# Development and Implementation of a Test Sequence for a Functional Tester

Petter Erikslund

Examensarbete för ingenjör (YH)-examen

Utbildningsprogrammet Elektroteknik

Vaasa 2017

**BACHELOR'S THESIS**

| | |
|---|---|
| Author: | Petter Erikslund |
| Degree Programme: | Electrical Engineering, Vasa |
| Specialization: | Automation |
| Supervisors: | Matts Nickull |

Title: Development and Implementation of a Test Sequence for a Functional Tester

Date May 26 2017 Number of pages 32

## Abstract

The purpose of this thesis is to develop and implement a test sequence for a final functional tester. The assignment was commissioned by Ampner Oy. The company produces products and services such as solutions for connecting renewable power sources to the grid and automated test systems.

Testing in electronics manufacturing is explained and the development environments NI LabVIEW and NI Teststand are introduced. The methods for developing the test sequence, such as sequence structure and test duration optimisation are documented and areas for further development are suggested.

The test sequence developed in this thesis is automatic and only requires the operator to place the DUT in the test fixture and start the test. The sequence starts by loading test limits from a file and initiating the instruments. The test cases are then executed sequentially. If the test is a pass, a sticker with identification information is printed and attached to the tested device.

Language: english Key words: FCT, Teststand, Automatic test equipment

**EXAMENSARBETE**

| | |
|---|---|
| Författare: | Petter Erikslund |
| Utbildning och ort: | Elektroteknik, Vasa |
| Inriktningsalternativ: | Automationsteknik |
| Handledare: | Matts Nickull |

Titel: Utveckling och implementering av en testsekvens för en funktionell testare

Datum 26.5.2017 Sidantal 32

**Abstrakt**

Målet med detta examensarbete var att utveckla och implementera en testsekvens för en slutlig funktionell testare. Arbetet beställdes av Ampner Oy. Företaget producerar produkter och tjänster såsom lösningar för anslutning av förnybara energikällor till elnätet och automatiserade testsystem.

Testning inom elektroniktillverkning förklaras och utvecklingsmiljöerna NI LabVIEW och NI Teststand introduceras. Metoderna för att utveckla programvaran för testsekvensen, såsom sekvensstruktur och tidsoptimering dokumenteras och områden lämpliga för vidareutveckling föreslås.

Testsekvensen är automatisk. Operatören placerar apparaten som ska testas i testfixturen och startar testsekvensen. Testsekvensen börjar med att ladda in testets gränsvärden från en fil och initierar mätinstrumenten. De individuella testfallen utförs sedan i löpande ordning. Om testet är godkänt förses apparaten med en etikett med identifikation för den testade apparaten.

Språk: engelska Nyckelord: FCT, Teststand, Automatiserad testutrustning

**OPINNÄYTETYÖ**

|  |  |
|---|---|
| Tekijä: | Petter Erikslund |
| Koulutus ja paikkakunta: | Sähkötekniikka, Vaasa |
| Suuntautumisvaihtoehto: | Automaatiotekniikka |
| Ohjaaja: | Matts Nickull |

Nimike: Toimintatesterin testisekvenssin kehittäminen ja toteutus

---

Päivämäärä 26.5.2017 Sivumäärä 32

---

**Tiivistelmä**

Tämän opinnäytetyön tarkoituksena oli kehittää ja toteuttaa testisekvenssi toiminnallista lopputesteriä varten. Työ on toteutettu Ampner Oy:n toimeksiannosta. Ampner tuottaa tuotteita ja palveluja uusiutuvien energialähteiden sähköverkkoon yhdistämiseksi ja automaattisia testausjärjestelmiä teollisuuslaitteille.

Työssä kerrotaan elektroniikan tuotantotestauksesta ja esitellään NI LabVIEW- ja NI Teststand -kehitysympäristöt. Työssä on dokumentoitu menetelmiä testisekvenssin kehittämiseksi, kuten sekvenssirakenteen ja testin keston optimointi. Lisäksi ehdotetaan aiheita jatkokehitykselle.

Tässä työssä kehitetty testisekvenssi on automaattinen ja vaatii operaattorilta ainoastaan tuotteen sijoittamisen testeriin ja testin käynnistyksen. Testisekvenssi aloitetaan lataamalla testin rajat tiedostosta sekä alustamalla mittalaitteet. Tämän jälkeen testit suoritetaan peräkkäin. Jos testi menee läpi, tulostetaan tuotteen tunnistetiedot sisältävä tuotetarra ja liimataan se tuotteeseen.

---

Kieli: englanti Avainsanat: FCT, Teststand, automaattinen testauslaitteisto

---

# Table of Contents

## List of abbreviations

|  |  |
|---|---|
| API – Application Programming Interface | |
| ATE – Automated Test Equipment | |
| DFT – Design For Testing/Design For Testability | |
| DMM – Digital Multi Meter | |
| DUT – Device Under Test | |
| FCT - Functional Testing | |
| GPIO – General Purpose Input/Output | |
| GUI – Graphical User Interface | |
| ICT – In-circuit Testing | |
| NI – National Instruments | |
| PCB - Printed Circuit Board | |
| PCBA – Printed Circuit Board Assembly | |
| QA – Quality Assurance | |
| QC – Quality Control | |
| R&D – Research and Development | |
| UUT – Unit Under Test | |

# 1 Introduction

This thesis was made for Ampner Oy, in Vasa during the autumn of 2016 as part of a customer order. The goal of the thesis was to develop and implement a test sequence for a final functional tester for an embedded electronic device. Due to the nature of the work the thesis will not go into detail about the DUT. The focus of the thesis will be on electronics testing in general and it will explain the processes involved in developing tester software.

As electronics manufacturing is a highly automated process and the products being produced are becoming more and more complex quality control and quality assurance is a prominent part of the electronics manufacturing business. There are almost as many tester types and testing techniques as there are different types of electronic products. This thesis will make an attempt to introduce the reader to techniques used to test PCBs, PCBAs and assembled devices. The test sequence developed as a part of this thesis is used to test a fully assembled device as part of quality control right before it is approved to be delivered to the customer.

NI LabVIEW and NI Teststand were used as the development environment. Teststand is not usually used as the test sequencer at the company, thus a thesis was natural and benefited both the author and the company. When the thesis work started the hardware design was already done. The goal for the thesis work was to develop the software and implement and test the system.

After a week of reading up on Teststand the work transitioned into implementing and customizing instrument drivers to work with Teststand. The instruments and DUT were tested individually and manually to ensure that communication and test concepts worked. Each test case was individually developed and tested according to the test specification delivered by the customer. As soon as the tester hardware was assembled implementation and testing of the software continued on the tester itself. The tester specification continuously evolved during the process as the DUT itself was still under development.

## 1.1 The company

Ampner Oy is an engineering company that produces products and services such as solutions for connecting renewable power sources to the grid and automated production and test systems. Today Ampner Oy has almost 2 million euros in revenue and employs 19 people. The company is located in Vaasa, Finland.

Testcom Solutions is a department at Ampner Oy that designs and produces automated production and test systems and test equipment for the electronics industry. The Testcom brand has been supplying testing products and services to the industry for over a decade. Some of the services Testcom Solutions offer are production and test requirements planning, Design for Automation (DFA), Design for Testability (DFT) and system commissioning and after sales services. The Testcom brand also comprises products like quality control software, power supplies and various I/O modules.

Ampner Power offer design and consultation services of power converters and grid connections.  So far, over 5000 MW of renewable energy have been connected to the grid using solutions and know how delivered by Ampner. (Ampner 2017).

The company was founded in 2012 by Pasi Törmänen and Mika Jantunen, who have extensive experience from the energy industry. In 2015 Ampner became the majority owner of the company Testcom, which led Ampner into a new industry. New capital from private and local investors was brought into Ampner in 2016 and the company is expanding into new international markets. (Stenbacka 2016).

## 1.2 Purpose and goals

This thesis is made as a part of a customer order which means that the scope of thesis is naturally defined. The thesis practical part consists of development and implementation of a test sequence for a final functional tester. A functional tester tests that all the functions of the Device Under Test (DUT) meets the specified requirements. The DUT in this case is a customer’s tailor made diagnostic and communication device. The goal of this thesis is to develop software for a pilot test station according to the end customer’s test requirement specification. Commissioning and verification testing of the complete system is also included.

Instrument drivers and some of the test functions are developed using National Instruments LabVIEW and the test sequence is implemented in NI Teststand. The goal of the test sequence is to first download and flash software to the DUT and then test all functionality of the DUT. All sequence functionality must be automatic. The operator will place the DUT in the fixture, press start, and scan the DUT information from the DUT. Depending on the outcome of the tests a sticker is printed automatically and placed on the DUT by the operator.

To help the reader to get a better understanding of the practical part of the thesis, an overview of electronics manufacturing and testing is also a part of the thesis. Throughout the thesis focus lays on testing in electronics manufacturing on the assembly levels PCB and PCBA or unit. Testing brought up in this thesis is concerned with the question; have we built our product right? Design validation and testing on chip and component assembly levels are not included.

# 2 Testing in electronics manufacturing

Electronics manufacturing is highly automated and all automated processes needs to be monitored to ensure that they are working as they are supposed to. One way of monitoring the behaviour is to examine the output. It is possible to assure that the process is producing quality products by testing the electronic products at different stages throughout the manufacturing process. Because the scope of this thesis is defined by the development and implementation of a tester, the theory brought up is kept limited to the background and the theory relevant to the specific tester.

## 2.1 Electronics manufacturing

The major stages in the manufacturing of an electronic product are:

1. Chip in wafer
2. Chip diced and tested
3. Chip-level interconnection and packaging
4. Package soldered to PCB
5. PCB assembled into a PCBA or unit
6. PCBA or unit implemented into final system

This thesis scope include testing from 4. Package soldered to PCB to 5. PCB assembled into a PCBA or unit. According to Edwards (1991, 13) companies in the electronics industry can be divided into similar groups depending on what they produce:

1. Component manufacturers
2. Semiconductor device manufacturers
3. PCB bare board manufacturers
4. PCB assemblers for third parties
5. High volume assembly of PCBs and final assembly of PCBs into complete systems

6. Low volume assembly of PCBs and final assembly of PCBs into complete systems
7. Assembly-only companies which build complete systems according to customer specifications

These company types do not indicate the complexity of the actual company. An actual company may operate on several of the suggested types.

## 2.2 Quality management

All companies who produces something has an interest in assuring that the product has the desired properties and functionality when it is delivered to the customer. The strive for a quality output is the aim of quality management. Quality assurance and quality control are parts of quality management. The terms are sometimes used interchangeably but can be defined as follows: Quality assurance (QA) focuses on preventing defects and consists of activities and processes that ensures that the products developed have desired qualities. Quality control (QC) focuses on detecting defects in the product and correcting them.

Validation and verification are tasks performed by conducting tests as a part of QA and QC. Validation aims to confirm that the product will perform its intended functions desired by the customer. It asks the question; Are we creating the right product? Verification aims to confirm that the finished product meets the given requirements. It asks the question; Have we created the product right? Validation and verification might need to be performed on many different levels from system level down to component level. (ISO 9000:2015).

It becomes less time consuming and less expensive to correct a fault if is detected as soon as possible in the manufacturing. If a faulty component is changed before it is soldered to a board it is cheaper and quicker to fix the issue than detecting the fault when the device is fully assembled. Thus, testing is an important part of quality management and the manufacturing process.

## 2.3 Design for Testability

Because electronic devices become increasingly more complex, testing them has become very expensive and time consuming. Because of this it is important to take testability into account at an early stage in the product design process. Otherwise it is very likely that the product will be unnecessarily hard or even impossible to test. For example, if there are no way to access critical signals for testing on a PCB, it may be impossible to test the product in production or during development. The only solution might then be to go back to the drawing board and redo the design.

Design for testability or design for test (DFT) aims to maximise test efficiency and economy. DFT comprises of design techniques that adds testability features to the designed product. Even though most of the focus of DFT lies on digital circuits it is also applied to analog/mixed-signal circuits. Testing is not only needed in manufacturing but may also be beneficial in the design phase and in system maintenance. Design modifications that can be linked to DFT are for example improved controllability and observability of internal nodes in the circuit. This would be done to allow the test engineer to get access to critical signals during a test. Adding features like built-in-tests (BIT) are also a common technique. BIT means that the product has the ability to perform a test on itself and report the results. BIT can be used both in production testing and system maintenance. (Koenemann 2006, 21-1).

## 2.4 Testing techniques and strategies

As mentioned in Chapter 2.1, there are many assembly levels in electronics manufacturing. Each one uses different procedures and techniques. The testing techniques and solutions brought up here are used for testing populated PCBs and PCBAs or units. The factors that affect the implemented test strategy can for example be complexity of the DUT, type of faults detected, production volume and already invested test infrastructure.

### 2.4.1 Automated test equipment - ATE

One of the first questions that need to be answered when developing a tester is if it should be manual, semi-automated or automated. There are advantages to all of them but in electronics manufacturing automated testers are the most common due to the large production volumes. A more manual testing setup can be preferred for example in R&D related testing. Automated test equipment can be configured into a larger test system to automatically perform a test on a DUT without any need for the operator to intervene. (Poole, a). Table 1 compares the two different approaches.

**Table 1. Comparison of manual and automated test equipment (Agilent Technologies 2013).**

| | Manual | Automated |
|---|---|---|
| Throughput | Low | High |
| Development cost | Low | High |
| Operator experience | High | Low |
| Flexibility | High | Low |

### 2.4.2 PCB visual inspection

When the PCB has been populated with components like resistors, capacitors, diodes and integrated circuits, it is visually inspected. This is done to confirm that all components have been properly assembled to the board. Features that are inspected are for example missing and wrong components, fillet size or shape, component skew, component polarity, height defects and solder joints. There are three different techniques used to perform visual inspection. They all have different abilities to detect defects.

- MVI – Manual Visual Inspection can be practical for low volumes and if the complexity of the board is low. If the volumes or complexity is higher an automated solution is preferred.

- AOI – Automated Optical Inspection equipment works by letting a camera automatically scan the board and compare it to a known good sample.

- AXI – Automated X-ray Inspection is similar to AOI but uses X-rays as its source instead of visible light. That way it is possible to detect defects hidden from view. (Poole, a)

### 2.4.3 In-circuit test – ICT

ICT refers to testing techniques where connections are made directly to points on the PCB and then connections, component values and sometimes even some DUT functionality are tested. An in-circuit tester can be very expensive to invest in, depending on the type, but can be beneficial at large quantities. It is often possible to test that the PCB is correctly made and that no faulty components have been installed with an in-circuit tester. Reasons for not gaining complete coverage of the PCB can for example be not enough test points or low value capacitors on the PCB. If low value capacitors are present on the board it might be impossible for the tester to measure them accurately due to the spurious capacitance of the tester itself.

Standard in-circuit testers connect to the DUT through a bed of nails as seen in Figure 1. The fixture containing the bed of nails is specific for different products being tested and can easily be changed out when a product with other connections is to be tested. (Poole, b)

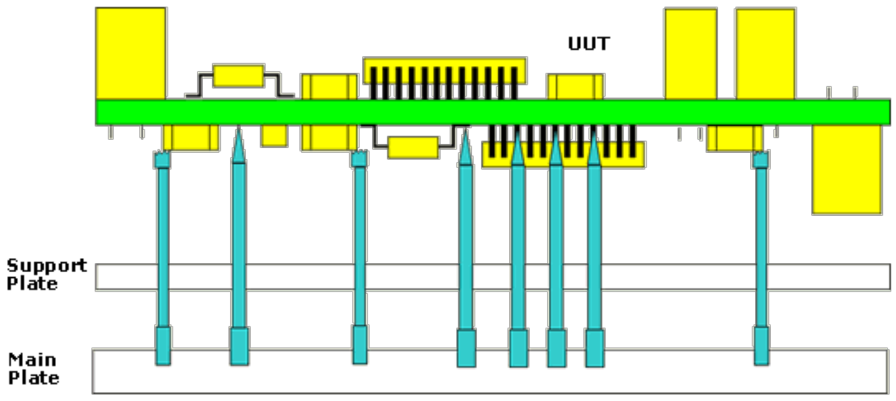


**Figure 1. Connection to DUT in ICT (Tarr).**

A different type of in-circuit tester is the flying probe tester which does not use a bed of nails. Instead it uses a set of probes that can move around to connect to various point of the board. By doing this it is possible to test much smaller quantities by not needing to invest in as much different testing hardware for each DUT model. (Poole, b)

Wiring testers also use ICT techniques. They use a bed of nails to test the wiring of a PCB or cable connectors to test cables. A wiring tester can often also test some component values and polarity.

### 2.4.4 JTAG - Boundary scan

One of the problems with ICT, especially testing large integrated circuits, is getting access to all test critical signals on the board. In 1985 a group of electronic manufacturers formed the Joint Test Action Group (JTAG) to establish a solution to the testing problem. The solution became IEEE Std 1149.1-1990, IEEE Standard Test Access Port and Boundary-Scan Architecture, which allows test instructions and data to be serially loaded into a device and test results to be serially read out.

Boundary scan works through four, or more, extra pins on the DUT. Two pins are needed for control and one each for input and output of serial test data. These connect to chosen integrated circuits on the DUT to form a chain, thus making it possible to test several integrated circuits with four pins at once. When a chip is set to boundary scan mode, it is possible to set and read all pins on the chip. This also makes it possible to test other components and wiring externally on the PCB. (Texas Instruments Inc. 1997)

### 2.4.5 Burn in

The highest failure rate for electrical devices is in their early life after production and at the end of life. Burn in ovens are used on components and devices to accelerate the aging past the initial failure prone age. This procedure is done so that one can be confident that the product works under varying conditions in accordance with the requirements and that it will maintain the performance. Burn in ovens usually test batches of products at once and can monitor their behaviour during a longer period of time in harsh temperature conditions.

### 2.4.6 FCT – Functional testing

The purpose of a functional test is to verify that the functional requirements of the DUT are met by simulating real world scenarios and checking how the product behaves. FCT is done as a last step of verification on each different level in the manufacturing chain. For example, when a PCB has been populated with components and the assembly has been verified to be correct by an in-circuit tester, the functionality of the PCB is verified by a functional tester. When the PCB has been assembled into the final device it is tested in a burn in oven. The device’s functionality is tested again on a functional tester to verify that it is ready to be installed into the final system.

It is often not practical to test all possible functionality due to the complexity of the product and time restrictions. If two different companies where to develop a functional tester for the same product, the resulting testers might look very different from each other. It is important to balance out what needs to be tested by weighting testing accuracy, time and financial restrictions against each other.

A functional tester uses the connectors on the PCB or device to test its functionality, so there is usually no verification of what happens on the inside. If there is a need for verification of internal signals, a tester can use several testing methods at the same time. For example, a functional tester might use boundary scan to verify some internal functions not visible to the outside world. (Cort 2002).

## 2.5 NI LabVIEW

LabVIEW is a development environment from National Instruments that is extensively used in electronics testing. It uses a graphical programming language called G. The programming language is one of the major differences from many other development environments. G uses a dataflow model instead of sequential lines of text code. This makes the thought process of writing code quite different from conventional scripting languages and usually needs some time getting used to. LabVIEW is the preferred development environment in the testing industry as it provides technology for acquiring data and processing signals, instrument control and for developing custom interfaces.

One of the key benefits of working with LabVIEW is the extensive library of pre-made functions, which makes getting started with new projects and code faster and more straight forward. In a testing environment where customisation and adaptability is important this is a tremendous advantage.

LabVIEW programs are called Virtual Instruments, or Vis, and consists of a front panel, block diagram and a connector panel. The front panel represents the user interface and is tied to the block diagram where the code is written. The connector panel is where input and output data are routed to and from the VI when the VI is called from another VI. Figure 2 shows the basic development environment of LabVIEW.

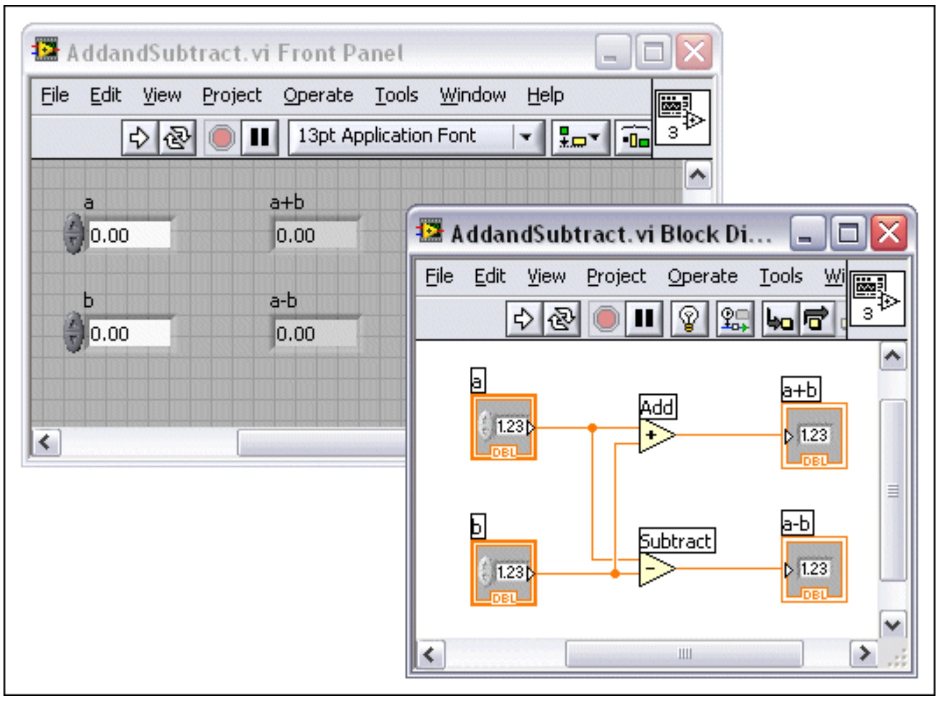

AddandSubtract.vi Front Panel
File Edit View Project Operate Tools Window Help
13pt Application Font
a
a+b
b
a-b
0.00
AddandSubtract.vi Block Di...
Add
Subtract
1.23
DBL

**Figure 2. The front panel and block diagram in LabVIEW (National Instruments 2013).**

Furthermore, like in other software development environments, there is functionality for debugging code by executing the code using breakpoints for example. If there are broken wires, one can use an error list showing the missing connections. When executing code, it is also possible to do a highlighted execution, which means that the data flow of the executing code is displayed. Another function for debugging is the probe tool which let you probe values in the code as it runs. (National Instruments 2013).

## 2.6 NI Teststand

Teststand is a test management software for test automation made by National Instruments. It includes a ready-to-run sequence engine that supports multiple test code languages, result reporting and parallel or multithreaded tests. (National Instruments 2016a).

One of the strengths of Teststand is the pre-built functionality and the ability for customization. It is used in all industries where automated testing is needed.

### 2.6.1 Software components

The Teststand engine is as the name suggests what drives the test system. It consists of a set of DLL files that exports an ActiveX Automation server API for other software components to communicate with the engine. Other applications, acting as ActiveX clients, can get or set properties and invoke methods on the server to control the engine through the API.

The sequence editor is the application in which the sequence and test system development is done. In the sequence editor it is possible to create, edit, execute, and debug sequences. Other features that can be accessed through the sequence editor are process models and station settings. As in any other application development environment such as LabVIEW and Microsoft Visual Studio it is possible to set breakpoints, trace through program executions and monitor variables for debugging purposes.

The user interface is an application which is used on the deployed system. It is used by the end user to control and monitor the tester. Teststand is shipped with pre-built user interfaces developed in all the supported test code languages. The source codes for the pre-built user interfaces are available so they are fully customizable. It is also possible to create a user interface from scratch containing only the desired functions.

The deployment utility creates an image and an installation file of the developed system containing all the necessary files for a functional deployment of the test system on another computer. Through a simplified GUI the developer can select all the files that need to be included and the deployment utility takes care of the rest.

### 2.6.2 Building blocks

**Steps**

A step is much like a line in a scripting language. They are individual elements of a test sequence that call a code module or perform some other operation. For example, a step can call a code module that initializes an instrument, makes a measurement, call a subsequence or jump to another step. Steps include properties which for example can specify what parameters to send to code modules and where to store parameters received from code modules. There are many pre-defined steps in Teststand to accommodate for the most frequently used tasks. The users can also store their own template steps.

**Sequences**

A sequence file contains a main sequence with a series of test steps. One of these steps can be a call for a subsequence. Other properties that sets a main sequence from a subsequence is that a main sequence includes sequence file global variables.

All sequences contain a setup, a main and a clean-up section which are called step groups. Steps are placed into the step groups to create a test. There is no significant difference between the sections but the clean-up section can be used for steps that needs to be executed in all scenarios. For example, if a step fails in the main section it is possible to configure the sequence to jump straight to the clean-up section. In general, the setup section is used for test setup functions like initializing instruments. The main section is used for the actual test steps like measuring a voltage and checking it against limits. The clean-up section is for example where steps that close communication with instruments are positioned.

**Code modules**

Code modules are program modules developed in an application development environment like LabVIEW or other programming language. The called code module performs a test or other action.

**Properties**

Properties consists of values that can be a number, string, Boolean, .NET object reference, or ActiveX object reference. For example, sequence properties and step properties can be modified programmatically each time a sequence is run with a property loader step. This means that limits for measurements and other properties in the sequence can easily be changed on the deployed Teststand system.

**Variables**

Variables are properties freely definable in a certain context. A variable can be global for the whole test station or local only for a specific subsequence and everything in between. Variables can be used to share data between steps and sequences and can be accessed through the Teststand API from code modules.

**Expressions**

Expressions are formulas that van calculate new values from one or more variables and properties. One can insert an expression as a separate expression step, before or after a step has executed, or as a property or variable. Expression operators and syntax that Teststand supports are the same as in C, C++, Java, and Visual Basic .NET.

A simple example of an expression:

Locals.Humidity = Step.Measurment * 100

Here the relative humidity is measured by a sensor that outputs a voltage between 0 and 1 volts. By multiplying a DMM measurement the scaled humidity value is acquired.

### Process models

The process model is a sequence file that specifies certain aspects of how the testing process works on a certain Teststand system. For example, in addition to the test sequence the testing process may include DUT identification, result indication, result logging and report generation. Teststand is shipped with built-in sequential, parallel and batch process models. These process models can be customized for specific test platform needs.

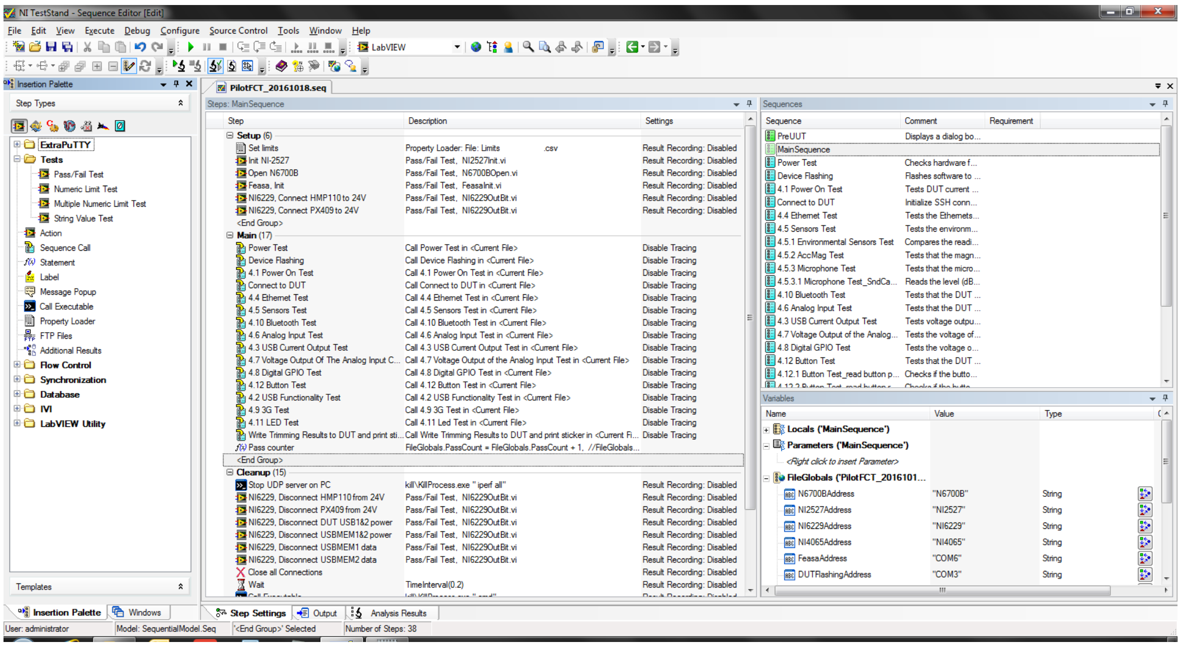
### Callback sequences

Callback sequences are used to define the behaviour of a test station when a specific event occurs. Callbacks are divided into three groups: model callbacks, engine callbacks and front-end callbacks. The groups are defined by the software component that invokes the callback and where the callbacks are defined. Model and engine callbacks are implemented in a test sequence by adding them through the Sequence File Callbacks dialog box. Front-end callbacks are called from the user interface and are located in FrontEndCallbacks.seq file.

Model callbacks are used to override the current process model's behaviour. For example, one can define a model callback to change how Teststand generates the report for that specific test sequence. Engine callbacks are used to invoke a callback at specific points during a test. For example, it is possible to define an engine callback to log to a database after every step in a test sequence. Front-end callbacks are used to run a specific sequence when a certain action is triggered by a user through the user interface. Features that do not depend on the process model are implemented as front-end callbacks. (National Instruments, 2016b).

### 2.6.3 Sequence editor GUI overview

The sequence editor is the main software component that is used during development of a Teststand system. Figure 3 shows the GUI of the sequence editor.



| Step | Description | Settings |
|---|---|---|
| **Setup** (6) | | |
| Set limits | Property Loader: File: Limits .csv | Result Recording: Disabled |
| Init NI-2527 | Pass/Fail Test, NI2527Init.vi | Result Recording: Disabled |
| Open N6700B | Pass/Fail Test, N6700BOpen.vi | Result Recording: Disabled |
| Feasa, Init | Pass/Fail Test, FeasaInit.vi | Result Recording: Disabled |
| NI6229, Connect HMP110 to 24V | Pass/Fail Test, NI6229OutBit.vi | Result Recording: Disabled |
| NI6229, Connect PX409 to 24V | Pass/Fail Test, NI6229OutBit.vi | Result Recording: Disabled |
| <End Group> | | |
| **Main** (17) | | |
| Power Test | Call Power Test in <Current File> | Disable Tracing |
| Device Flashing | Call Device Flashing in <Current File> | Disable Tracing |
| 4.1 Power On Test | Call 4.1 Power On Test in <Current File> | Disable Tracing |
| Connect to DUT | Call Connect to DUT in <Current File> | Disable Tracing |
| 4.4 Ethernet Test | Call 4.4 Ethernet Test in <Current File> | Disable Tracing |
| 4.5 Sensors Test | Call 4.5 Sensors Test in <Current File> | Disable Tracing |
| 4.10 Bluetooth Test | Call 4.10 Bluetooth Test in <Current File> | Disable Tracing |
| 4.6 Analog Input Test | Call 4.6 Analog Input Test in <Current File> | Disable Tracing |
| 4.3 USB Current Output Test | Call 4.3 USB Current Output Test in <Current File> | Disable Tracing |
| 4.7 Voltage Output Of The Analog Input C... | Call 4.7 Voltage Output of the Analog Input Test in <Current File> | Disable Tracing |
| 4.8 Digital GPIO Test | Call 4.8 Digital GPIO Test in <Current File> | Disable Tracing |
| 4.12 Button Test | Call 4.12 Button Test in <Current File> | Disable Tracing |
| 4.2 USB Functionality Test | Call 4.2 USB Functionality Test in <Current File> | Disable Tracing |
| 4.9 3G Test | Call 4.9 3G Test in <Current File> | Disable Tracing |
| 4.11 LED Test | Call 4.11 Led Test in <Current File> | Disable Tracing |
| Write Trimming Results to DUT and print sti... | Call Write Trimming Results to DUT and print sticker in <Current Fi... | Disable Tracing |
| Pass counter | FileGlobals.PassCount = FileGlobals.PassCount + 1, //FileGlobals... | |
| <End Group> | | |
| **Cleanup** (15) | | |
| Stop UDP server on PC | kill\KillProcess.exe " iperf all" | Result Recording: Disabled |
| NI6229, Disconnect HMP110 from 24V | Pass/Fail Test, NI6229OutBit.vi | Result Recording: Disabled |
| NI6229, Disconnect PX409 from 24V | Pass/Fail Test, NI6229OutBit.vi | Result Recording: Disabled |
| NI6229, Disconnect DUT USB1&2 power | Pass/Fail Test, NI6229OutBit.vi | Result Recording: Disabled |
| NI6229, Disconnect USBMEM1&2 power | Pass/Fail Test, NI6229OutBit.vi | Result Recording: Disabled |
| NI6229, Disconnect USBMEM1 data | Pass/Fail Test, NI6229OutBit.vi | Result Recording: Disabled |
| NI6229, Disconnect USBMEM2 data | Pass/Fail Test, NI6229OutBit.vi | Result Recording: Disabled |
| Close all Connections | | Result Recording: Disabled |
| Wait | TimeInterval(0.2) | Result Recording: Disabled |

| Sequence | Comment | Requirement |
|---|---|---|
| PreUUT | Displays a dialog bo... | |
| MainSequence | | |
| Power Test | Checks hardware f... | |
| Device Flashing | Flashes software to ... | |
| 4.1 Power On Test | Tests DUT current ... | |
| Connect to DUT | Initialize SSH conn... | |
| 4.4 Ethernet Test | Tests the Ethernets... | |
| 4.5 Sensors Test | Tests the environm... | |
| 4.5.1 Environmental Sensors Test | Compares the readi... | |
| 4.5.2 AccMag Test | Tests that the magn... | |
| 4.5.3 Microphone Test | Tests that the micro... | |
| 4.5.3.1 Microphone Test_SndCa... | Reads the level (dB... | |
| 4.10 Bluetooth Test | Tests that the DUT ... | |
| 4.6 Analog Input Test | Tests that the DUT ... | |
| 4.3 USB Current Output Test | Tests voltage outpu... | |
| 4.7 Voltage Output of the Analog... | Tests the voltage of... | |
| 4.8 Digital GPIO Test | Tests the voltage o... | |
| 4.12 Button Test | Tests that the DUT ... | |
| 4.12.1 Button Test_read button p... | Checks if the butto... | |

| Name | Value | Type |
|---|---|---|
| **Locals ('MainSequence')** | | |
| **Parameters ('MainSequence')** | | |
| *<Right click to insert Parameter>* | | |
| **FileGlobals ('PilotFCT_2016101...** | | |
| N6700BAddress | "N6700B" | String |
| NI2527Address | "NI2527" | String |
| NI6229Address | "NI6229" | String |
| NI4065Address | "NI4065" | String |
| FeasaAddress | "COM6" | String |
| DUTFlashingAddress | "COM3" | String |

User: administrator | Model: SequentialModel.Seq | '<End Group>' Selected | Number of Steps: 38

**Figure 3. Teststand GUI of the sequence editor.**

The steps pane is located in the middle of the screen in Figure 3. This is where the individual steps for the selected sequence are displayed. The sequence pane is located in the top right corner and shows all the sequences of the sequence file. The insertion palette is located to the left in the screen. The insertion palette is divided into two windows: step types list and templates list.

# 3 Development of the functional tester

As stated earlier, the tester hardware was already done when the thesis work commenced. Online resources where extensively used for information and guidance where given by colleagues at Ampner throughout the work. The work of developing and implementing the tester software could be divided into three main parts: instrument drivers, development of test sequence and commissioning.

## 3.1 Device Under Test

When developing a test or a tester it is important to understand the DUT as well as possible. A short explanation of the DUT will be given, but specific details about the DUT will not be included in this thesis due to the nature of the work. The DUT is a diagnostics and communication device. All the functions of the device are to be tested plus a button and all the LEDs on the device. Functions of the device are:

- 3G modem
- Bluetooth
- USB
- Ethernet interface
- digital GPIO
- accelerometer
- magnetometer
- temperature
- relative humidity
- air pressure
- sound
- analog inputs and outputs

When the device is ready for final testing it will be fully assembled except an identification label, which will be added when testing is finished. All connections to the device will be done through the existing connectors on the device.

The device's software is a Linux operating system which can be accessed via the Ethernet port by SSH connection during testing. Several test programs are available on the device to simplify sensor reading and control of functions. After testing the ability to connect to the DUT are restricted for security reasons.

## 3.2 Tester hardware

The tester hardware is based on a Testcom product called CUBE. It is a modular 19-inch rack with a removable fixture on top. The scope of this thesis does not include the tester hardware but some of it will be explained for a better understanding of the tester software. Many devices and mechanical solutions where implemented so that the test would be automatic and require minimal actions from the operator. A mechanical drawing of the test station is shown in Figure 4.

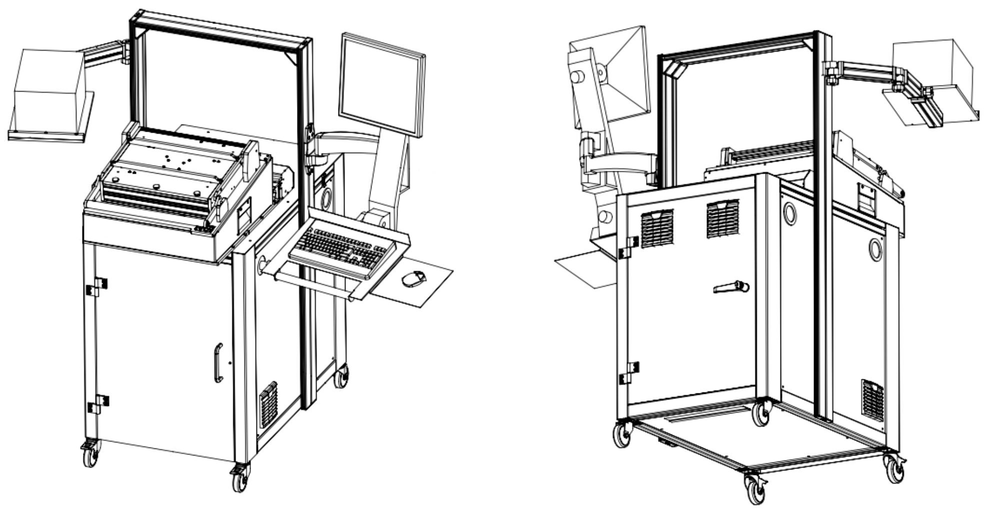


**Figure 4. CUBE test station (Ampner 2016).**

The modular tester hardware is divided into three main parts: test station, test fixture and test cassette. The reason for the modular design is to allow for easy maintenance and the possibility of testing several different products on the same test station. The test station is the main part of the tester and includes most of the instruments and testing devices. The test fixture, shown in Figure 5, connects to the test station through a connector interface which allows for removing the test fixture in a matter of seconds. The test fixture includes DUT specific test equipment.

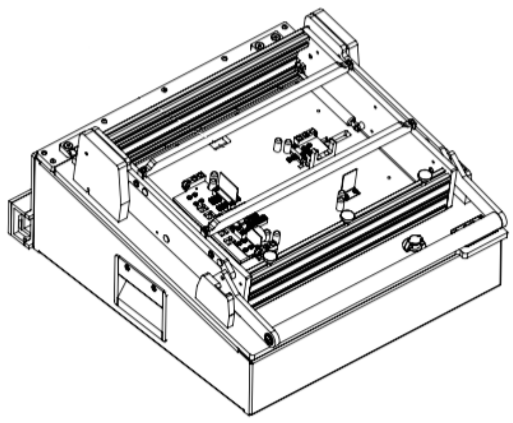


**Figure 5. Test fixture (Ampner 2016).**

Inside the test fixture is the test cassette, shown in Figure 6, which includes the connectors and mechanics for connecting to the DUT. On this specific final functional tester, the DUT connectors are located on the side of the device. The test connectors need to push in towards the device to connect. This is accomplished by having the connectors sliding on radial bearings when the lid of the fixture is closed.

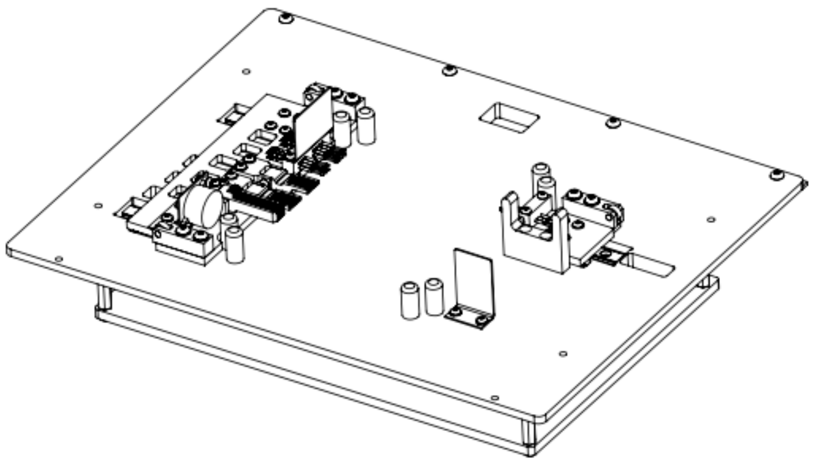


**Figure 6. Test cassette (Ampner, 2016).**

The instruments performing the measurements and control of the test are:

- NI PXI-1033 – MXI Express controller
  - NI PXI-4065 – 6 1/2-Digit PXI DMM
  - NI PXI-2527 – 64-Channel 300V CAT I Multiplexer
  - NI PXI-6229 – 32AI, 48DIO, 4AO DAQ
- Agilent N6700B – Modular Power System Mainframe, with N6745B and N6743B installed
- Feasa 10F – LED analyser

Other devices in the tester are:

- PC
- 3D barcode reader
- Label printer
- Bluetooth Sensor Tag
- Signal multiplexers and relay cards
- Humidity and temperature probe for reference
- Barometric pressure transducer for reference
- Buzzer
- Solenoid for pressing DUT button
- USB flash drives

## 3.3 Tester software

All the hardware on the tester is controlled by software on a Windows PC located in the testers frame. The test sequence is controlled and monitored through NI Teststand. As this was a pilot tester the development took place on the actual final PC. Normally test development would be done on a separate PC and then deployed to the tester.

SSH communication with the DUT is done through a command-line interpreter program called ExtraPuTTY. Other software used in the test sequence consists of executable scripts for sending data to the sticker printer and sending ping requests to the DUT.

The tester requirements were used as a basis for the development of the tester software. Among other things it specified a set of functional test cases that needed to be performed by the tester. The purpose, procedure and expected results were specified in the short test case specifications.

### 3.3.1 Instrument drivers

The instrument drivers had to be written or modified to work with the tester. All drivers, except the printer driver, were implemented in G code. Most of the drivers had been used in other projects and only needed minor adjustments so that they would be executable from within Teststand.

The purpose of an instrument driver is to simplify the development of a test sequence. The instruments may have different interfaces to the tester PC. An instrument driver is a piece of code that takes commands and values as inputs and returns one or several results from the instrument. If all instrument drivers in a tester have similar input and formats it simplifies the sequence development.

As an example, a power supply might be connected to the computer through a serial interface. To set the voltage of one of the channels one would have to initialize a serial connection, send the command, wait for a response and close the connection. Different protocols and commands might be needed for each instrument. Making the instrument drivers behave like each other, means that the test sequence development becomes easier.

### 3.3.2 Sequential structure in Teststand

In this case a sequential process model and a mostly sequential sequence structure were implemented. With a parallel process model, it is possible to execute tests on several DUTs at once. The reason for the sequential process model in this case, was the added cost of hardware for connecting several DUTs to the tester at once. The decision was made to go with a sequential sequence structure as this seemed to inflict less problems with the DUT communication.

When running a test, it is possible to let the user interface trace the steps currently executing. But the user interface that comes with Teststand can only show one thread at a time. This may be a problem when using several threads in the sequence execution, meaning that the sequence is executing several steps in parallel. This might be perceived as a confusing feature for the operator. And thus, another reason why the sequential sequence structure was chosen. Figure 7 shows the two different execution structures considered for this project.

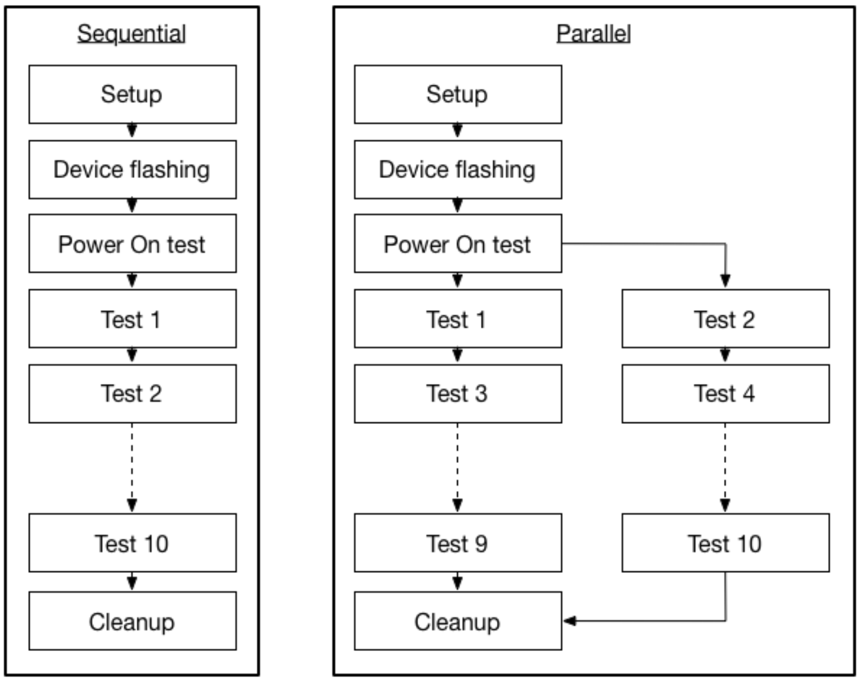


**Figure 7. Sequential and parallel execution structure.**

### 3.3.3 Timing of tests and test duration optimisation

Because the tester will be used to test a high volume of devices it is desirable to keep the test duration as short as possible. The desired complete test duration was one minute. With Teststand there are many ways to get the test duration down. But there is always a risk that the test becomes more unreliable if the sequence is over optimized for time. For example, some data was lost when multithreading was used while communicating with the DUT. This over optimization likely increased the workload of the tester and DUT. The sequence was modified to look out for the data loss to prevent the issue.

Further the test duration was optimized by arranging the order of the test cases so that it suited the DUTs boot up procedure. The test cases that did not need the DUT to be fully booted was set to be performed during boot up. The modem in the DUT used during the 3G test took around 45 seconds to be ready after the power had been applied to the DUT. This meant that the 3G test was performed at the end of the sequence to avoid having the sequence waiting on the modem to be ready.

### 3.3.4 Optional structures

Multithreading means that a CPU or a single core are used to run tasks concurrently. Multithreading in Teststand can be done in many ways but one way that was investigated and used in this thesis was to call subsequences and configuring the call to be started in a new thread. This means that the called subsequence is executed in parallel with the sequence that called it.

Some of the test cases required no instrument use, for example the USB functionality tests, Bluetooth test and 3G test. It was investigated if these could be run in parallel to the rest of the test cases to get a shorter test duration. But as mentioned earlier, this created some reliability issues to the test sequence. Multithreading functionality was only used for a short push button test on the pilot tester.

### 3.3.5 Data acquisition

The majority of the tester functionality depend on acquiring data from instruments, sensors and the DUT. The data is then handled in LabVIEW VIs and in the sequence. One of the more trivial measurements in this particular tester is acquiring data from a pair of reference sensors to be compared with the DUT environmental sensors. The reference sensors give out a voltage value with the sensor reading. This value is measured with the DMM in the tester and then it is scaled in the sequence so that it can be compared with the DUT output.

There are many occasions when a request is sent to the DUT to return data. This can be modem identification parameters or test data from one of the many built in test programs on the DUT. Two different approaches where used. Either the program or command was executed and the result printed directly to the command-line and recorded into the sequence. Another approach used was to run the test program as a background process on the DUT and save the result as a .txt file which data was copied into the sequence at a convenient time.

In the sequence variables and parameters were kept as local as possible. For example, if a variable in a subsequence were not needed in a higher-level sequence, it only existed in that subsequence. But some variables and parameters were needed in a subsequent sequence and then they needed to be transferred between the sequences. This was done by changing the settings of the sequence call.

### 3.3.6 Report generation and the property loader

Teststand has built in report generation, so it is easy to create a basic test report. Result collection, which collects step results and relevant data, is performed automatically during the execution of a sequence. Teststand then automatically generates a report file at the end of the sequence. The standard test report can be modified in many different ways. It is for example possible to choose which step data should be included in the report. It is also possible to change the layout, design, format and at what conditions reports are generated. For this pilot tester, the contents of the report were still not clear during the work on the thesis so the test report was only used and modified to aid the development process. The database logging was later implemented by the customer using their own customised solution.

The absolute first step in the sequence is a property loader step. This step lets the user change limits and other properties of a test easily by editing a .csv file on the tester PC. This feature is critical during commissioning and when the tester is used in production. If the limits set during development of the test turns out to be wrong or just slightly off from the DUTs current specifications, it is much simpler to be able to edit one .csv file on a server instead of ten different sequence files in ten different locations.

## 3.4 Implementation and commissioning

By running the sequence in small parts on the assembled tester it was possible to identify issues and solve them one by one. Because the DUT itself was still being developed communication with the DUT developers where critical to ensure that both the DUT and the tester were working as they were intended to.

To get the test sequence work as a whole the structure was modified slightly until it worked with the DUT boot up procedure. Other aspects that where considered when modifying the sequence at this state were test duration optimisation, limitations of the DUT and limitations of the tester.

The reliability and accuracy of the test sequence was tested by doing large numbers of test runs. Modifications were made when problems were encountered and the sequence was tested again.

When enough confidence had been gained in the reliability and functionality of the tester it was shipped to the customer. The customer then continued to implement their own solutions on the test station. Some modifications where requested to deal with issues raised after commissioning. This meant travelling to the customer's location and performing minor mechanical modifications. During this meeting, the tester was reviewed and issues that needed to be followed up on when the tester was taken into use in production were pointed out.

# 4 Result

The developed test sequence consists of one sequence file containing 23 sub sequences. In Figure 8 the complete test sequence hierarchy is shown.

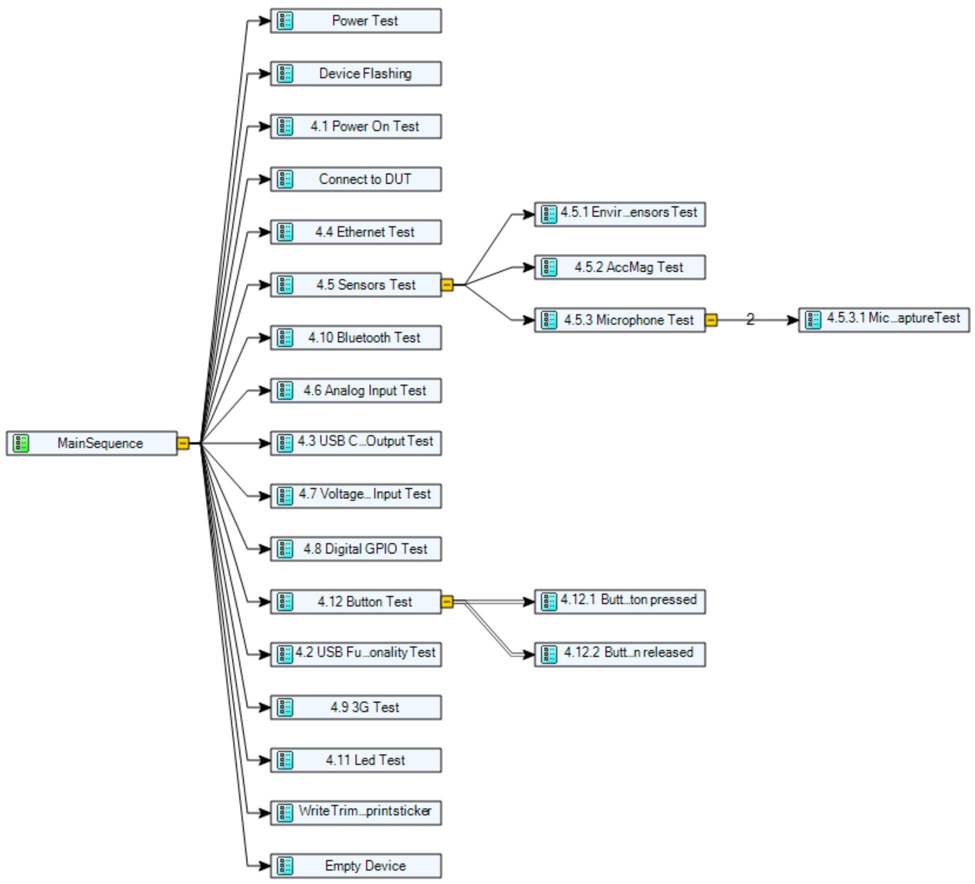
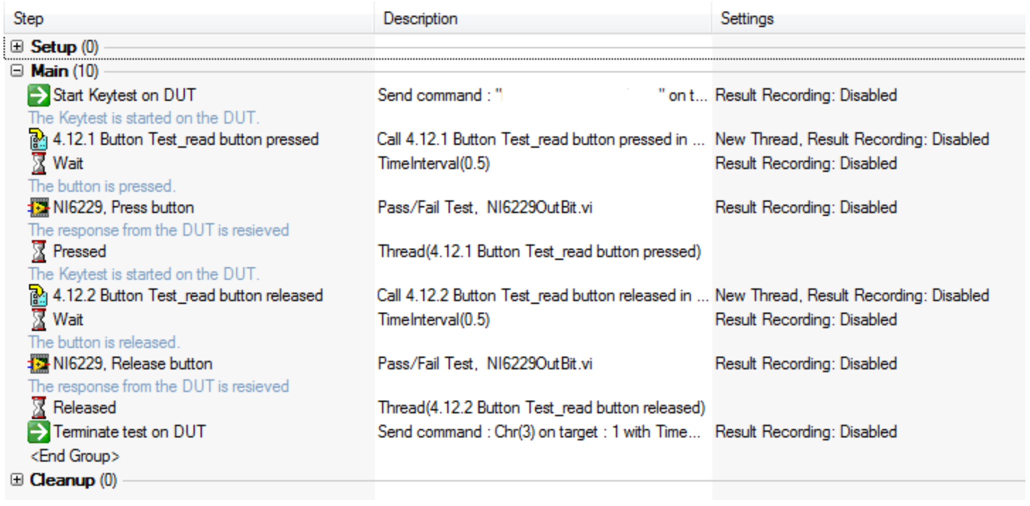


**Figure 8. Sequence hierarchy.**

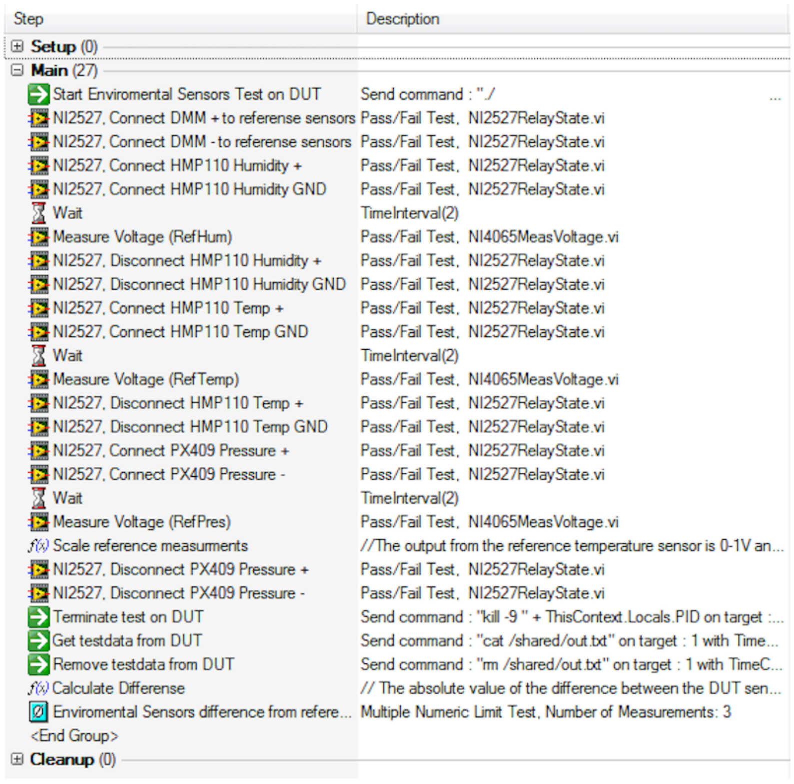
The sequences starting with a number, e.g. 4.1, are test cases specified in the tester requirements specification. The other sequences are for housekeeping and other required functionality. All sub sequences are executed sequentially except 4.12.1 and 4.12.2. These two sequences are executed as new threads. This was done so that it is possible to push the button and read the button state on the DUT, both at the same time. The subsequence 4.12 is shown in Figure 9.



| Step | Description | Settings |
|---|---|---|
| ⊞ **Setup** (0) | | |
| ⊟ **Main** (10) | | |
| Start Keytest on DUT | Send command : " " on t... | Result Recording: Disabled |
| The Keytest is started on the DUT. | | |
| 4.12.1 Button Test_read button pressed | Call 4.12.1 Button Test_read button pressed in ... | New Thread, Result Recording: Disabled |
| Wait | TimeInterval(0.5) | Result Recording: Disabled |
| The button is pressed. | | |
| NI6229, Press button | Pass/Fail Test, NI6229OutBit.vi | Result Recording: Disabled |
| The response from the DUT is resieved | | |
| Pressed | Thread(4.12.1 Button Test_read button pressed) | |
| The Keytest is started on the DUT. | | |
| 4.12.2 Button Test_read button released | Call 4.12.2 Button Test_read button released in ... | New Thread, Result Recording: Disabled |
| Wait | TimeInterval(0.5) | Result Recording: Disabled |
| The button is released. | | |
| NI6229, Release button | Pass/Fail Test, NI6229OutBit.vi | Result Recording: Disabled |
| The response from the DUT is resieved | | |
| Released | Thread(4.12.2 Button Test_read button released) | |
| Terminate test on DUT | Send command : Chr(3) on target : 1 with Time... | Result Recording: Disabled |
| <End Group> | | |
| ⊞ **Cleanup** (0) | | |

**Figure 9. Button test sequence including execution in new thread of subsequence 4.12.1 and 4.12.2.**

The 4.5 Sensors Test is shown in Figure 10. It was written to first start the test program on the DUT. The test program continuously outputs humidity, temperature and air pressure to a .txt file. When the test program has been started, the tester measures the voltage output of the reference sensors located close to the DUT in the test cassette. The voltages are scaled to represent the same units as the DUT readings. This process is timed so that enough readings from the DUT can be acquired. The difference between the reference and DUT sensors are tested against expected values.



| Step | Description |
|---|---|
| ⊞ **Setup** (0) | |
| ⊟ **Main** (27) | |
| Start Enviromental Sensors Test on DUT | Send command : "./ ... |
| NI2527, Connect DMM + to referense sensors | Pass/Fail Test, NI2527RelayState.vi |
| NI2527, Connect DMM - to referense sensors | Pass/Fail Test, NI2527RelayState.vi |
| NI2527, Connect HMP110 Humidity + | Pass/Fail Test, NI2527RelayState.vi |
| NI2527, Connect HMP110 Humidity GND | Pass/Fail Test, NI2527RelayState.vi |
| Wait | TimeInterval(2) |
| Measure Voltage (RefHum) | Pass/Fail Test, NI4065MeasVoltage.vi |
| NI2527, Disconnect HMP110 Humidity + | Pass/Fail Test, NI2527RelayState.vi |
| NI2527, Disconnect HMP110 Humidity GND | Pass/Fail Test, NI2527RelayState.vi |
| NI2527, Connect HMP110 Temp + | Pass/Fail Test, NI2527RelayState.vi |
| NI2527, Connect HMP110 Temp GND | Pass/Fail Test, NI2527RelayState.vi |
| Wait | TimeInterval(2) |
| Measure Voltage (RefTemp) | Pass/Fail Test, NI4065MeasVoltage.vi |
| NI2527, Disconnect HMP110 Temp + | Pass/Fail Test, NI2527RelayState.vi |
| NI2527, Disconnect HMP110 Temp GND | Pass/Fail Test, NI2527RelayState.vi |
| NI2527, Connect PX409 Pressure + | Pass/Fail Test, NI2527RelayState.vi |
| NI2527, Connect PX409 Pressure - | Pass/Fail Test, NI2527RelayState.vi |
| Wait | TimeInterval(2) |
| Measure Voltage (RefPres) | Pass/Fail Test, NI4065MeasVoltage.vi |
| Scale reference measurments | //The output from the reference temperature sensor is 0-1V an... |
| NI2527, Disconnect PX409 Pressure + | Pass/Fail Test, NI2527RelayState.vi |
| NI2527, Disconnect PX409 Pressure - | Pass/Fail Test, NI2527RelayState.vi |
| Terminate test on DUT | Send command : "kill -9 " + ThisContext.Locals.PID on target :... |
| Get testdata from DUT | Send command : "cat /shared/out.txt" on target : 1 with Time... |
| Remove testdata from DUT | Send command : "rm /shared/out.txt" on target : 1 with TimeC... |
| Calculate Differense | // The absolute value of the difference between the DUT sen... |
| Enviromental Sensors difference from refere... | Multiple Numeric Limit Test, Number of Measurements: 3 |
| <End Group> | |
| ⊞ **Cleanup** (0) | |

**Figure 10. 4.5 Sensors Test sequence.**

### 4.1.1 Operating the tester

The steps involved with operating the pilot functional tester are:

1. The test station is powered on and the Teststand user interface is started.
2. The user logs in to Teststand with user credentials.
3. The sequence can be started in two different modes, either running a series of multiple tests or just a single pass. If a series of multiple tests is chosen steps four to eight are looped.
4. The barcode reader is used to scan the batch and serial number of the DUT and the DUT is then placed in the fixture.
5. After this point the test sequence is automatic and the operator does not have to intervene.
6. The test result is showed clearly on screen and a result sticker is printed.
7. The test data is loaded to a database in the background.
8. The sticker is attached to the DUT and the DUT is placed into the correct location according to the test result.

### 4.1.2 Test duration

In early sequence development, the duration of the different stages of the test sequence were estimated to be:

- 30 seconds for device flashing
- 15 seconds for booting the device
- 45 seconds for testing
- 90 seconds in total

The shipped sequence had longer times in every category. Most of the time added to the test duration was delays in the code to make sure that no errors occurred. The test duration was longer than desired when the tester was delivered to the customer due to SSH and USB communication reliability issues. At the end of the thesis work the actual times were around:

- 40 seconds for device flashing
- 30 seconds for booting the device
- 70 seconds for testing
- 130 seconds in total

The test duration was deemed sufficient for the moment and were to be optimized after further development.

## 4.2 Unresolved issues/DUT functions not tested

The hysteresis function of the digital GPIO input was tested in the GPIO test. This was done by incrementally increasing and then decreasing the voltage applied to the input between 1 and 5 volts, as shown in Table 2.

**Table 2. GPIO Hysteresis test.**

| Step | Applied voltage | Expected value |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 2 | 0 |
| 3 | 5 | 1 |
| 4 | 1 | 1 |
| 5 | 0 | 0 |

But during a couple of percent of test executions, the recorded value in the sequence on steps 3 or 4, were 0 instead of the expected 1. This issue was examined by the tester developers and the DUT developers to understand what the reason for the sporadic failures was. The issue remained unresolved during the thesis work and was set to be followed up as the tester was applied in production.

The other unresolved issue was interference in the high-speed USB signals in the tester. Because of the limited number of USB ports on the DUT, USB signals needed to be switched through relay cards when changing the connections. During implementation of the test sequence it became apparent that the connections between relay cards, tester PC, USB memory devices and the DUT were not sufficient in keeping the communication confidently stable. The USB test connectors were of the type in Figure 11, which are of high quality and made to withstand a high number of mating cycles, but they do not connect the shield of the USB cable. Neither the USB multiplexers in the tester had the shields connected resulting in hard to trace errors during device flashing and USB functionality tests.

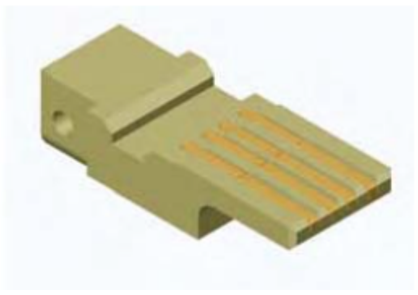


**Figure 11. UBS test connector (Engmatec 2013).**

The issue was reduced by changing cabling. The issue was tolerable on the commissioned tester, but further investigation into alternative connectors and USB multiplexers will be done in the future.

## 4.3 Customer approval

To verify that a stable process works according to specifications the process capability can be analysed. The customer used $C_p$ and $C_{pk}$ when verifying the tester. $C_p$ and $C_{pk}$ are statistical indices that indicate how well the output of the process sits inside the specified limits. Gage R&R analysis was also performed to assess the measurement precision of the tester. Gage R&R gives statistics like repeatability, reproducibility and residual or pure error for the process. (NIST/SEMATECH 2012). The tester was approved by the customer and taken into use in production at the end customer.

# 5 Conclusion

The thesis work resulted in a tester with hardware and software functioning according to the tester requirements specification. The issues that arose and that where not resolved during the work were not critical and improvements will be made in the future to resolve these issues.

Throughout the development and implementation process of the tester the scope of the thesis work was redefined slightly. This was possible because it was a pilot tester and the customer had some resources themselves to continue the development process after it had been commissioned. The reason for this flexible arrangement was the tight timeframe of the project. Although every wish the customer mentioned was not implemented, the delivered tester was successfully produced according to the customer's tester specifications in the defined timeframe.

## 5.1 Lessons learnt

Even though the thesis work was seen as a success there is always things that could have been done differently and lessons learnt. To communicate with the DUT a software called ExtraPuTTY was used as a SSH client. Secure Shell (SSH) is a protocol between a server and clients for IP networks. ExtraPuTTY was chosen because of its stated ability to work together with NI Teststand. When installing ExtraPuTTY .dll files are copied into Teststand folders so that ExtraPuTTY can be called with steps from Teststand. For this project ExtraPuTTY has been sufficient but not without many aggravating moments. The documentation of using ExtraPuTTY together with Teststand is very limited and most of the support has been found on forums online. The installation process was very complicated and unclear. Help for the installation of ExtraPuTTY can be found in the (NI forum 2013) link in the bibliography. ExtraPuTTY also had problems communicating with the DUT reliably. Sometimes the SSH client did not wait for the end of line character from the DUT resulting in sequence errors.

Another unexplainable incident involving ExtraPuTTY happened when the tester had been delivered to the customer. A report from ExtraPuTTY popped up after each test execution. This feature was finally turned off by setting a new file path for the reports and turning of ExtraPuTTY report generation in test steps. In ExtraPuTTYs defence it is a free software and most of the work is done for you but I would definitely suggest that investigation of alternatives would be beneficial if a SSH client with Teststand API is needed in the future.

Some of the test data was transferred from the DUT to the test sequence through the SSH connection by reading the print out in the terminal on the PC to the sequence. An alternative way was to let the Linux command be run in the background and save the print out into a .txt file on the DUT. The data was then transferred to the sequence at a later stage. The advantages of the second approach are that it frees up the terminal for other tasks and there is a smaller risk of ExtraPuTTY not waiting for the end of line character. It would have shortened the test duration and probably made the sequence more reliable if all tests using data from the DUT would have used the second approach, but due to time limitations the changes could not be made during the thesis work.

There are many different specification documents used in hardware testing. As in most areas of engineering, the quality of the specification documents often corresponds with the resulting product. To the word specification more words are added to specify the purpose of the document. The names of these documents are sometimes used interchangeably, which may cause confusion.

It could be argued that in this specific project the specifications was not comprehensive enough. The test requirements specification and the tester requirements specification were combined into one document. The functions to be tested were documented, but function specific measurements were not yet clearly defined because the DUT was still in development. I can imagine that, in general, the DUT is often not a finished product at the time of writing the tester requirement specification. But the development and commissioning of the tester can be done much faster and cheaper with clear specifications. On a functional tester like this at least all the required measurements and acceptable values should be clearly defined.

Another issue linked to the fact that the DUT was still in the development phase was the number of devices present during development of the tester. We only had one device available in the beginning of tester software development. This meant that it was critical for the project that the device did not break. Later a new version of the device was available as well. The second device helped in testing the tester but it would have definitely been preferable to have had a few copies of the same version of the device during the implementation and testing phase of the tester.

## 5.2 Pros and cons using Teststand as a test sequencer

There are several alternatives to using NI Teststand as the test sequencer. Ampner has developed and sustained a LabVIEW based test sequencer for many years. The advantages of using a test sequencer programmed in house are that it is independent and completely customizable. A deployed system does not necessarily need any NI licences to run, so it is possible to cut costs. It is also possible to include any imaginable feature that the customer wants. A LabVIEW based test sequences normally uses text based test scripts. It can be argued that it is easier to read and develop text based scripts than Teststand sequences.

A Teststand sequence can be strenuous to read especially when using many expressions. My personal experience of using Teststand has been very positive. The online forum support has been very helpful. As with other NI products, the manuals are quite extensive. A Teststand deployment needs a runtime licence to run. However, in regards to costing, the costs of developing and maintaining a completely customizable test sequencer also needs to be considered.

Other advantages of Teststand are the features already built in and the wide use of the software in the industry. Because it is well known the processes of marketing, sales, production and after sales can be aided by the fact that the customers are familiar with the development environment. With the free extensive support online it gives the customer the chance to problem solve and further develop the system if they so wish. All these advantages result in shorter development times and better competitiveness compared to a test sequencer built in house.

## 5.3 Further development

The tester developed in this thesis will be further tested in the production line at the end customer. A second copy, with improvements, of the tester has been commissioned. Because of this, the tester will continue to be further developed after the thesis work. As mentioned in Chapter 5.1, there are several areas that could be improved upon that will be investigated.

In relation to the topic of this thesis, the areas that seem to be mostly beneficial to further development are the USB connections in the tester, the self-test programs on the DUT, and the sequence structure. The USB connections need to be changed so that the connections are reliable without the interference earlier mentioned. The self-test programs could be modified and new programs could be implemented to aid the sequence, thus making the test faster and more reliable. For example, test results could automatically be written to a .txt file. The sequence structure could be adapted to the new DUT self-tests and multithreading could be used to shorten test duration. However, if the structure is changed and extensive multithreading functionality is introduced further development of the user interface will also be needed.

## 5.4 Comments

As expected time is always an issue. If a project does not have a deadline, it most likely will never be completed. This thesis work was no different. Throughout the project finding time to handle less than critical issues was hard. Extensive knowledge of testing hardware and LabVIEW coding were accessible in-house, but Teststand was completely new for me and had not been used as the primary test sequencer at the company. So, a great deal of precious time went to figuring out and learning the development environment. In hindsight, this was probably the fastest and most valuable way to move from and academic environment into the working life of an engineer.

# 6 References

Agilent Technologies, 2013. *Test-System Development Guide* [Online] http://cp.literature.agilent.com/litweb/pdf/5989-5367EN.pdf [Retrieved: November 8, 2016].

Ampner, 2016. *Internal document.* [Retrieved: January 10, 2017].

Ampner, 2017. *The Power Systems Architects in Renewables.* [Online] https://wordpress-ampner.appcloud.jubic.net/wp-content/uploads/2017/03/Ampner-The-Power-System-Architects-in-Renewables_web.pdf [Retrieved: May 14, 2017].

Cort, A., 2002. *Functional Testing of PCBs.* [Online] http://www.assemblymag.com/articles/83988-functional-testing-of-pcbs [Retrieved: May 14, 2017].

Edwards, P.R., 1991. *Manufacturing Technology in the Electronics Industry*. Suffolk: St Edmundsbury Press.

Engmatec, 2013. *Testing plugs.* [Online] https://www.engmatec.de/fileadmin/content/ENGLISCH/PDF-Medien/Testing-Plugs_2013.pdf [Retrieved: May 14, 2017].

ISO 9000:2015. *Quality management systems — Fundamentals and vocabulary.* [Online] https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-4:v1:en [Retrieved: May 14, 2017].

Koenemann, B., 2006. Design-For-Test. in: L. Lavagno, G. Martin & L. Scheffer ed. *Electronic Design Automation for Integrated Circuits Handbook.* Vol 2 21-1-21-35. New York: Taylor & Francis.

National Instruments, 2013. *Getting Started With TestStand.* [Online] http://www.ni.com/pdf/manuals/373436f.pdf [Retrieved: November 8, 2016].

National Instruments, 2016a. *What is Teststand*. [Online] http://www.ni.com/teststand/ [Retrieved: November 15, 2016].

National Instruments, 2016b. *Using Callbacks in NI Teststand*. [Online] http://www.ni.com/product-documentation/6605/en/ [Retrieved: November 16, 2016].

NI forum, 2013. *Starting TestStand with ExtraPuTTY installed*. [Online] http://forums.ni.com/t5/NI-TestStand/Starting-TestStand-with-ExtraPuTTY-installed/td-p/2596877 [Retrieved: October 25, 2016].

NIST/SEMATECH, 2012. *E-Handbook of Statistical Methods.* [Online] http://www.itl.nist.gov/div898/handbook [Retrieved: May 14, 2017].

Poole, I., n.d. a *Automatic test equipment ATE*. [Online] http://www.radio-electronics.com/info/t_and_m/ate/automatic-test-equipment-basics.php [Retrieved: May 14, 2017].

Poole, I., n.d. b *ICT, In Circuit Test Tutorial*. [Online] http://www.radio-electronics.com/info/t_and_m/ate/ict-in-circuit-test-tutorial.php [Retrieved: May 14, 2017].

Stenbacka, B., Ampner – nytt Vasaföretag som växer. *Vasabladet,* 20.12.2016, page 1 & 9.

Tarr, M., n.d. *ICT Fixtures.* [Online] http://www.mtarr.co.uk/courses/topics/0251_fixt/index.html [Retrieved: April 13, 2017].

Texas Instruments Inc., 1997. *IEEE Std 1149.1 (JTAG) Testability Primer.* [Online] http://www.ti.com/lit/an/ssya002c/ssya002c.pdf [Retrieved: May 14, 2017].