CHALMERS
UNIVERSITY OF TECHNOLOGY

Remote Control of Test Stations

Software solutions in TestStand

Bachelor's thesis in Electrical engineering

Kevin Johansson
Henrik Kahl

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2015

BACHELOR'S THESIS 2015

Remote Control of Test Stations

Software solutions in TestStand

Kevin Johansson
Henrik Kahl

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2015

Remote Control of Test Stations
Software solutions in TestStand
KEVIN JOHANSSON AND HENRIK KAHL

Supervisor at Chalmers: Manne Stenberg, Chalmers University of Technology department of Signals and System
Supervisor at SAAB: Lennart Berlin, Specialist Microwave Measurement Technology
Examiner: Bertil Thomas, Chalmers University of Technology department of Signals and System

Bachelor's Thesis 2015
Department of Signals and Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2015

Remote Control of Test Stations
Software solutions in TestStand
KEVIN JOHANSSON AND HENRIK KAHL
Department of Signals and Systems
Chalmers University of Technology

Abstract

The problem that arise when testing transmission and receiver units for radar applications can be noted when a sensitive noise figure test is run concurrently with a high-power transmission test. A hardware solution to diminish the induced interference is costly, and a software solution is instead sought. This paper describes several solutions that attempts to tackle the problem that has been presented at SAAB's testing facility. The work has been limited to general solutions and does not cover the implementation of one (or several) solutions. The solutions presented herein provides a significant improvement in simulations over the current testing program put in place. Implementation of the solution will require redesigning the current testing program, and it is significant at larger scales to implement a priority functionality to the presented solution.

Keywords: SAAB, EDS, Radar, Distributed Testing, LabVIEW, TestStand, Master-Slave, Prioritisation.

Fjärrstyrning av teststationer
Mjukvarulösningar i TestStand
KEVIN JOHANSSON OCH HENRIK KAHL
Signaler och System
Chalmers tekniska högskola

Sammanfattning

Problem uppstår när man testar sändnings- och mottagarenheter för radartillämpningar då känsliga brusfaktormätningar genomförs samtidigt som högeffekts sändningstester. En hårdvarulösning för att minska inducerade störningar är dyrt, och en mjukvarulösning är istället eftersökt. Detta dokument beskriver flera lösningar som försöker ta itu med problemet som har presenterats vid SAABs testanläggning. Arbetet har begränsats till generella lösningar och omfattar inte implementation av en, eller flera lösningar. De lösningar som presenteras här ger en avsevärd förbättring vid simuleringar gentemot det nuvarande testprogrammet. Genomförande av lösningen kommer att kräva omkonstruktion av aktuellt testprogram, samtidig som stora vinster kan uppnås med en automatiserad testning vid en storskalig produktion.

Nyckelord: SAAB, EDS, Radar, Distribuerade test, LabVIEW, TestStand, Master-Slave, Prioritering

Acknowledgements

We would like to thank our supervisor at SAAB, Lennart Berlin for the support during this project. Our supervisor at Chalmers Manne Stenberg, and our examiner Bertil Thomas for their input. We would also like to thank Sakib Sistek for making this project possible for us, the personnel at National Instruments' for helping out with software support. And last but not least we would like to thank our colleagues at SAAB EDS for answering our questions.

Kevin Johansson and Henrik Kahl, Gothenburg, June 2015

Contents

List of Figures

Abbreviations

DUT	Device Under Test
EDS	Electronic Defense Systems
NI	National Instruments
RF	Radio Frequency
RFI	Radio Frequency Interference
UUT	Unit Under Test - same as DUT
VI	Virtual Instrument

1

Introduction

SAAB Group is a global company that serve both the military as well as the civil market with defense and security solutions within their six business areas. The SAAB Group has over 14'000 employees in over 100 countries.

This thesis is carried out within the business area of 'Electronic Defense Systems' (EDS) at the office in Kallebäck, Gothenburg.

1.1 Background

During the production of circuit boards for radar applications at SAAB's factory in Kallebäck a series of tests are carried out on each board. If one test station performs noise figure tests and another station decides to do transmission tests at the same time then the transmission tests will, because of their high output power, interfere with the noise figure tests on the first station.

All test stations run a specialised version of National Instruments' TestStand that has been developed for the sole purpose of testing equipment. TestStand is a software developed to create, manage and run automated test and verification systems. As well as a developer view, TestStand offers a user friendly interface for the test operator with possibilities to control the tests and run entire sequences or just part of it.

1.2 Purpose

The purpose of this thesis is to find a number of software solutions to control what types of tests that are allowed to run in the production and testing environment at any given moment.

Today the test engineers at SAAB manually redo tests, or accept the faulty ones even if they think they are muddled with noise due to the interference that transmission tests might induce; interference upon sensitive tests stems from performing high power tests at the same time. The goal of the project is, by setting up rules in the test executive and get the test stations to communicate with each other, to prevent transmission tests and noise figure tests to run at the same time.

1.3 Deliminations

The simplest solution to this project might be to improve the electromagnetic shielding around the units during tests, but since the tests are carried out very early during the production process of the circuit boards it is not possible. This project is therefore delimited to a software solution for the problem.

This project will only look at solutions based on a standard TestStand installation, but with some limitations set up by SAAB EDS and their testing environment.

1.4 Definition of task

- Is it at all possible to run tests in autonomous selected order?
- Is it possible to let a master computer synchronise which tests are allowed to run at the moment?
- Will this type of test execution improve the test results and save time compared to today's methods?

2
Technical Background

This chapter contains the theoretical background needed for this project. It will describe the functionality behind a radar system, why tests are distributed in a network, and what software is used in this project.

2.1 Radar

The general purpose of a radar system stems from the desire to detect objects in a certain direction/perimeter from the radar itself; this is of interest when it is essential to avoid collision, detect hidden objects, or attain a general overview of the surrounding landscape. Gaining such advantages can be beneficial in maritime, aerial and defense areas. But radar can also provide measurements within e.g. weather observations, and is as such a quite versatile tool.

As with most applications and equipment, the purpose and functions of it may appear simple and mundane, but more often than not it is built on a complex foundation. The following information will explain it all in layman's terms: from the bare essential parts, to the system as a whole during operation, and under what circumstances it may run.

For radar, the first step in achieving an image is to emit an electromagnetic energy pulse, a process that starts with the transmitter. [1]

2.1.1 Transmitter

The transmitter's job is to produce a waveform signal with enough average and mean Radio Frequency (RF) power, suitable RF bandwidth, stability, and reliability. For this project the focus have been on solid-state amplifiers used in conjunction with phased array antennas due to the product that SAAB utilises.

In this case the solid-state amplifiers consists of high power transistors. Each individual transmitter has low power, but combined they have the much larger quantity of power that is needed for radar application. This accumulated power can then be sent on to the antenna.

2.1.2 Duplexer

Due to the high power output of the transmitter during its pulses, it is necessary to implement a duplexer to protect the sensitive receiver and still retain the convenience of using only one antenna. This is achieved by a switch that is designed in the frequency band of the transceiver, and that is able to withstand the power output from the transmitter. The duplexer will continuously switch between the transmitter and receiver giving them alternate access to the antenna.

2.1.3 Receiver

The receiver works on the premise that the transmitted signal does not merge with the inbound signal that the antenna picks up. As mentioned in section 2.1.2, the duplexer solves this problem by isolating the path of the signal from the antenna to receiver, which allows for a clear signal to be received. The receiver then proceeds with amplifying this signal, and is designed in such a way that the amplification does not increase the noise figure more than necessary: low-noise amplifier.

2.1.4 Antenna

To allow the radar to transmit and receive signals it is theoretically required that two antennas are available, but since a duplexer is commonly used there is no need for two separate antennas and only one antenna is therefore needed. There are numerous different antennas used in radar, but it is only of interest to talk about phased array antennas since the units handled in the project are components of SAAB's phased array radar systems.

The structure and function of a phased array can be noted in figure 2.1, where the transceiver unit is connected to the phase shifters that phase shifts the signal for each antenna that they are individually connected to; this provides an ability to angle the signal along the antenna elements. However, exceeding $\pm 60°$ shifting reduces the quality of the transmitted and received signal substantially, and is generally the limitation of phase shifting in a phased array antenna. The frequency can also be changed to allow a directional change perpendicular to the phase shifting. (The information regarding Phased Array antennas were acquired during conversations with supervisor Lennart Berlin)

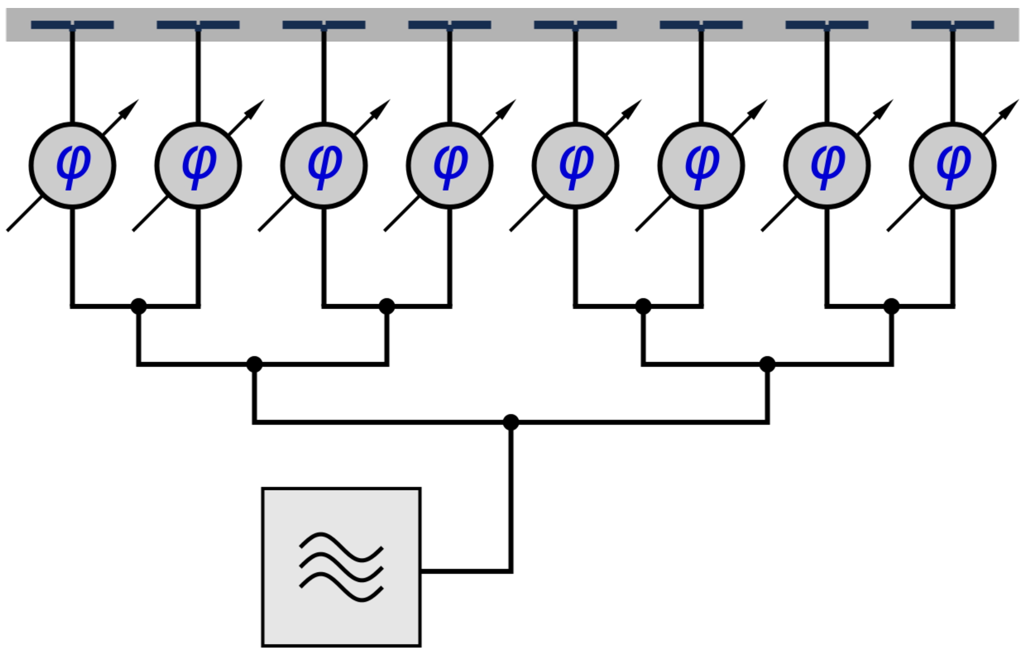


Figure 2.1: Simplified schematic of a phased array antenna. From the top: antenna elements, phase shifters, transceiver, [2], CC-BY

2.1.5 Indicator

A radar system would be rendered moot without the ability to handle received information. It is therefore imperative to read and convert the received information and display it for human use. To simplify the process: the signal is amplified and rendered on a screen for interpretation. However, not all radar applications require human interpretation to use the received information, such systems may include automatic traffic surveillance systems to monitor speeding.

2.2 Radio frequencies

As radar systems transmits its signals it is common for the radar's design to omit interference that might occur in the receiving part. It is to be noted that at the stage of production that the units are at, they are far from sufficiently shielded. This leads to problems when high power transmission tests occur in the proximity of other test stations in progress of testing the noise figure.

The problem is caused by Radio Frequency Interference (RFI) that occurs in electrical or electronic devices when energy is radiated or conducted. This is prominent when high-power radio frequencies are transmitted, such as in Radar. If a high-power transmission test were to occur it will with great probability induce interference in nearby devices.

2.3 Distributed testing

Traditionally, automated testing is carried out on local test stations, which are often situated on a computer that run the tests and store results.

In a modern production environment multiple products are often produced simultaneously, this creates a need for test stations to be able to do different kinds of tests on the different products that are manufactured. To solve this issue a distributed test system is a good idea, and by setting up a network environment where all test stations fetch the test scenario needed for the current product from a "test scenario server" this is accomplished. In figure 2.2 a sample environment is shown where three test stations, each compatible with all the different test objects, fetch the test scenario needed from the "test scenario server" depending on which kind of test object is connected. After the tests are done the test results are saved to the "Test result server".

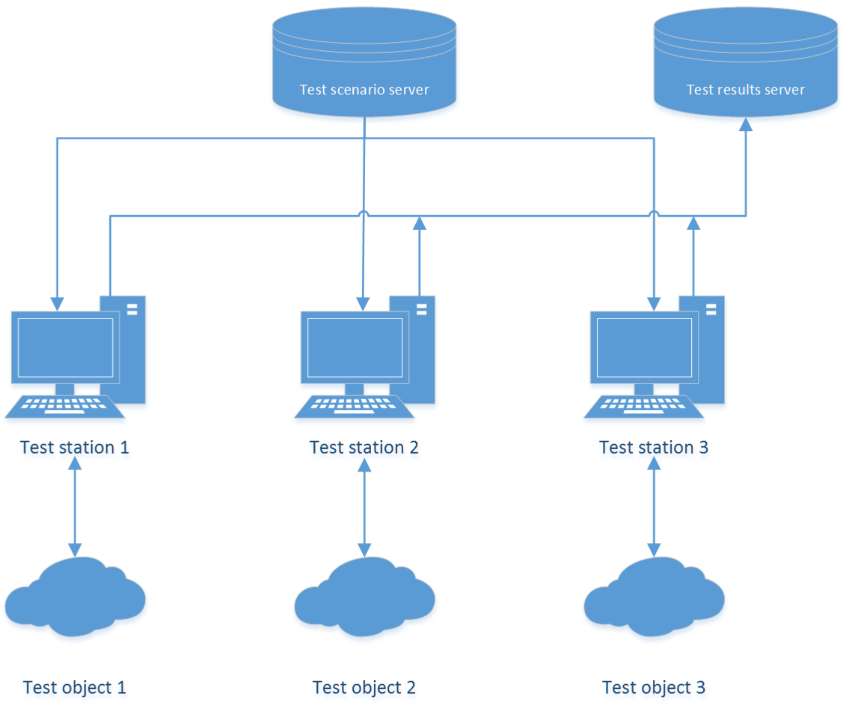


Figure 2.2: A sample overview of a distributed testing environment

Another benefit of distributed testing is that if changes are done to a test scenario not all test stations needs to be manually updated, as soon as the new scenario is available at the server all stations will automatically use it for all future testing.

2.4 National Instruments LabVIEW

LabVIEW(Laboratory Virtual Instrument Engineering Workbench) is developed by National Instruments(NI) in Austin, Texas. NI released their first LabVIEW version in 1986.

LabVIEW is a software development application for creating virtual instruments (VI) in a graphical interface based on two parts both shown in figure 2.3: the front panel designer, where the graphical user interface is situated; and the block-diagram designer, where the source code is designed and resides. These virtual instruments can be designed to communicate with local instruments connected directly to the computer running LabVIEW, or over the network with network connected instruments as well as databases to store measurement results and instrument settings. [3]

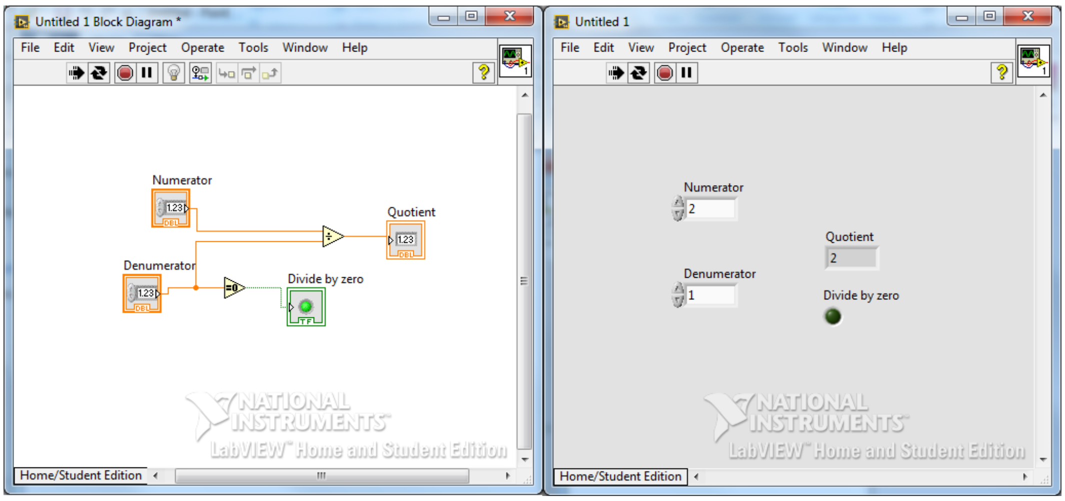


Figure 2.3: Left: The source code for a simple division calculator. **Right:** Front panel showing the calculator.

The front panel is shown for the user during execution of a LabVIEW program, and therefore hosts all controllers and indicators the developers want the user to see and interact with.

The source code or block diagram is built up of graphical blocks containing all common functions found in most programming languages. In addition to the common functions there are ready built libraries that can, for example, communicate with databases and other network resources.

2.5 National Instruments TestStand

TestStand, by National Instruments, is a software to develop and execute sequences for automated testing. TestStand cannot complete any test itself, but by running programs previously built in LabVIEW or any other applicable programing language such as C or C++ it can communicate with the Device Under Test(DUT).

Results from tests performed by TestStand can either be displayed directly in TestStand or published as reports in a variety of formats and directly to a number of database formats.

2.5.1 Process models

The process models can be described as a framework for the complete test sequence, without the process model the developer would have to set up the test environment manually for each project. With the process model in place, TestStand takes care of tasks like identifying the DUT and generating reports. [4]

There are three different process models in TestStand: Sequential Model, Parallel model and Batch model. [5]

Sequential Model

The Sequential model is the most basic version of the process models in TestStand; it allows for tests to be performed consecutively - one device at a time.

Parallel Model

In the parallel model multiple devices can be tested at the same time - and on the same test station. This is useful if there are multiple resources connected to the same test station that are needed during the test procedure. There is no need to start and stop all tests at the same time since there are control steps available in TestStand to make sure only one DUT is connected to each resource at any given time.

Batch Model

The batch model can be seen as a combination of the two methods mentioned above. A batch of DUTs can be connected to the test bench at the same time and can therefore be carried out on all the DUT's simultaneously, or sequential if they need a resource that can be used only once at the time. [6]

3 Method

This chapter describes in short the initial plan for the project,how the plan was followed, and how the project as a whole will be presented.

3.1 Components

Since this project completely aims to find a software solution to the problem no other hardware then the PC workstations handed out by SAAB has been used. Since license costs for LabVIEW and TestStand are very high, Chalmers University of Technology's license has been used for this project after approval from National Instruments.

3.2 Planning and specification of requirements

The project started off by collecting the required resources and making a rough plan for the upcoming work, this first project plan is presented as a Gantt chart in appendix A. Interviews were made with key personnel for the project, and a specification of requirements was made. These specifications are presented in chapter 4: Process.

Initially, the plan was to investigate how a change in testing order would harm the test results and statistics; the circumstances for each individual test might change the perimeters for following tests e.g. heating of components may differ depending on previous performed tests. But due to limited time and heavy workload that the colleagues at SAAB were subjected to, this part of the project had to be removed but might make a reappearance in future work.

After an initial learning period were LabVIEW and TestStand's 'Getting started'-manual [7] were rigorously read and followed, the work with finding potential solutions to the described problem was started.

3.3 Solutions and tests

The solutions presented in this project has been developed through an agile process where small changes has been tested and evaluated continuously during the project.

3.4 Presentation and report

The finalised project is to be presented in this report, as a business case for the staff at SAAB EDS, and in an oral presentation at Chalmers.

4
Process

The process chapter will explain how the work have developed throughout the project, which problems that have been encountered and the solutions to them.

4.1 Specification of requirements

The project started out with an extensive accumulation of information and specification. To carry out the project certain information was required in regard to which specifications the result needed to fulfil and how these requirements would be meet.

A couple of interviews were conducted with a number of people at SAAB's facilities in Kallebäck: the ones responsible for the development of automated test executives, and the personnel involved in the production and daily use of the test stations. This gave us a solid ground to build the project on.

Today SAAB conducts a series of test on circuit boards in an early stage of the production. Each of these tests are compiled of multiple short tests, testing different functions on the circuit boards, that all takes up to one hour to complete. Each small test can be divided into three different categories:

- **Receive** - Tests the receiver on the circuit board, this test is insensitive and can be ran at all times
- **Transmit** - Tests the transmitter on the board, this test will disturb the noise figure test if run simultaneously
- **Noise figure** - Tests the noise figure of the receiver, this test will be disturbed by the transmission test if run simultaneously

In every solution presented in this project all programming has been based on these three aforementioned categories, every program has been built so that it runs three different sections of code, and the goal has always been to minimise the risk of running interfering code blocks simultaneously.

4.1.1 Program description

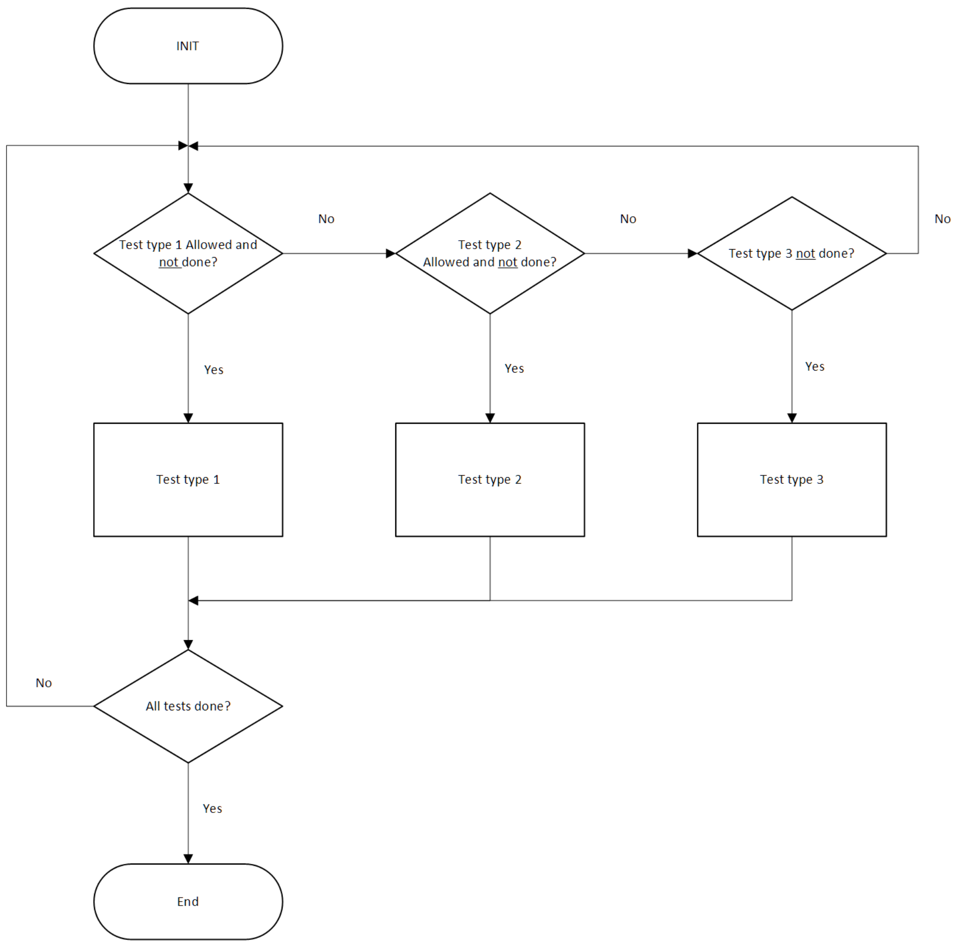


Figure 4.1: Overview of the intended program flow

In figure 4.1 a basic block-diagram of the intended test sequence flow is shown. The principle is that if possible, the tests will be conducted in the same order as today, but if another test station is busy doing a type of test that might get disturbed or disturb the system it will automatically select another test to run while waiting for the possibility to perform a disturbant tests.

The final program has to be started from the local test station, so even if the solution requires a server to control the run state of the test sequence, it has to be started by an operator at the station that is going to perform the test. This is so since not all tests will take the same amount of time, and because the DUTs are manually connected to the testing equipment and registered by scanning a barcode on the physical DUT.

4.2 Software

Due to the projects nature – software oriented rather than hardware oriented, a significant amount of time was spent accumulating theoretical knowledge and proficiency in the new software TestStand. This knowledge could be acquired with the help of SAAB employees and through NI's own documentation. The documentation and knowledge acquired from SAAB was necessary due to the company's own specialised software that is built on NI's TestStand application, but also because of SAAB's programming standards that its employees abide.

There was an initial intention to learn both LabVIEW and TestStand. However, as was quickly realised, LabVIEW could be dismissed at an early stage due to the versatility of TestStand. This limited the amount of documentation that had to be processed, and the (laborious) learning process of another programming language could be disregarded. At this part of the project all of the time was spent planning, on-the-go, as the agglomerated documents were thoroughly read.

The learning process of TestStand was by and large simplified with access to NI's extensive introduction manual 'Getting Started', but as with all manuals it lacked the necessary information for how one would be able to implement certain features, something other documentation could not provide either; herein, lies one of the challenges of this work.

4.3 Synchronisation steps

To provide a solid solution in a software such as TestStand, it is more often than not beneficial to use a readily available function that performs said task without errors. One such function, which stood out from the rest, was the Auto-Schedule function.

4.3.1 Auto-schedule - Overview

To perform multiple tests at the same time, a parallel testing technique is required. TestStand offers a couple of different choices and the one that seems to fit this application the best was 'Auto-Schedule'.

With limited testing equipment, one traditionally have to perform tests sequentially on one DUT at a time. If there is equipment available that enables your measurement instruments to be connected to multiple DUT's simultaneously via a switch, the tests can instead be carried out in parallel. This means that as soon as a DUT is done with the needed instrument the next DUT can use it; hence, the complete batch will be finished faster.

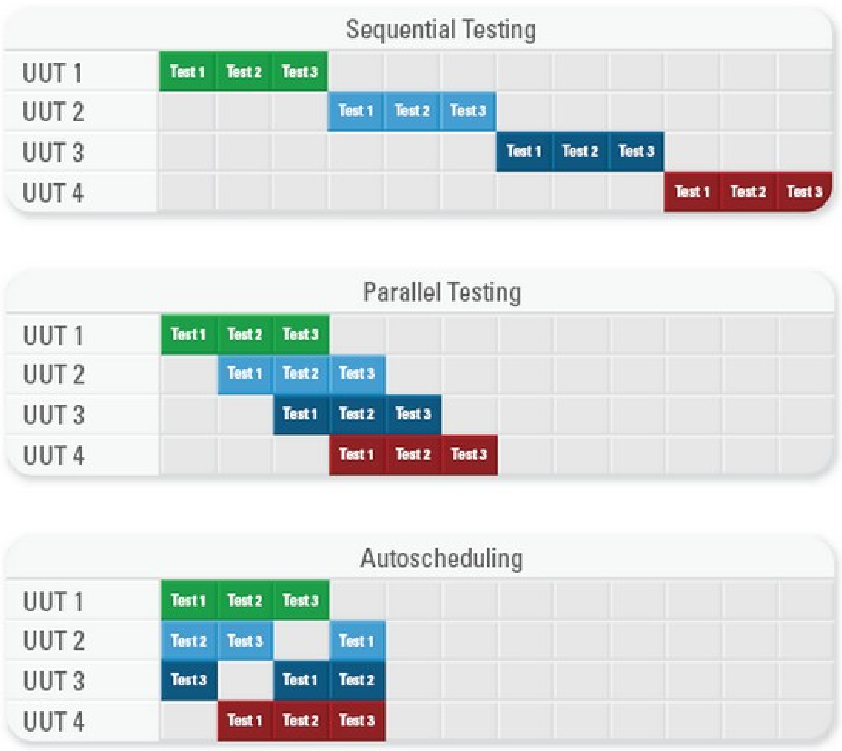


Figure 4.2: An overview of the different testing methods described in section 4.3.1 (Here UUT equals DUT), image by National Instruments [8]

In TestStand's Auto-Schedule function the tests are no longer required to be completed in a predetermined order, instead the computer automatically assigns a DUT to a free instrument. The difference between the described testing methods is visualised in figure 4.2

This leads to the realisation that tests are not guaranteed to be performed in the same order every time they are done. If, for example a DUT is heated up from a certain test, it might affect the following test results and make them deviate from each other.

4.3.2 Auto-schedule - This project

After finding out about the Auto-schedule function, the work started to form mainly from the acquired documentation distributed with the software, but also through the support forums at National Instruments' web page. With working example code, made for a standard test station with all DUT's connected to the same test bench, the work turned towards achieving a stable local network communication.

At first, a program intended to work in the same manner as National Instruments' example code as described in their manuals was created, but this time in a local network environment. This ended up being unsuccessful as nothing seemed to work the way it should. The problem was in the limiting constraints that the function had been made with. An e-mail contact within National Instruments' technical support was therefore contacted when it became necessary, it did actually turn out to be impossible to get this particular solution to work according to National Instruments; auto-scheduling was therefore rejected.

4.4 Master-Slave

To minimise the amount of variables sent back and forth between different test stations it was necessary to introduce an intermediary - in this case a Master computer. The Master, whose sole purpose is to issue commands, would act as a hub for which the Slaves (test stations) could communicate through. Without a Master there would be a need to send an excessive amount of information, something that would tax the overall system and clutter it if too many test stations were to be in use.

Without access to the test stations themselves, nor the actual software run on them (Amelia), there was a need to simulate the solution. As such, the TestStand program utilised the sequence capability within TestStand. Figure 4.1 in chapter 4.1.1 shows the basic structure of the program, and was realised almost exclusively through TestStand.

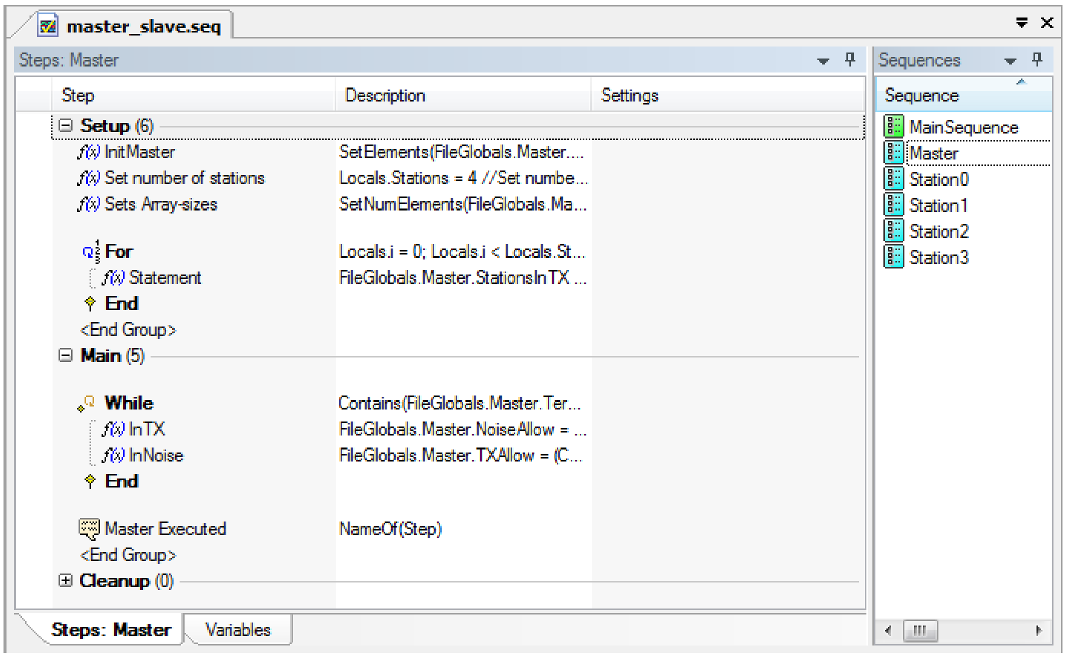
The program consists of several sequences: a 'Main Sequence', a 'Master Sequence', and an indefinite number of 'Station Sequences', and is designed in such a way that a single computer will run it. As such, it simulates not only a Master computer but also several Slaves concurrently.

The 'Main Sequence' was designed in such a way that each new sequence was called and executed in a new thread on the processor, so that the program could simulate several Slaves concurrently; the program would continuously execute the 'Master Sequence' throughout the lifetime of the 'Main Sequence'. With the 'Sequence Call' Step and the 'Wait' Step it was possible to actualise this, while also allowing each Slave to start with an interval in-between them to roughly simulate how the real tests would start.

The 'Master Sequence' was designed for the sole purpose of allowing either the Noise tests or the TX tests to run on the Test Stations. Following is the basic principle behind the 'Master Sequence' which toggles the permission flag of the Noise and TX tests.

```
while (SlavesRunning) {
    NoiseAllow = StationsInTX ? False : True;
    TXAllow = StationsInNoise ? False : True;
}
```

This will only work by specifically designing the 'Station Sequence' to toggle the 'StationsInTX' and 'StationsInNoise' variables when necessary. In figure 4.3 both the general sequence structure and 'Master Sequence' of the program is shown



Step	Description	Settings
Setup (6)		
InitMaster	SetElements(FileGlobals.Master....	
Set number of stations	Locals.Stations = 4 //Set numbe...	
Sets Array-sizes	SetNumElements(FileGlobals.Ma...	
For	Locals.i = 0; Locals.i < Locals.St...	
Statement	FileGlobals.Master.StationsInTX ...	
End		
<End Group>		
Main (5)		
While	Contains(FileGlobals.Master.Ter...	
InTX	FileGlobals.Master.NoiseAllow = ...	
InNoise	FileGlobals.Master.TXAllow = (C...	
End		
Master Executed	NameOf(Step)	
<End Group>		
Cleanup (0)		

Sequence
MainSequence
Master
Station0
Station1
Station2
Station3

Figure 4.3: The left panel shows the 'Master Sequence' code, and the right panel shows the available sequences of the program

The 'Station Sequences' work in conjunction with the 'Master Sequence' to simulate the testing environment. It works by using the flags that the 'Master Sequence' continuously toggles, and provides help in return by toggling the 'StationsInTX' and 'StationsInNoise' variables when entering a test. To simulate each test, a VI which just put the computer on hold for a predetermined time was used.

Something that became apparent from observations of the real test environment was that the length of each type of test varied, so the duration of each type of test was set accordingly.

The structure of the ’Station Sequence’ is far more complex than the ’Master Sequence’, much due to the fact that it is designed to run in a specific way but also because it has to accommodate the need to quickly and reliably add more ’Station Sequences’.

4.5 Master-Slave with remote execution

Based on the previous Master-Slave solution, and in collaboration with the National Instruments support staff the idea to solve the communication issues with remote execution of TestStand sequence files came up.

In TestStand, one can at any time call a different sequence that is run either in the same thread, in a new thread or on a remote computer. To run a sequence on a remote computer, both computers have to be in the same local network and configured to allow remote execution of TestStand sequences.

One method to control which type of test is allowed to run at a specific moment would be to let a Master computer remotely control the test sequences and call the allowed tests on the test stations when allowed to do so. However, this contradicts the statement of section 4.1.1 that testing of a DUT shall always be initiated on the local test station.

On the master computer a sequence file aimed to keep track of what type of tests currently being executed need to be executed continuously. This sequence will collect incoming signals from the station sequences and set clearance signals that are passed back to determine if a certain test are allowed to be executed as described in section 4.4.

Also a station sequence file is stored on the master computer. This sequence starts when a test station calls it after gathering essential information about the DUT, such as serial number and type, graphicaly explained in figure 4.4. The station specific sequence might contain different sets of rules for the test execution based on the type of DUT and calls a basic sequence on the test stations only containing the actual instrument communication programs.

To communicate between the different sequences and computers in the network a TestStand, built-in step called queue were used. The station sequences put its current state in a queue and the master sequence dequeues it and calculate the current state flags for the other stations to handle.

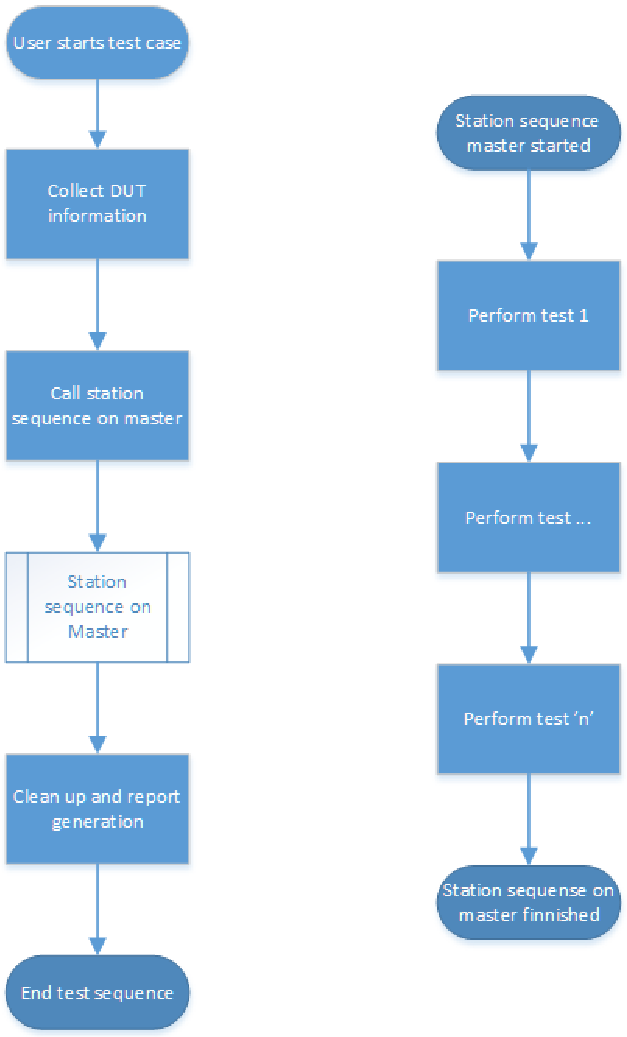

User starts test case
Collect DUT information
Call station sequence on master
Station sequence on Master
Clean up and report generation
End test sequence
Station sequence master started
Perform test 1
Perform test ...
Perform test 'n'
Station sequense on master finnished

Figure 4.4: Flowchart visualising the remote execution sequences described in section 4.5

4.6 Prioritisation

When many (6 or more) test stations are running at the same time a new problem arise: wasting time. The tests begin to wait for one another so that they can execute, and the more tests that are waiting to start, the more time accumulates. In an attempt to reduce the amount of time spent waiting an extension to the Master-Slave solution was developed.

The underlying functionality of this extension was to make a predetermined schedule for each test station based on the previously started test; once made, it would stay the same for the remainder of its execution. This makes the number of variables sent between the Master and Slave computers abate or disappear completely.

To realise this, a sequence to compare and fill an array was written. It worked on the premise that Noise and TX tests had a higher priority level than the RX test. Below follows a visualisation.

Noise	1
TX	1
RX	0

As can be seen, the noise and TX test has a priority level of 1 and are equal, while the RX test has a priority level of 0 and is below both noise and TX in importance.

When the sequence compares the previously started test station's priority array, it takes into consideration where in the array that the test station currently is. Then it begins to compare the value of the index that is pointed at by the program along with the index right after the pointed index.

0	TX
1	Noise
2	RX
3	TX

If the previously started test station's priority array is pointing at index 0 then it will look at both index 0 and 1, compare them to each other, and insert a value into the newly started test station's priority array according to the presented information below. Then it will look at index 1 and 2, 2 and 3, and so on.

Noise	+	RX	->	Noise
TX	+	RX	->	TX
Noise	+	TX	->	RX

Doing this for three pairs of values in consecutive order will provide a predetermined priority array for the new test station.

5
Results

The purpose of this project was to find a way for test sequences in TestStand to autonomously communicate, and in that way control test station so that if there are multiple stations working at the same time they do not disturb each other.

5.1 Auto-schedule

As mentioned in section 4.3.2 the auto-schedule function is intended to be used on a single test station when connecting multiple DUT's to it. In our scenario with multiple test stations, but only one DUT at each the limitation of the standard step type in TestStand showed to be too narrow to fit our intended solution.

5.2 Master-Slave

The first examined Master-Slave solution works when run on a single machine, but since this project's goal is to run the slave sequences on different machines connected via local network, TestStand lacks the built-in function to share data in-between the different stations.

One possible solution to this would be to communicate over a database. To accomplish this one would have to design a LabVIEW VI to read and write data to the said database.

With a third unit in the chain of communication, valuable time would be lost and the stations will have to wait for the communication to be completed. This might lead to one station starting one kind of test, and before the master have the information about it another station starts a different kind of test, inevitably leading to disturbance in the production room.

5.3 Remote execution

As a follow up to the Master-Slave solution, which lacks the ability to communicate between threads, and after some input from the National Instruments support crew, the work on a solution based on TestStand's ability to remotely execute test sequences began.

With the remote execution functions in place and a communication queue built up the thread-to-thread communication works and the time spent for communication seems negligible.

This solution might require more time per test station than it currently do in the test facility at SAAB. However, the Master-Slave with remote execution solution reduces time spent overall, and especially so when more and more test stations are introduced to the testing environment.

Tests were run in three different configurations: two, four, and eight test stations. For simulation purposes the test time of each test station was set to 6 minutes compared to the real test stations where the tests normally takes 60 minutes. Furthermore, the two-station test was done twice since it seemed like the time of starting a test station might affect the total runtime. As seen in appendix B for tests carried out in a two-station environment the time of start up of the second station significantly affects the total time spent for both stations. This is since the runtime of different sub tests might differ in time and therefore hold the other station waiting for a long time.

With the four-station configuration an increase in waiting time can be seen, but it is to be noted that this configuration is also affected (just as the two-station one) by the starting time of each test station. However, the increase in waiting time is almost insignificant in this case.

It is not until an eight-station configuration is established that a noticeable increase in waiting time is affecting the system. One of the tests wait for a total of 44 seconds during its execution time, which amounts for a 12.2 % increase in execution time.

The increased waiting time when more test stations run concurrently was the reason as to why a prioritisation solution was pursued.

5.4 Prioritising

The task to prioritise and optimise the Master-Slave solution was more complex than first imagined. It isn't impossible, but to achieve a prioritisation that is quicker than the original Master-Slave you will end up with a messy, obtuse, and quite advanced solution. This is discouraged by the fact that distributing such a test network and building upon it at other locations would require more effort than something simpler and therefore costs more money. It is also more likely that it will be unstable.

The solution presented in section 4.6 is a simple and incomplete attempt at prioritising the Master-Slave solution. Alas, it actually increases the execution time at this stage. Changing the predetermined list into an ever-changing priority list would allow for better execution times, but at the cost of having to send more information to and from the Master computer.

Another way to work with the prioritisation of these tests is to somehow track the time of each test in progress, compare the tests being executed, and from that start the type of test that provides the fastest execution time for the whole system. An attempt at such a solution was made, but it was something that never came to be.

5.5 Conclusion

The results of this project made it apparent that an automatic control of which tests are allowed to run might improve the testing environment in the early state of the unit's production. This is not necessarily true for every single test case, but over a longer period of time it would decrease the manual labour required and increase the amount of units successfully tested.

In the case that only a few test stations are attached to the test environment a noticeable improvement due to the nature of the Master-Slave solution in comparison to the current installation can be seen. However, the solution is not optimal at a bigger number of test stations.

A priority-based solution that improves the execution time of the Master-Slave solution would be the next step forward, but it is something that has not been presented in the report - not for lack of trying.

5.6 Environmental and ethical aspects

Since the project aims to improve an application already in use no further environmental studies have been carried out, one clear aspect is that the work environment for those involved in the production might be improved since the manual handling of DUTs are decreased.

Radar systems might be used both in military as well as civil applications. SAAB has signed the United Nations Global Compact that express that the company should work for human rights and preservation of the environment as well as working against corruption. [9]

Bibliography

[1] M. I. Skolnik and A. (e-book collection), *Radar handbook*, English. New York: McGraw-Hill, 2008, ISBN: 9780071485470; 0071485473; 9780071621137; 007162113X.

[2] wdwd & based on a PNG drawing from de:Benutzer:Averse, *Phased array parallelfeed*. [Online]. Available: `http://commons.wikimedia.org/wiki/File\%3APhased_Array_parallelfeed.svg` (visited on 2015-05-28).

[3] National Instruments. (2015). Labview, [Online]. Available: `http://www.ni.com/labview` (visited on 2015-05-25).

[4] National Instruments. (2014). Best practices for ni teststand process model development and customization, [Online]. Available: `http://www.ni.com/white-paper/7958/en/` (visited on 2015-05-25).

[5] National Instruments. (2013). About the teststand process models, [Online]. Available: `http://www.ni.com/white-paper/2694/en/` (visited on 2015-05-25).

[6] National Instruments, *Teststand process models*, 370360A-01, 2001-03. [Online]. Available: `http://www.ni.com/pdf/manuals/370360a.pdf` (visited on 2015-05-19).

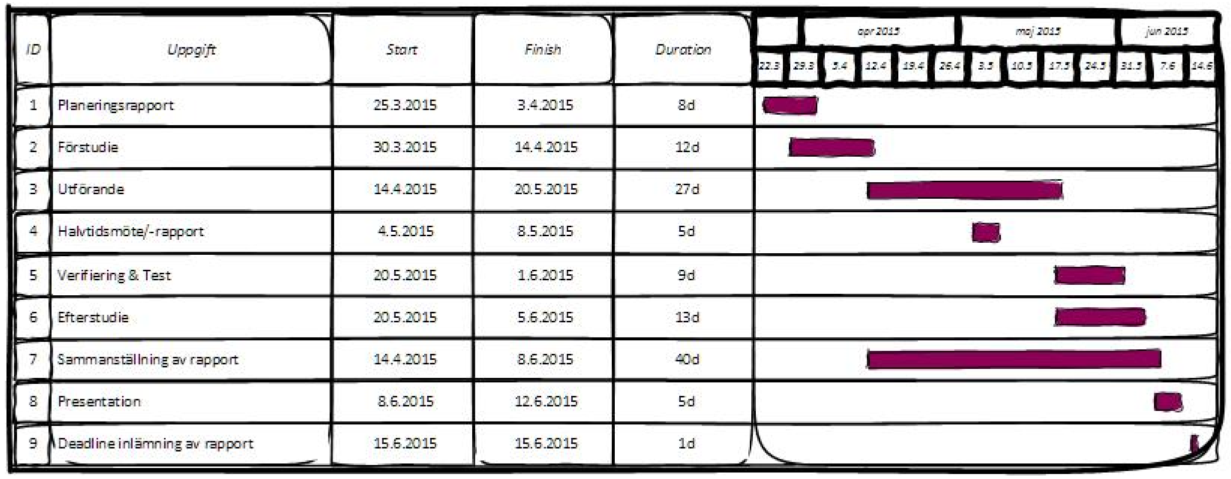
[7] National Instruments. (). Getting started with teststand, [Online]. Available: `http://www.ni.com/pdf/manuals/373436f.pdf` (visited on 2015-04-03).

[8] National Instruments, *Testing techniques in teststand*. [Online]. Available: `http://www.ni.com/teststand/optimize/` (visited on 2015-06-03).

[9] Håkan Buskhe. (). Code of conduct, [Online]. Available: `http://saabgroup.com/responsibility/responsible-business/code-of-conduct/` (visited on 2015-06-09).

A

Gantt chart



ID	Uppgift	Start	Finish	Duration
1	Planeringsrapport	25.3.2015	3.4.2015	8d
2	Förstudie	30.3.2015	14.4.2015	12d
3	Utförande	14.4.2015	20.5.2015	27d
4	Halvtidsmöte/-rapport	4.5.2015	8.5.2015	5d
5	Verifiering & Test	20.5.2015	1.6.2015	9d
6	Efterstudie	20.5.2015	5.6.2015	13d
7	Sammanställning av rapport	14.4.2015	8.6.2015	40d
8	Presentation	8.6.2015	12.6.2015	5d
9	Deadline inlämning av rapport	15.6.2015	15.6.2015	1d

B

Run times

B.1 Two stations

First run

Station	Start time	End time	Time spent
1	15:42:19	15:48:22	00:06:03
2	15:42:50	15:49:35	00:06:45
		Average time:	00:06:24
		Standard deviation:	00:00:21

Second run

Station	Start time	End time	Time spent
1	15:51:05	15:57:07	00:06:02
2	15:52:15	15:58:17	00:06:02
		Average time:	00:06:02
		Standard deviation:	00:00:00

B.2 Four stations

Station	Start time	End time	Time spent
1	15:27:03	15:33:11	00:06:08
2	15:25:01	15:31:11	00:06:10
3	15:25:34	15:31:37	00:06:03
4	15:25:48	15:31:50	00:06:02
		Average time:	00:06:06
		Standard deviation:	00:00:03

B.3 Eight stations

Station	Start time	End time	Time spent
1	15:10:57	15:17:08	00:06:11
2	15:10:46	15:17:01	00:06:15
3	15:10:33	15:17:01	00:06:28
4	15:14:01	15:20:10	00:06:09
5	15:10:58	15:17:24	00:06:26
6	15:11:21	15:17:29	00:06:08
7	15:11:42	15:17:49	00:06:07
8	15:12:08	15:18:52	00:06:44
		Average time:	00:06:18
		Standard deviation:	00:00:12