1; // return 1 if everything is ok

}  // End of measurementPreparation()
```

Program code of the CalcMeasTime -function

```
// function for calculating measurement time etc.
int CalcMeasTime(int IndexKoord, int numMeasurements)
{

    unsigned int hilf_hour, hilf_min, hilf_sec, timeInSecGes,
currentSensorInLoop;
    unsigned int hilf_hour3, hilf_min3, hilf_sec3;

    double timeInSecGes2;
    static double diff, diff1;

    char buffer[512];
    char buffer2[512];
    char buffer3[512];

    static time_t oldTime;
    static time_t startTime;
    time_t nextTime;
    static int round;
    static int calc_index;
    static double sum;
    static int points;

    if ( calc_index != 1)
    {
         // Print "Calculating" into GUI
         SetCtrlVal(PANEL_2,PANEL_2_suspectedExpTime,
"Calculating...");
         SetCtrlVal(PANEL_2,PANEL_2_suspectedExpTime_2,
"Calculating...");
         SetCtrlVal(PANEL_2,PANEL_2_suspectedExpTime_3,
"Calculating...");

         // Start time
         time ( &startTime );

         oldTime = startTime;

         calc_index = 1;
         sum = 0;
         points = 0;
         return 0;
    }


    if( calc_index == 1)
    {

        // Get time
        time ( &nextTime );

        // The time between last two measurements in seconds
        diff = difftime(nextTime, oldTime);

        oldTime = nextTime;

        // Seconds from the start of the measurement
        diff1 = difftime(nextTime, startTime);

        // estimated time left
        timeInSecGes = diff * (glob.CoordPoints -
glob.ActualCoordPoint);
```

```
        if(glob.ActualCoordPoint == 0)
            round++;

        // time/coord.pos (avrg)
        sum = sum + diff;
        points++;
        timeInSecGes2 = sum/points;

        if(timeInSecGes > 35999) // if time is more than 99:99:99,
set time to 0
        {
            timeInSecGes = 0;
        }

        // Time calculation for whole TAS measurement
        if(glob.CT[MEASMODE] == 0)
        {
            currentSensorInLoop = glob.currentSensorInLoop;     //
Current sensor in measurement loop
            timeInSecGes = diff * ( (numMeasurements *
glob.CoordPoints) - (( currentSensorInLoop * glob.CoordPoints) +
(1+(IndexKoord-9)/3)) );
        }

        // Convert times from seconds into hh:mm:ss
        // Estimated time left
        hilf_hour = timeInSecGes / 3600;
        hilf_min = (timeInSecGes-(hilf_hour*3600))/60;
        hilf_sec = timeInSecGes - ((hilf_hour*3600)+(hilf_min*60));

        // Time between last 2 coord.points
        hilf_hour3 = diff / 3600;
        hilf_min3 = (diff-(hilf_hour3*3600))/60;
        hilf_sec3 = diff - ((hilf_hour3*3600)+(hilf_min3*60));

        // Format calendar time to string
        sprintf(buffer, "%02d:%02d:%02d",
hilf_hour,hilf_min,hilf_sec);
        sprintf(buffer2, "%2.2f",timeInSecGes2);
        sprintf(buffer3, "%02d:%02d:%02d",
hilf_hour3,hilf_min3,hilf_sec3);

        // Set calculated values into GUI
        SetCtrlVal(PANEL_2,PANEL_2_suspectedExpTime, buffer);
// estimated time left
        SetCtrlVal(PANEL_2,PANEL_2_suspectedExpTime_2, buffer2);
// time/coord.pos (avrg)
        SetCtrlVal(PANEL_2,PANEL_2_suspectedExpTime_3, buffer3);
// time between last 2 coordinate points

    }

    return 0;

}
//----------------------End of time calculations --------------------
```

Program code of the SetNewPosition -function

```
/ positioning of the XYZ -holder
SetNewPosition(int indexcoord)
{
    int SendThreadError = 0;

        // XY moved, Z moving
            Set_TxBufferMP(PANEL_2, ZT);

            // Send ZT to DSP
            SendThreadError
=CmtScheduleThreadPoolFunction(DEFAULT_THREAD_POOL_HANDLE,

Send_BytesZT_ThreadFunction,NULL,&ThreadID[1]);

            // Show the text output
            if(SendThreadError !=0)
                Show_ThreadMessage(SendThreadError);
            else
                Show_ElseMessage(STATUS_1);

            CleanUpSendThreads(ZT_THREAD);

        // Z moved, XY moving
            Set_TxBufferMP(PANEL_2, XY);

            // Send XY to DSP
            SendThreadError
=CmtScheduleThreadPoolFunction(DEFAULT_THREAD_POOL_HANDLE,

Send_BytesXY_ThreadFunction,NULL,&ThreadID[0]);

            // Show the text output
            if(SendThreadError !=0)
                Show_ThreadMessage(SendThreadError);
            else
                Show_ElseMessage(STATUS_1);

            CleanUpSendThreads(XY_THREAD);

        // wait until both COM1 & COM2 positioned
        while(SollPos.lX_Koord!=AktPos.lX_Koord ||
SollPos.lY_Koord!=AktPos.lY_Koord ||
SollPos.lZ_Koord!=AktPos.lZ_Koord)
        {
            ProcessSystemEvents();
        }

   return 0;
}
```

Program code of the ConfigWholeTAS -funtion

```
// configurations during measurement for whole TAS measurement
ConfigWholeTAS(int sr, int phase, int index)
{

    int continueMeasurement = 0;
    int error = 0;
    static int previous = -1;
    char pathToOldData[MAX_PATHNAME_LEN];
    char *msg = malloc(512);
    memset(msg, 0, 512*sizeof(char));  // initialize msg

    if(phase == 1)  // if before measurement
    {

        if((sr == 1) && (glob.tas_index < glob.nReceiver) &&
(glob.tas_index != previous)) // if receiver selected
        {

            // receiver conversion
            if ((glob.tas_index==0) || (glob.tas_index==3) ||
(glob.tas_index==6))
            {
                // Stop measurement for modifying the hardware
                sprintf(msg, "Please change the receiver to channel
%i. Continue measurement by pressing
'Yes'.",(int)((glob.tas_index/3)+1));
                continueMeasurement = ConfirmPopup ("Measurement
stopped", msg);
                memset(msg, 0, 512*sizeof(char));

                // Terminate measurement if "No" pressed
                if(continueMeasurement == 0)
                {
                    MessagePopup("Measurement ended","Measurement
aborted by user!");
                    return 1;
                }

            }

            createSubFolder(glob.pathToExp, glob.sensorName,
glob.PathToSensorFolder, glob.pathToLogFile,
glob.ReceiverNr[glob.tas_index]);
            modifyKoordFile(glob.ReceiverNr[glob.tas_index]);
            initReceiver(glob.ReceiverNr[glob.tas_index],
glob.tasnumber, glob.whichGain);
            ResetTextBox(PANEL_2, PANEL_2_serialStatus, "");
            sprintf(msg, "Receiver %d
active...",glob.ReceiverNr[glob.tas_index]);
            InsertTextBoxLine (PANEL_2, PANEL_2_serialStatus, 0,
msg);
            memset(msg, 0, 512*sizeof(char));
            SetCtrlVal(PANEL_2, PANEL_2_RNumber,
glob.ReceiverNr[glob.tas_index]);

            // Update variables
            glob.currentSensorInLoop++;
            glob.choosenSenderReceiver =
glob.ReceiverNr[glob.tas_index];
            glob.koordinatenLoopEnd = 0;
            glob.tas_index++;
            previous = glob.tas_index;
            glob.CT[INDEXCOORD] = 0;

            if(glob.tas_index > 9)  // check if all receivers
measured
                glob.CT[INDEXCOORD] = MaxIndexKoord+1;

        }

        if((sr == 0 || sr == 2) && (glob.tas_index < glob.nSender) &&
(glob.tas_index != previous))  // if sender or sender&receiver
selected
        {

            createSubFolder(glob.pathToExp, glob.sensorName,
glob.PathToSensorFolder, glob.pathToLogFile,
glob.SenderNr[glob.tas_index]);

            modifyKoordFile(glob.SenderNr[glob.tas_index]);
            initSender(glob.SenderNr[glob.tas_index],
glob.tasnumber);
            ResetTextBox(PANEL_2, PANEL_2_serialStatus, "");
            sprintf(msg, "Sender %d
active...",glob.SenderNr[glob.tas_index]);
            InsertTextBoxLine (PANEL_2, PANEL_2_serialStatus, 0,
msg);
            memset(msg, 0, 512*sizeof(char));
            SetCtrlVal(PANEL_2, PANEL_2_SNumber,
glob.SenderNr[glob.tas_index]);

            glob.currentSensorInLoop++;
            glob.choosenSenderReceiver =
glob.SenderNr[glob.tas_index];
            glob.koordinatenLoopEnd = 0;
            glob.tas_index++;
            previous = glob.tas_index;
            glob.CT[INDEXCOORD] = 0;

            if((sr == 0) && (glob.tas_index > 4))
                glob.CT[INDEXCOORD] = MaxIndexKoord;
        }

        Read_Koords();

    }
    if(phase == 0) // if after measurement
    {

        if((sr == 1) && (index > MaxIndexKoord) &&
(glob.tas_index<glob.nReceiver))
        {
            glob.CT[INDEXCOORD] = 0;
            previous = -1;
        }

        if((sr == 2) && index > MaxIndexKoord)
        {

            if(glob.tas_index>=glob.nSender)
            {
                glob.CT[MEASSR] = 1;     // sweep over receiver
                glob.hydrophon = 3;      // send with tas holder
                continueMeasurement = ConfirmPopup ("Measurement
stopped", "Sender measurement complete. Prepare hardware for receiver
measurement and press 'Yes'");
```

```
                // Terminate measurement if "No" pressed
                if(continueMeasurement == 0)
                {
                    MessagePopup("Measurement stopped","Measurement
aborted by user!");
                    return 1;
                }

                glob.tas_index = 0;

            }

            glob.CT[INDEXCOORD] = 0;
            previous = -1;
        }



    }

    if(phase == 2)  // if end of measurement
    {
        ///end message
        MessagePopup("Measurement completed","Measurement completed
succesfully!");

        strcpy (pathToOldData, glob.pathToExp);
        strcat (pathToOldData, PATH_SEPERATOR);
        strcat (pathToOldData, "Template, modTemplate und LogFile");

        if(FileExists(pathToOldData,0) == 0)
        {
            error = rename(glob.PathToSensorFolder, pathToOldData);
        }else
        {
            sprintf(msg, "cmd.exe /C rmdir /S /Q
C:\"%s\"",pathToOldData);
            error = LaunchExecutableEx (msg, LE_HIDE, NULL);
            memset(msg, 0, 512*sizeof(char));

            error = rename(glob.PathToSensorFolder, pathToOldData);

        }

        if(msg!=NULL)
        {
            free(msg);
        }
    }


    return 0;
}
```

Program code of the auto measurement Callback –function

```
nt CVICALLBACK StartAutoMeasurement (int panel, int control, int
event,
        void *callbackData, int eventData1, int eventData2)
{

    char *msg = malloc(512);    // msg for writing parameters into
log file

    switch (event)
    {

        case EVENT_COMMIT | 3:

        // Undim STOP -button

SetCtrlAttribute(PANEL_2,PANEL_2_NA_COMMAND,ATTR_DIMMED,FALSE);

        // Dimm Start measurement -button
        SetCtrlAttribute(PANEL_2, PANEL_2_AT_COMMAND_1, ATTR_DIMMED,
TRUE);

        // Dimm all inputs during measurement
        Dimm_Input();

        // Update help textbox
        ResetTextBox (PANEL_2, PANEL_2_HELP_TEXTBOX,
MEAS_RUNNING_HELP);

        // msg for the logfile entries, allocate with 0
        memset(msg, 0, 512*sizeof(char));

        // Read the rest parameters/settings from GUI into global
variables
        GetCtrlVal(PANEL_2, PANEL_2_type, &glob.type);
// Signal type
        GetCtrlVal(PANEL_2, PANEL_2_fB, &glob.fB);
// Bandwidth
        GetCtrlVal(PANEL_2, PANEL_2_fC, &glob.fC);
// Center frequrency
        GetCtrlVal(PANEL_2, PANEL_2_Outputvoltage, &glob.voltage);
// Output voltage
        GetCtrlVal(PANEL_2, PANEL_2_CHIRP_LEN, &glob.chirplen);
// Chirp length
        GetCtrlVal(PANEL_2, PANEL_2_GPIG, &glob.gpig);
// Config AWG Y/N
        GetCtrlVal(PANEL_2, PANEL_2_id, &glob.id);
// GPIB map
        GetCtrlVal(PANEL_2, PANEL_2_addr, &glob.addr);
// GPIB adress
        GetCtrlVal(PANEL_2, PANEL_2_channel, &glob.channel);
// Channel
        GetCtrlVal(PANEL_2, PANEL_2_INC, &glob.inc);
// Inc. value for looping
        GetCtrlVal(PANEL_2, PANEL_2_MAX, &glob.max);
// Max value for looping
        GetCtrlVal(PANEL_2, PANEL_2_Gain, &glob.whichGain);
// Receiver attenuation  0=1.0, 1=0.5, 2=0.25, 3=0.125
        GetCtrlVal(PANEL_2, PANEL_2_AT_NUM_2, &MessSchuesse);
// Measurements per point
        GetCtrlVal(PANEL_2, PANEL_2_AT_NUM_1, &glob.uiAbstand);
// Distance between points
```

```
        // Read control variables into global control table
        GetCtrlVal(PANEL_2, PANEL_2_miniloop_on, &glob.CT[MINILOOP]);
// Miniloop on/off
        GetCtrlVal(PANEL_2, PANEL_2_on, &glob.CT[SIGNALGEN]);
// Signalgenerator on/off

        // Write parameters to experiment data log file

sprintf(msg,"%i\n%f\n%f\n%f\n%f\n%i\n%i\n%i\n%i\n%f\n%f",glob.type,gl
ob.fB,glob.fC,glob.voltage,glob.chirplen,glob.gpig,glob.id,glob.addr,
glob.channel,glob.inc,glob.max);
        writeExpLogFile(msg);
        memset(msg, 0, 512*sizeof(char));

        // Write parameter to log file
        sprintf(msg,"\nSelected parameters:\n\nBandbwidth:\t%llf
MHz\nCenterfrequency:\t%llf MHz\nOutputvoltage:\t%llf
Volt\n\n\n",glob.fB,glob.fC,glob.voltage);
        writeLogFile(msg);
        memset(msg, 0, 512*sizeof(char));

        ProcessSystemEvents();

        // Do preparations before running measurement
        if(measurementPreparation(glob.CT[MEASMODE]) == 0)
            return 0; // quit if any errors in
measurementPreparation();

        // the main measurement loop
        Do_Measurement_Loop();

        break;
    }

    if(msg!=NULL)
    {
       free(msg);
    }

    return 0;
}
```