TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

VILLE SIPINEN

A GENERIC TEST PLATFORM FOR AUTOMATED CAMERA MODULE VALIDATION

Master of Science Thesis

Examiners: Professor Matti Vilkko and Assistant Professor David Hästbacka
Examiners and subject approved on 31st January 2018

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Automation Technology
SIPINEN, VILLE: A Generic Test Platform for Automated Camera Module Validation
Master of Science Thesis, 53 pages
January 2018
Major: Automation Software Engineering
Examiners: Professor Matti Vilkko and Assistant Professor David Hästbacka
Keywords: Automation, testing, validation, camera module, software

Nowadays every smart phone has at least one camera, sometimes even three or more. Even most of the very basic phones has a camera inbuild. This huge demand on camera module business has brought new challenges also to development since the number of versions has riced together with tighter schedules.

Camera module validation is a big part of the development phase as every development version needs to be validated carefully. When working with camera module suppliers, every round of validation contains a set of different tests including but not limiting to image quality, hardware and functional, electromagnetic compatibility and reliability tests. So far, the tests have been mostly developer driven and at most semi-manual including image capturing manually and reporting results manually. This kind of things consumes a lot of time and money as the engineer is tied to do simple manual tasks when one could be doing more valuable work. Not to mention the losses for the company if the product doesn't reach the market in time because of the schedule wasn't reached. Test automation brings heavily needed time savings but there hasn't been good enough software that is capable to automate tests in various different categories mentioned above. The goal with the test automation is to also limit human made errors to minimum and to generate comparable results so that both the supplier and the customer can do the same tests and get the same results. This is important to be able to rely on suppliers' test results.

This thesis introduces a way to create automated testing system on top of the existing test software. The idea is to add another layer of software on top of graphical user interface and introduce scripting framework with modules that enables easy development of new automated tests and generic reporting. General test cases are introduced in this thesis to get overall horizontal picture of the validation process.

The goal of the thesis was to deliver a very general platform to enable automated camera module validation in as many sectors as possible. The platform does give a lot of benefits in the validation and was taken successfully into use in various tests. The system is huge and there comes a lot of new feature requests all the time. There are also some ideas to further develop for example in report generation and script syntax checks.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Automaatiotekniikan koulutusohjelma
SIPINEN, VILLE: A Generic Test Platform for Automated Camera Module Validation
Diplomityö, 53 sivua
Tammikuu 2018
Pääaine: Automaation ohjelmistotekniikka
Tarkastajat: Professori Matti Vilkko ja Assistant Professor David Hästbacka
Avainsanat: Automaatio, testaus, validointi, kameramoduuli, ohjelmisto

Nykyisin jokaisessa älypuhelimessa on vähintään yksi kamera, joissakin jopa kolme tai enemmän. Jopa kaikista yksinkertaisimmissa peruspuhelimissa on kamera usein sisäänrakennettuna. Suuri kysyntä on tuonut uusia haasteita kameramoduulien kehitykseen, sillä tiukempien aikataulujen lisäksi myös erilaisten versioiden määrä on kasvanut.

Kameramoduulien kehitysvaiheessa validointi on suuressa roolissa. Työskennellessä kameramoduulitoimittajien kanssa, jokainen validointikierros sisältää ainakin kuvanlaadun, laitteiston toiminnallisuuden, sähkömagneettisen yhteensopivuuden sekä luotettavuuden testauksen. Tähän asti testit ovat olleet pääasiassa kehittäjien tekemiä, usein korkeintaan puoliautomatisoituja, pitäen sisällään käsin tehdyn kuvankaappauksen sekä raportoinnin. Tällaiset käsin tehtävät työt vievät valtavasti aikaa sekä rahaa, kun kehittäjät ovat sidottuna yksinkertaisiin tehtäviin eivätkä ratkomassa isompia ongelmia. Puhumattakaan yritykselle tulevista tappioista, jos tuote ei ennätä markkinoille ajallaan esimerkiksi vaillinaisen testauksen vuoksi. Testiautomaatio tuo paljon kaivattua ajansäästöä, sillä aiemmin ei ole ollut saatavilla ohjelmistoa, mikä pystyisi automatisoimaan näin laajan kirjon eri tyyppisiä testejä. Testiautomaation tavoitteena on lisäksi vähentää käyttäjän tekemiä virheitä, sekä tuottaa vertailtavissa olevia tuloksia. Tällöin sekä toimittajat että tilaaja voivat tehdä samat testit omissa tiloissaan ja saada vertailukelpoiset tulokset. Tämä on ensiarvoisen tärkeää, jotta tilaaja pystyy luottamaan toimittajilta saatuihin testituloksiin.

Tämä diplomityö esittelee yhden ratkaisun testiautomaation kehittämiselle olemassa olevan testijärjestelmän päälle. Ajatus on tuoda olemassa olevan graafisen käyttöliittymän päälle yksi ohjelmakerros lisää, jolla ohjataan erillisten komentosarjojen avulla koko ohjelmistoa. Lisätty modulaarinen kerros mahdollistaa uusien automaattisten testien helpon kehityksen ja yleiskäyttöisen raportoinnin. Diplomityössä esitellään yleiset testitapaukset, jotta saadaan horisontaalinen käsitys siitä, millainen koko validointiprosessi on.

Diplomityön tavoitteena oli kehittää testialusta kameramoduulien validointiin, joka olisi käytettävissä mahdollisimman monessa eri tyyppisessä testissä. Testiohjelmisto otettiin käyttöön osana validointiprosessia onnistuneesti useassa eri testitapauksessa. Koko testijärjestelmä on hyvin laaja ja vaatimuksia uusille ominaisuuksille tulee jatkuvasti.

Työn lopuksi käsitelläänkin hieman parannus- ja jatkokehitysehdotuksia liittyen muun muassa raporttien tuottamiseen ja komentosarjojen syntaksin tarkistukseen.

PREFACE

This Masters of Science thesis, A Generic Test Platform for Automated Camera Module Validation, has been a long journey when trying to find the time, energy and will to finalize in the end so big and important milestone. The journey has taken more than a few years while working hard at one of the world's leading technology companies, Nokia. It has been amazing time to learn so much from my colleagues and finally to get inspired to finalize this thesis. I would like to thank them all, it has been exciting time. Great thanks to my mentors Ossi Pirinen and Juha Alakarhu for encouraging support.

I would like to thank Professor Emeritus Seppo Kuikka, Professor Matti Vilkko and Assistant Professor David Hästbacka for examining the thesis and all the other help received. Moreover, I would like to thank my whole family for the endless support and believe of me. Special thanks to my beloved wife, Sanna, for the endless love and care.

Tampere, January 21th, 2018

Ville Sipinen
Kemiankatu 15 B 9
33720 Tampere
Finland
Email: ville.sipinen@outlook.com

TABLE OF CONTENTS

TERMS AND DEFINITIONS

API	Application programming interface
ADC	Analog to digital converter
CCD	Common Camera Driver
CCD	Charged Coupling Device
CCI	Camera Control Interface
CHWD	Common Hardware Driver
CMOS	Complementary Metal Oxide Semiconductor
CSI-2	Camera Serial Interface 2
DUT	Device under test
EMC	Electromagnetic compatibility
FPGA	Field-Programmable Gate Array
GUI	Graphical user interface
HW	Hardware
IE	Image Engineering
ILC	Inter-laboratory comparison
IQ	Image quality
MEMS	Microelectromechanical systems
MIP	Mobile Imaging Playground
MIPI	Mobile Industry Processor Interface
MTF	Modulation Transfer Function
OIS	Optical image stabilizer
R&D	Research and Development
RGB	Red, Green, Blue color model
SMIA	Standard Mobile Imaging Architecture
UI	User interface
VUP	Verification Upstream

1 INTRODUCTION

Since the beginning of 21st century when the first mobile phones with built-in digital camera saw the day light, the amount of camera phones has been growing rapidly. Nowadays every smartphone contains at least one digital camera. Quite often there are one camera module on both sides of the smartphone, front and read camera. In 2016 the total number of sold smartphones worldwide were 1 495 million units. Less than ten years ago, in 2007 the corresponding number was less than 10 % of it, 122 million units [1]. Numbers are displayed in Figure 1.1.

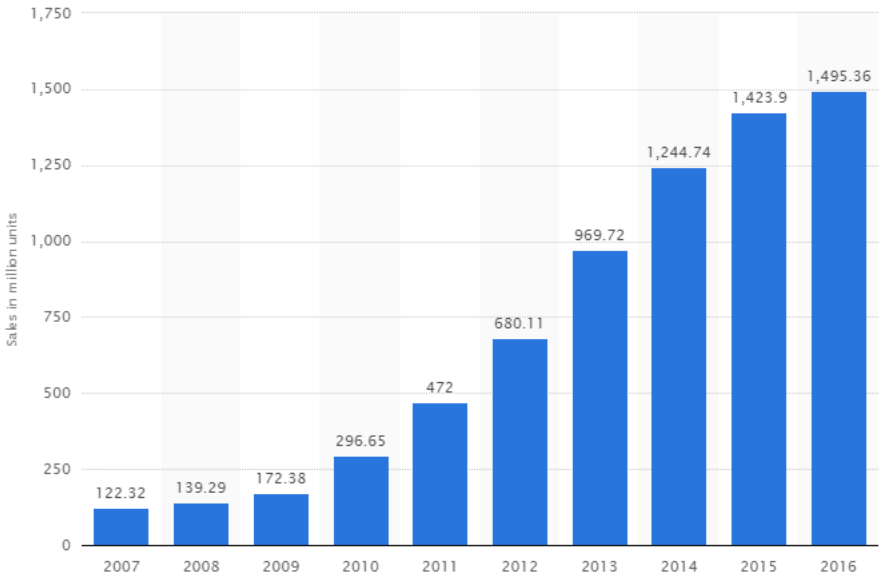

Figure 1.1. Number of smartphones sold to end users worldwide from 2007 to 2016 (in million units) [1].

Higher demand of the camera modules, both in units and versions, brings new challenges also to development phase, where all the time the schedules gets tighter and requirements tougher. Saving time in the development phase comes essential so latest at this phase the automation of repeatable tests gets higher priority. These tests are part of the camera validation process.

What types of camera module testing exists? Camera module testing can be divided roughly to three different categories: testing that happens during the research and development phase (R&D), testing that is done during the camera module manufacturing i.e. production testing, and camera module calibration in production. This thesis focuses on testing which is done during the R&D. The R&D testing can be also called as camera validation, since these are the tests that validates, for example, the camera module version sample's functionality. The camera validation consists of multiple subcategories but three

most relevant for this thesis are functional, image quality and electromagnetic compatibility testing. These three subcategories can benefit a lot from the test automation so others, like reliability, is left out of this thesis. An example of the automated validation is done for the functional testing.

Camera validation is needed to verify that the camera module meets the requirements. The validation can be done manually but it is a really time-consuming process. Automation not only speeds up the testing but also releases a man from doing all the manual work so one can concentrate on more productive work. The third benefit gained from the automated process is that the validation can be done to increased amount of camera modules as well as more often during the development. That way more data can be gathered which leads to better knowledge of the camera module's functionality and possible errors in design can be found from the very first samples to save major extra costs coming from the late changes. Repeatability is also one of the most important reasons to use automated tests to close one big uncertainty, human, off. The automated testing makes sure that human cannot by accident select for example wrong camera parameters but the test script handles that. One must keep in mind though that the operator of the automated tests must know what tests are executed and which instruments are used.

Camera module manufacturers and software engineers in smartphone business have been doing relevant development around test automation. Big volumes in camera module production has been pushing test method development forward [2], where the aim is to identify issues in manufacturing process that causes defects in camera modules and decreases yield. Complex software systems with fast development cycles are being tested with automated test methods [3]. In software testing the automation is taken so far that even the test case generation can be done automatically [4].

Target of this thesis is to design and implement a test platform to automate camera module validation and to show what kind of test cases can be tested with the system. The target is not to implement all the test cases but to give some examples and tools to allow users to easily build more automation for the future validation process. This is how the human workload can be decreased and at the same time the knowledge about the device under test (DUT) can be increased. Target is to find out, how one can be more efficient by letting automation to do most of the repeatable tasks and by reducing errors in measurements to minimum. Both benefits can increase company's profitability not so much by cutting down the R&D costs but being able to do more and better with the same investment.

The automated testing system is developed on top of already existing software called Mobile Imaging Playground, MIP. The software was made for algorithm and camera testing and it is developed internally at Nokia. What comes to camera testing, MIP didn't have any automation in place. That was the starting point when this thesis was started. Another option, LabVIEW is also presented shortly and evaluated if it would have been an alternative for MIP. MIP was chosen as a platform because it is already widely used internally and by camera module suppliers. Also, existence of the camera drivers among many other instrument drivers in the MIP had a significant impact to the decision.

This thesis will describe what is the camera module validation, how it is done and how it could be optimized. It will present and compare the old and new methods of the camera module validation by using time as the most valuable metric. Also, the faultlessness in test results is a major metric for the comparison. The thesis will first introduce MIP in chapter two. In chapter three, camera module components are explained, theory of the camera validation process from functional, image quality and electromagnetic compatibility point of view is examined, and test instruments as well as test automation software development are introduced. Chapter four will introduce the generic script framework and its implementation is walked through. This framework will work as a basis of the automation and in chapter five it is evaluated and compared to old methods. In this chapter will be seen how usable the framework is and how much it gives benefits in terms of time, workload and reproducibility. The final sixth chapter includes the conclusion and some thoughts about further development of the framework in the future.

2 MOBILE IMAGING PLAYGROUND

Mobile Imaging Playground, MIP, is Nokia's own, internally developed, camera and algorithm test software. First, in 2006 it was developed as a generic algorithm test platform and couple years later also camera team found it useful to develop camera test software on top of MIP's platform. As MIP is a plugin based software, meaning that different components can be developed independently of each other, testing can be split into small pieces. Small individual parts ensure easy and efficient testing of the component which makes the entire system more robust. This also helps to build different test systems using common existing plugins so one doesn't have to invent the wheel again and again. This chapter describes the basics of MIP covering its architecture, platform, basic components and camera interface.

2.1 Overview

MIP is a plugin based software developed by Nokia. It is used for testing different types of algorithms and nowadays also Nokia's camera testing is built on top of MIP. Nokia's Verification Upstream -team, later called as VUP, uses MIP not only in-house but it is also used by camera module suppliers and that way tests done in VUP can be aligned within Nokia and its suppliers.

As the software's name gives a hint, MIP's user interface is so called playground. In the playground, user can take plugins into use dynamically. User takes plugins in use by drag and dropping those from a plugin-list to the playground. Plugins on the playground can be connected to each other to deliver information from one to another. For example, one plugin can handle camera configuration and capture an image, then next plugin converts the image to another format and third plugin can show the image to the user. That way with three plugins a user can control camera, take image and see the result.

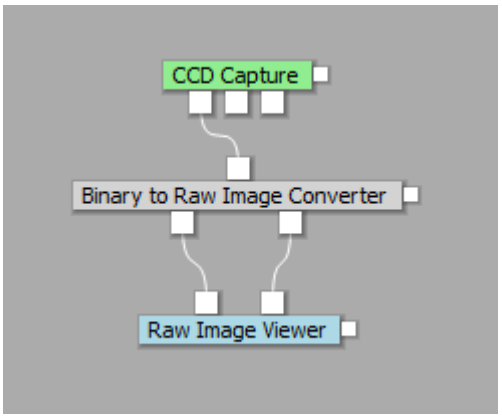

Figure 2.1. A basic playground to capture and view an image.

The *CCD Capture* plugin has three outputs, the first one outputs the binary raw data captured from a camera and the two others are for additional metadata. In this case only the image is wanted so two other outputs can be ignored. The first output is connected to *Binary to Raw Image Converter* plugin's only input. This means that whenever the data is received, the plugin is executed right away. *Binary to Raw Image Converter* converts the binary data to more generic raw data and extracts embedded metadata. The embedded metadata is defined by Standard Mobile Imaging Architecture (SMIA). The plugin has two outputs and the first one is for the raw image data and the second one is for the embedded metadata. In this case both outputs are used and fed to a *Raw Image Viewer* plugin's inputs. Similarly, as previous plugin's outputs, the *Raw Image Viewer*'s inputs takes identical type of data in; raw image data and embedded metadata. The metadata is needed to be able to show the image in correct format, e.g. the resolution and color channel order. When the plugin chain gets executed, user can view the captured image from the *Raw Image Viewer* plugin's user interface.

The idea is to make the plugins as generic as possible, so one can use the same plugin in multiple scenarios. For example, it is not needed to capture an image with CCD Capture -plugin to be able to view it in Raw Image Viewer -plugin. One can change the CCD Capture -plugin to another one which for example loads previously saved image from a hard drive and outputs that one for the viewer.

2.2 Architecture

MIP is a platform that enables easy plugin development and that way it is easy and fast to add new features to MIP software. All the plugins can be compiled separately without the platform and are loaded dynamically when dragged to the playground, so number of the plugins in MIP doesn't affect to MIP's efficiency. This also enables a possibility to update individual plugins without need of full program update.

2.2.1 Common Camera Driver

One of the biggest plugin families is a project called *Common Camera Driver* (CCD) which includes all the plugins that is used to control camera modules. It is built up to simulate a real product i.e. smart phone, so camera module can be tested against same interfaces as in the final product in MIP in early phase of the module development. When camera module is used on its own, it is placed on a capture board and through that the camera is controlled via PC. The CCD supports two different types of capture boards. CCD abstracts capture boards with *Common Hardware Driver* (CHWD) layer and furthermore the whole camera API is abstracted to *CCD Client* -interface. Other lower level interface is *Camera Driver*, which is used to control the camera sensor and actuators like auto focus or optical image stabilization. In Figure 2.2 all the parts of the CCD can be seen.

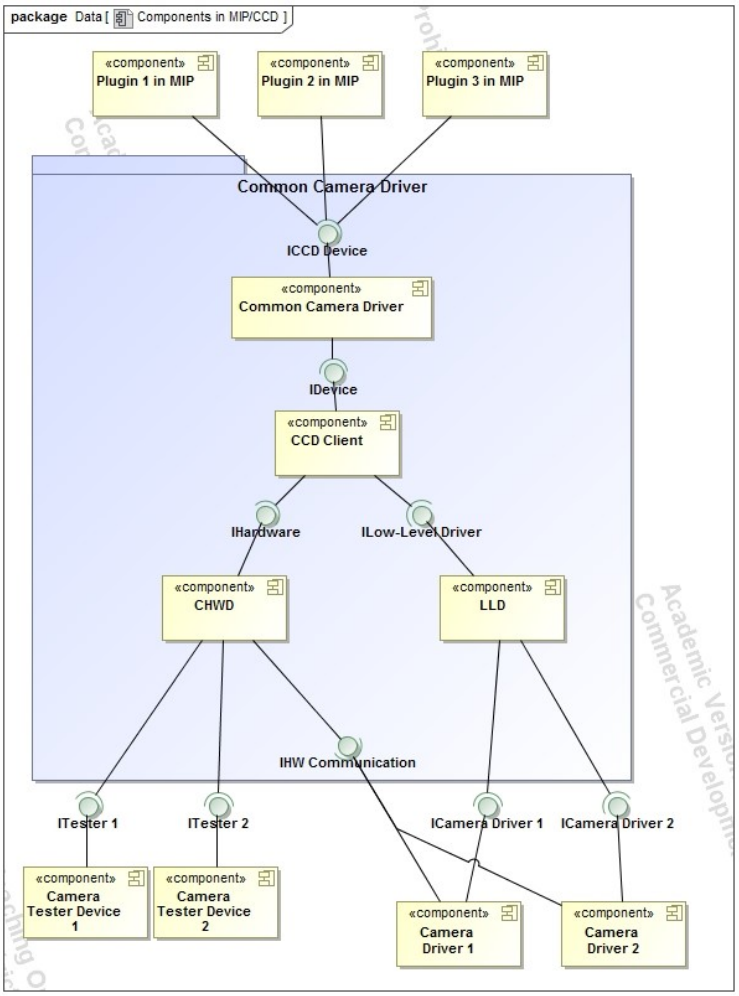

Figure 2.2. Components in Common Camera Driver.

Figure 2.2 shows on the top of the figure the three example plugins that uses the CCD Device interface. CCD Device provides a single interface that all components must use when controlling any camera module. The CCD uses Singleton software design pattern as it restricts all the plugins to use the same instance of it [5]. There can be multiple instances of CCD Devices at the same time but all of them uses the same static instance of CCD Client. Number of CCD Devices is the number of attached camera modules. Each of CCD Devices has device id which is also set to one camera module and camera tester device. CCD Client uses CHWD interface to control camera testers. Through this, tasks like tester initialization, frame capturing via Camera Serial Interface (CSI-2) protocol and current consumption measurement are done. CCD Client uses Low-Level Driver (LLD) to communicate with camera modules. Communication is done via I2C-protocol. LLD wraps all camera drivers under so again the device id tells which camera module is now under command. Camera testers provides an interface for the I2C-communication, so camera drivers use the Hardware Communication interface to write and read registers from camera sensor. Same usage of the device id applies here: according to the id, correct tester device is taken as an active one and that is used for the communication.

2.2.2 Platform

MIP's platform handles communication between plugins on the playground meaning that the right information is delivered to the right plugin in right time. User connects plugins to each other by dragging a connection line from an output of one plugin to an input of another plugin. One plugin can output information also to multiple other plugins. Platform collects the plugins input signals and when all the data is received, it calls the plugin's Run-function which starts the execution of the plugin.

The platform uses multi-threading when starting to execute plugins. Every plugin is run in own thread. Only exception in threading is a thread for the graphical user interface (GUI). Every plugin's custom-made GUI is run in common GUI thread and this must be taken into account when designing plugins with custom GUI. If there is too much processing in the common thread, it will cause GUI to freeze when heavy calculation is processed and will lead to bad user experience because of slow response. That is why all time-consuming processing should be always done in plugin's own thread.

2.2.3 Plugins

There are three different types of plugins available in MIP: sources, filters and targets. Source plugins are the ones which starts the execution of the plugin chain and normally source plugin also generates or loads the data which is then somehow modified or new data is calculated from it. Source plugins can be started manually from the right-click menu, or in some cases from the plugin's UI, but in every case also by triggering the plugin with a trigger. Triggers are special objects in the playground which starts the connected source plugin after the connected filter or target plugin is completed. Usage of the trigger is shown in Figure 2.3.

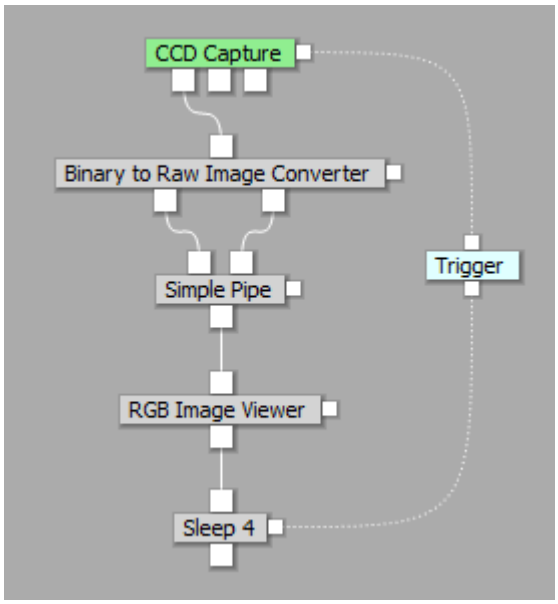

Figure 2.3. An example playground of usage of a trigger.

As can be seen from Figure 2.3, trigger is connected to at least two plugins, filter or target plugin triggers the connected source plugin. That way one can make a loop in the playground as in the example a continuous image capture with view finder is.

To start a filter or target plugin, all the inputs must get right type of data. There is one exception: plugin can have optional inputs. Optional inputs are marked as green inputs and those can be left empty if not needed. But if some output is connected to optional input, it must get the data from the output, otherwise the plugin's execution will not start.

All plugins must inherit BasePlugin class which inherits all the interfaces needed to be called by the platform. The most crucial function to be overridden by the plugin is the Run-function [6].

PluginResult Run(IList inputlist, IList outputlist)

This is the function that is called when the plugin is triggered (source plugins) or all of its inputs is received (filters, targets). All inputs are in *inputlist* parameter and for example the first input's data can be found by indexing the list by zero. Data comes as an object thus it must be converted to correct type before it can be used. Similarly, the outputs are added to *outputlist* and those are then outputted from the plugins output-ports. There is no need to convert data to object when adding it to *outputlist*. The data added to lists are in FIFO order, meaning that first object put in the list is the first one that comes out.

Each plugin has a user interface. The UI can be either automatically generated or custom made. When using automatically generated user interface, MIP shows all parameters in the UI that user is can modify. The custom UI may be designed as needed and it can include very complex graphical user interface components. The custom UI also allows using hidden parameters and those can be saved into a XML-file which is loaded automatically when loading the plugin to the playground. These hidden parameters are exactly same parameters which are visible in default UI, but in custom UI those can be used e.g. as a trackbar value container and thus can be hidden from the user.

2.2.4 Playground

Playground is the place where user drag and drops plugins on and then connects the plugins to each other. That way user can create plugin chains which usually manipulates generated data somehow. A simple example is a playground which loads an image from the hard drive and then displays it in another plugin. Most complex playgrounds might have dozens of plugins and loops and can contain for example full image processing pipeline. A single playground is saved as a portable single XML-file. MIP platform handles all the XML generation and reading so user do not have to have any knowledge about XML. Playground is loaded to MIP by opening the previously saved XML-file. MIP UI displays all the plugins saved to playground and their relationships between each other. An example of a playground was already presented in figure 2.3.

2.3 Technologies

MIP's platform is written fully in C# using Microsoft's Visual Studio integrated development environment (IDE). Most of the plugins are also written in C# but it is not the only option. A plugin can contain multiple projects and every project can be programmed with different or multiple languages [6]. For example, a plugin's graphical user interface (GUI) might be programmed in C#, Visual C++ or Visual Basic and the "core" part may be written in plain C. Visual Studio compiles a single dynamically loadable library (DLL) from each project so parts which contains different languages are compiled to separate libraries.

Usually a plugin contains two libraries and an XML-file. First DLL inherits the BasePlugin class and second DLL contains the core algorithms etc. The core DLL is optional so in some cases there can be only one DLL, or in case of larger plugin, even more than two. The XML-file contains plugin's documentation, configuration and interface. That is used when the plugin is drawn to the playground for user to connect it to another plugins.

As C# is highly object-oriented language, MIP takes an advantage of that. It uses classes, interfaces and inheritance very efficiently everywhere in the platform as well as when defining the plugins. That makes the plugin generation easy because of generic, common base of all plugins.

2.4 Camera Application Interface

Communication with a camera module is a relevant part of the thesis. In this chapter, an abstract camera application interface is defined to show what kind of functions the validation tests use in the communication. The interface defines basic functionality of a camera module as well as some additional functions used to control a frame grabber for example in power consumption measurement. The interface contains following functions:

- PowerOn
  PowerOn function rises voltage lines up in certain order to start the camera module. After that a specified register set is written to the camera module to set its state as needed.
- PowerOff
  PowerOff function stops the camera module from streaming if it was on and then shuts down the voltage lines in a predefined order.
- StreamingMode
  This function sets specified streaming mode on. Streaming mode specifies the sensor resolution, region of interest, bit depth, and data format.
- ChangeSettings

ChangeSettings has parameters to change exposure time, digital gain and analog gain.

- CaptureFrame

  This function captures a single frame using current settings. The requested frame will be the next available frame.

- ReadRegister

  Read a register value from the sensor over I2C.

- WriteRegister

  Write a register value to the sensor over I2C.

- ChangeLensPosition

  Drive lens to certain position. Position can be for example 10-bit value from 0 to 1023. At 0 position lens is at “far-end” meaning no current is driven to the voice coil motor and at 1023 position the lens is at “near-end” meaning focused as close as possible. In the near-end position maximum current is used.

- ChangeOisPosition

  Drive optical image stabilizer to certain position. In case of two-axis OIS, the position is described as x and y coordinates.

- Measure

  Measure current or voltage using frame grabber’s capabilities. Depending on parameters, current or voltage can be measured as a single measurement or by averaging several measurements.

3 CAMERA VALIDATION

This chapter introduces camera validation in a level of abstraction that is needed to be able to design and develop an automated camera validation system. To be able to validate a device one must know what it is, what it contains and what are the relevant tests to be performed. This chapter explains the basics of the camera module used in a smart phone from the relevant parts to the validation. After that the camera validation process and the test cases are presented. In the test cases, the focus is in those cases, which needs automation the most. In chapter 3.3 some of the instruments are introduced to give some concrete example of the used hardware. And finally, in chapter 3.4 the software side is taken care of. What is needed from the software point of view when validation software is being implemented and what kind of tools there is already in the market.

3.1 Camera module in smart phone

To be able to implement an automated camera validation system, one must know what is needed to test and why. This chapter introduces the device under test, a camera module used in a smart phone. Chapter briefly goes through what is inside of typical camera module and what is needed to test it.

A typical camera module consists of two main parts; image sensor and optics. Image sensor has a photosensitive silicon which contains from couple million to over 40 million pixels. Amount of the pixels is the resolution of the sensor. Most of the camera modules used in smart phones has a CMOS image sensor. The CMOS sensors have brought the prices down compared to CCD sensors.

Nowadays some high-end camera modules have also an optical image stabilizer (OIS). Typically, OIS moves either the lens system or the sensor. It uses gyroscope to detect device movement and tries to compensate the movement by moving the lens system or sensor in the opposite direction.

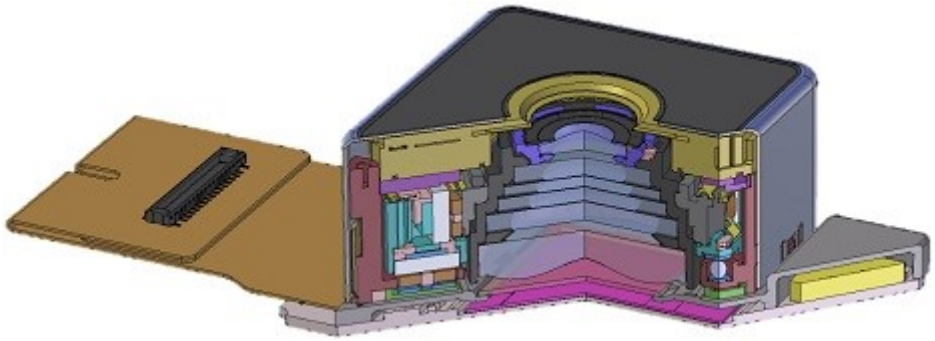

Figure 3.1. Camera module from Nokia Lumia 1020 smartphone [7].

The image above shows one of the most complicated camera module built for a smartphone. It is the Nokia Lumia 1020's camera module with 41 mega pixels back side

illuminated sensor, auto focus lens system and optical image stabilization. The module contains over 130 individual components and still its size is only 25 mm by 17 mm. [7] Other end of the smartphone camera modules are modules, which contains fixed focus lens system and small sensor, for example some front cameras might have 2 mega pixel resolution. This kind of low-end camera module's size might be only a few millimeters.

3.1.1 Sensor

Sensor is the camera module's heart, the eye, which captures incoming light focused by the lens system and transforms it to charge in the silicon. The silicon is sensitive to light over the whole visible spectrum and even a lot further. To be able to capture colorful images, the image sensor needs color filters. In normal camera module, there is an infrared filter on top of the whole sensor to block most of the infrared light. On top of every pixel, there is a micro lens and color filter to focus the incoming light rays into the pixel and block unwanted wavelengths. Most of the image sensors in smartphones are RGB sensors, meaning that each of the pixels has either red, green or blue pass-through filter on top, passing only certain range of wavelengths in to the pixel. Merging these R, G and B pixels in post-processing results a color image.

Complementary Metal Oxide Semiconductor (CMOS) sensors are nowadays the popular ones over Charged Coupling Device (CCD) sensors in smartphones. CMOS has some advantages over CCD. It is cheaper and consumes less power while nowadays can have good image quality. This makes CMOS sensors ideal for smartphones. There are of course downsides as well. Generally, noise levels are higher, and sensitivity is lower in CMOS sensors and thus overall image quality is lower. Compared to global shutter that is often used in CCD sensors, most CMOS sensors are rolling shutter sensors. Rolling shutter creates its own artifacts i.e. roller shutter effect. That is when something is moving fast in the subject that is being photographed, the object gets distorted. An example of this artifact is shown in the image below.

Figure 3.2. Rolling shutter effect visible in airplane's propeller.

There is an airplane's propeller visible in the image above. The rolling shutter effect cuts the fast-rotating propeller into many pieces. This effect comes from the sensor readout. In CMOS sensors, the pixel reading is done so that every row has its own readout hardware. It means that each of the row's pixels are read one by one and that causes a small delay between read of the first and the last pixel of the row. Since every pixel is wanted to be exposure exactly the same time, the pixel exposure needs to be delayed respectively to readout time.

3.1.2 Auto focus

Auto focus (AF) is essential part of camera module system to reliably obtain sharp images captured by the image sensor. In AF, the lens system, or part of it, is moved toward or away from the image sensor to focus incoming light rays from certain distance to the image sensor. Still today most of the camera modules with auto focus are using either voice coil motors (VCM) or piezoelectric actuators to move the lens system. There have been studies [8] and a few product implementations from microelectromechanical systems (MEMS) and shape memory elements like PoLight's TLens [9], but the VCM remain as the dominant technology.

Hybrid auto focus is a modern technology in the mobile phone camera field. It combines image plane phase detection AF and traditional contrast detection AF [10]. Image plane phase detection uses specially dedicated pixels distributed over the image sensor pixel array to detect the phase of incoming light rays. From the phase of the light rays can then be calculated to which direction and how much the lens system needs to be moved to obtain optimal focus. In traditional contrast based focusing the system calculates contrast from different regions of the image and adjusts the focus until the highest contrast is found. So far phase detection has been available only on more expensive DSLR cameras

which has own auto focus sensor. Sony IMX318 claims to be able to focus as fast as 0.03 seconds. It is possible by using the hybrid AF as well as doing the processing in image sensor's internal signal processor [10]. Earlier with contrast based auto focus the focusing might take up to a second, so difference is huge. This will bring smart phone cameras one big step closer to DSLRs since the slowness of the focusing has been a big issue especially when photographing fast moving objects as pets or children.

3.1.3 Image stabilization

Image stabilization can be done in two different ways; optically or digitally. Digital image stabilization can be applied to a video after shooting it, but for still images there are no good solutions available. Optical image stabilization compensates movement of a camera by moving synchronously either the lens system or the sensor. Camera movement may occur for example when user is taking an image with smartphone by free hand. Hands tend to start shaking when one tries to hold the camera as steady as possible while capturing an image. This shaky movement creates motion blur to the image. Very accurate gyro component detects the movement of the camera device and commands the actuators to move lens system to opposite direction [7]. Some cameras have OIS on sensor, where for example 3-axis electronic image stabilization does the compensation. Usually, camera modules used in smartphones uses OIS in lens system as in figure below.

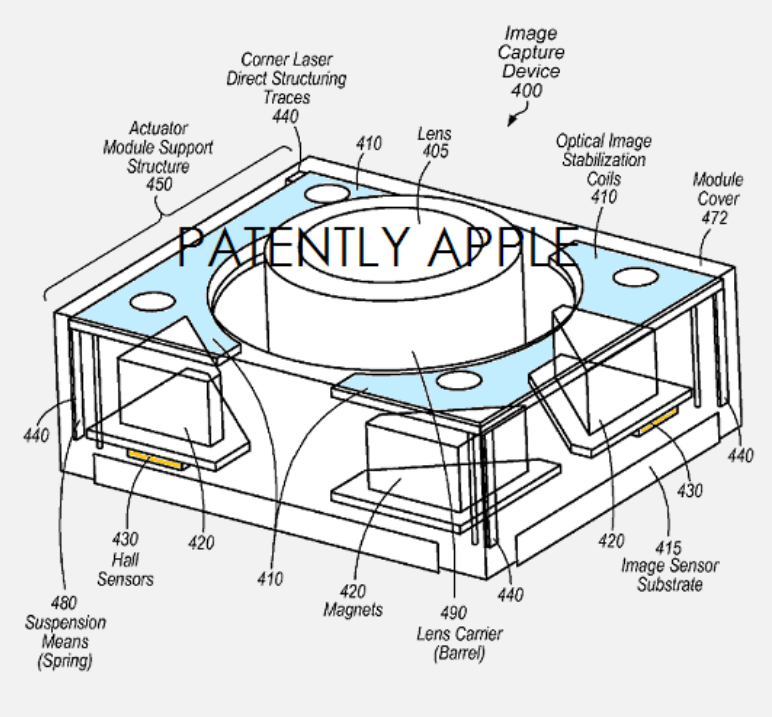

Figure 3.3. A solution for optical image stabilization patented by Apple. [11]

The solution above uses hall sensors, magnet coils and gyroscope to compensate camera movement by moving the lens system with electricity fed to the magnetic coils. The hall sensors provide a closed loop to the system for the best accuracy.

With optical image stabilizer, one can get brighter images with less motion blur especially in darker scenes because with OIS longer exposure times can be used when without it the images would become blurry because of handshake. The OIS benefits also video shooting where it can be combined with digital image stabilization for the steadiest and nice-looking video footage.

3.1.4 Standard Mobile Imaging Architecture

Standard Mobile Imaging Architecture (SMIA) specification is in 2004 announced architecture for mobile device's camera modules. It is an open standard and was designed by Nokia and STMicroelectronics and released for all third parties to use without fees or royalties. The specification's target is to standardize interfaces and system partitioning and thus be able to use any SMIA compliant camera module in any SMIA compliant mobile device or another host [12]. SMIA specification was an answer to rapidly crowing world of mobile imaging devices. Because there was no standard for mobile imaging, the market was full of different solutions.

SMIA specification specifies both hardware and software layer's basic functionality covering electrical, mechanical and functional interfaces. It standardizes also full register set used to control image sensor. Also, camera module characterization, its optical performance and reliability is included into the specification. On the other hand, SMIA does not specify image quality, leaving that to be designed in every camera model separately. This gives more flexibility to manufacture both high-end and low-end camera modules.

Being able to use standardized electrical and mechanical interfaces, customers can use second sourcing to reach mobile devices' high volumes. When using second sourcing the performance of the products must be able to measure in a common way. SMIA commits on that and presents methods to measure for example optical performance so the results can be compared between different products. [12]

In SMIA cameras, the SMIA specification gives very strict minimum requirements what comes to camera sensor functionality [12]. The specification defines the following fields: electrical interface and pin out, operating modes including power up and power down sequences, data format, video timing and cropping, integration time and gain control, how to report sensor capabilities, test modes, camera control interface register map, baseline profile, image scaling and compression, as well as recommended host behavior [13]. The sensor must fulfill all the requirements to be considered as a SMIA compatible sensor.

Bayer filter

SMIA focuses on raw Bayer output image sensors. Raw Bayer means that the sensor has a color filter array called Bayer filter on top of pixels which is well known method for arranging RGB images. The filter contains a pattern of three different colors, red, green

and blue. Each pixel is covered with one color and that way a single pixel has an illumination of one color. For one red and blue, there is two green pixels and those can be aligned for example like in the Figure 3.4 below.

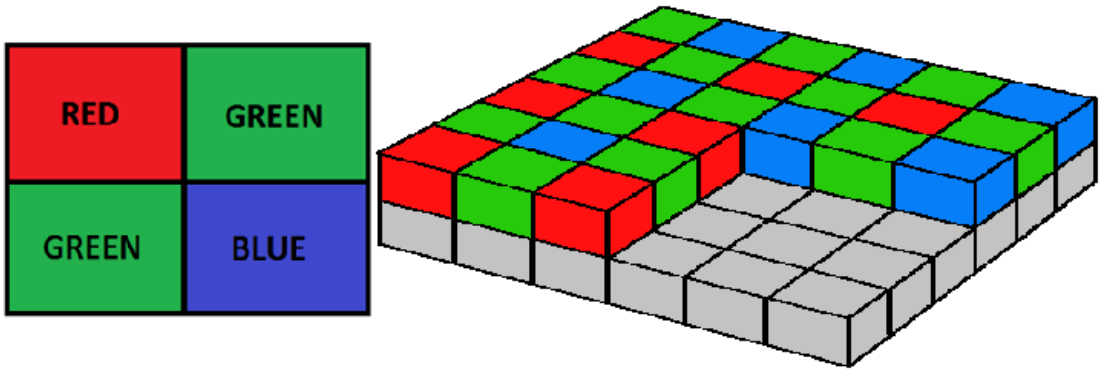

Figure 3.4. Bayer pattern color filter on image sensor.

The first pixel is only for the red color, second one is for the green color and then on the second line the first pixel is for the green color and the second is for the blue color. This kind of pattern is repeated through the sensor as can be seen in the right side of Figure 3.4. The lower grey array is the sensor's pixels and upon that is the color filter. Another thing which is visible in the figure above is that actually red and blue colors are present only in odd or even lines and columns. This means that 50 % of the pixels are green, 25 % red and 25 % blue. This heavily green weighted color pattern tries to mimic human eye as it is a lot more sensitive to green color than others. The color order mentioned above is not the only one SMIA allows to use. In Figure 3.5 is shown other three options.

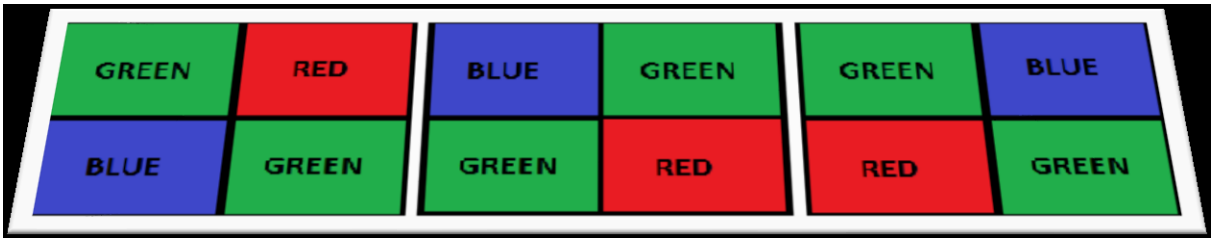

Figure 3.5. Other supported orders in Bayer pattern.

As can be seen from above Figure 3.5, order of the colors stays static, only rotation changes. There are also many other type of color array structures than Bayer pattern but SMIA uses only these four. Used color array order is defined in specific register in sensor.

SMIA camera modules doesn't have ISP on the module side so it transforms raw image to the host as binary data. It means that image needs to be processed on host's side to be able to save it in user friendly manner, for example in JPG format.

3.1.5 Camera Hardware Interfaces

To be able to transmit data from camera module to image processing device, a fast data interface is needed. Currently most widely used interface is Camera Serial Interface 2 (CSI-2). CSI-2 interface consists from one up to four data lanes, separate clock lane and Camera Control Interface (CCI). MIPI Alliance has a specification for CSI-2 where data transmission and control interfaces are defined [14]. Specification is created to replace old common parallel interface because it is not fast enough to support higher resolutions. High resolution and better dynamics generates a lot more data than before and that is why bandwidth must be increased. Existing parallel interface would be difficult to expand, and it would consume much more power. [15, p. 16]

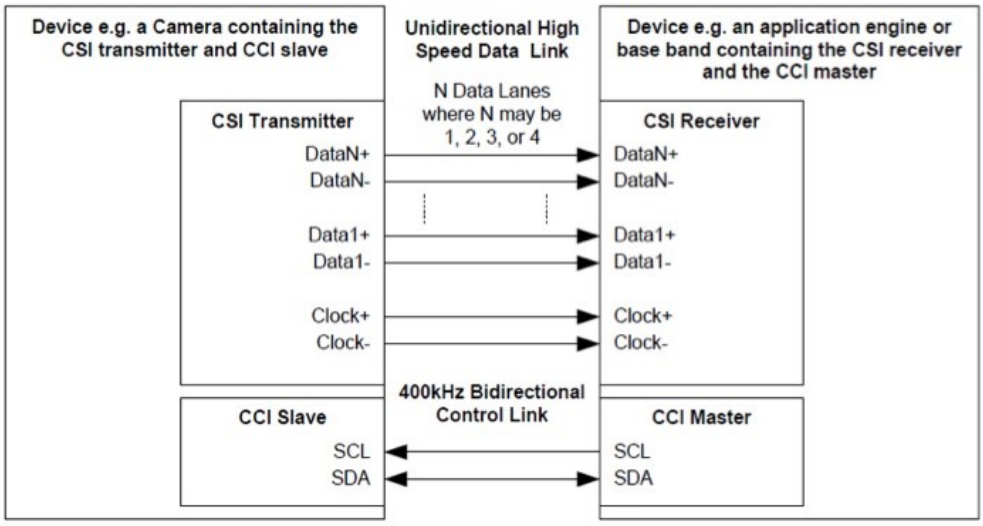

Figure 3.6. CSI-2 interface between a camera and a host device [15].

As the figure above shows, camera hardware interface consists of two parts: data transmission side and control side. CCI is compatible with Inter-Integrated Circuit (I2C), as it is a subset of the I2C protocol [15]. I2C is commonly used serial computer bus for controlling multiple devices, called “slaves” by host device, called “master”. CCI supports 400 kHz operation and 7-bit slave addressing. Multiple slaves in the CCI may occur if for example a smartphone has both front side and back side camera. Through the CCI master can read and write slaves’ registers to configure slaves in certain state and enable or disable features.

3.2 Introduction to camera validation process

During camera validation, the camera module is tested in multiple different ways. The tests are performed to see if the camera module fulfills requirements specified in SMIA specification and any other requirements specified to certain project on top of that. Typically, validation includes three categories: functional/hardware (HW), image quality (IQ) and electromagnetic compatibility (EMC) testing. There is even more testing done during

the validation, for example reliability testing, but these three categories are the ones which needs automation the most. Thus, others are left out of this thesis. Purpose of the thesis is to design and develop a tool which gives ability to build more automation to this kind of testing/validation, not to implement all the test cases. It would be way too big task to do as part of the thesis work. That is why next coming test cases are presented in an abstract way – not going too deep into the details. Basic understanding of test cases is needed to be able to design a proper test system.

3.2.1 Functional testing

In functional testing, the device under test (DUT) is tested through its interface and all functionality the DUT should support is tested. In case of SMIA based camera module, functionality testing is mostly testing that all the SMIA registers functions as meant, meaning that when writing to writable register and then reading it the value is changed and when writing to a read-only register the value is not changed. Some of the registers has a minimum and maximum limit. These limits are tested as well with at the limit and over the limit values. Then the functionality that should happen when the register value is changed must be checked. For example, if register which enables image flipping on the sensor is changed to flip the image, the output image must be checked that it is really flipped and only flipped if nothing else required.

Another part of the functional testing is to test DUT's power consumption. In mobile technology the power consumption is one of the most important things to be considered because of the limited battery size. The average current consumption of a sensor is measured over single frame when the sensor is streaming video with the fastest readout rate [13]. Limits for the power consumption are defined and for example image compression on the sensor affects to it. The current consumption is measured from each of power supply lines, i.e. I_{dig} and I_{ana}. These lines correspond to digital and analog voltage respectively. Total power consumption of the DUT is combination of the power lines and is calculated as in figure below.

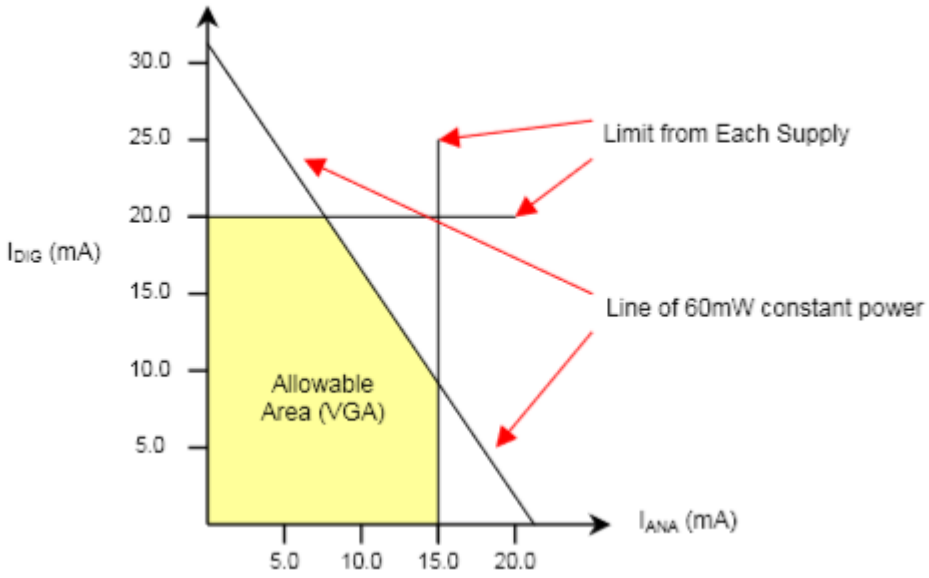

Figure 3.7. An example of an average power consumption. [13]

As can be seen from the figure above, both power lines have own current limit and the total constant power is limited as well. When the sensors functionality is tested, the average power consumption must stay within these limits.

Another example of a power consumption test is how the current rises when the lens system is moved from the rest position to another extreme position, i.e. macro. This is because all lens systems that uses voice coil motors to move the lenses, needs to have a spring to make counterforce. When the voice coil motor moves the lens system against the spring, it consumes more power. Each project has own limits where the camera module must stay within in every use case. Limits are made for standard use case and for worst case.

3.2.2 Image quality

Image quality (IQ) testing is a big part of camera module verification and validation during development phase. It includes various tests that measures camera module's image quality performance in multiple sections. IQ testing contains sections which measures inter alia noise, resolution, defect pixels, black level, saturation, camera uniformity (i.e. shading), sensor linearity and color performance. In this chapter, each of the performance measurements are described briefly.

Noise (Signal-to-noise ratio, Fixed pattern noise)

Signal-to-noise ratio (SNR) measures the ratio between the input signal and the standard deviation of the signal, meaning noise. Fixed pattern noise is noise that doesn't change between images. It is often seen as row or column noise in CMOS sensors caused by same hardware used to read full row or column. Both noise types can be calculated from a

uniform gray test chart. ISO 15739 standard defines the way to capture images for the test and how to calculate noise parameters. [16]

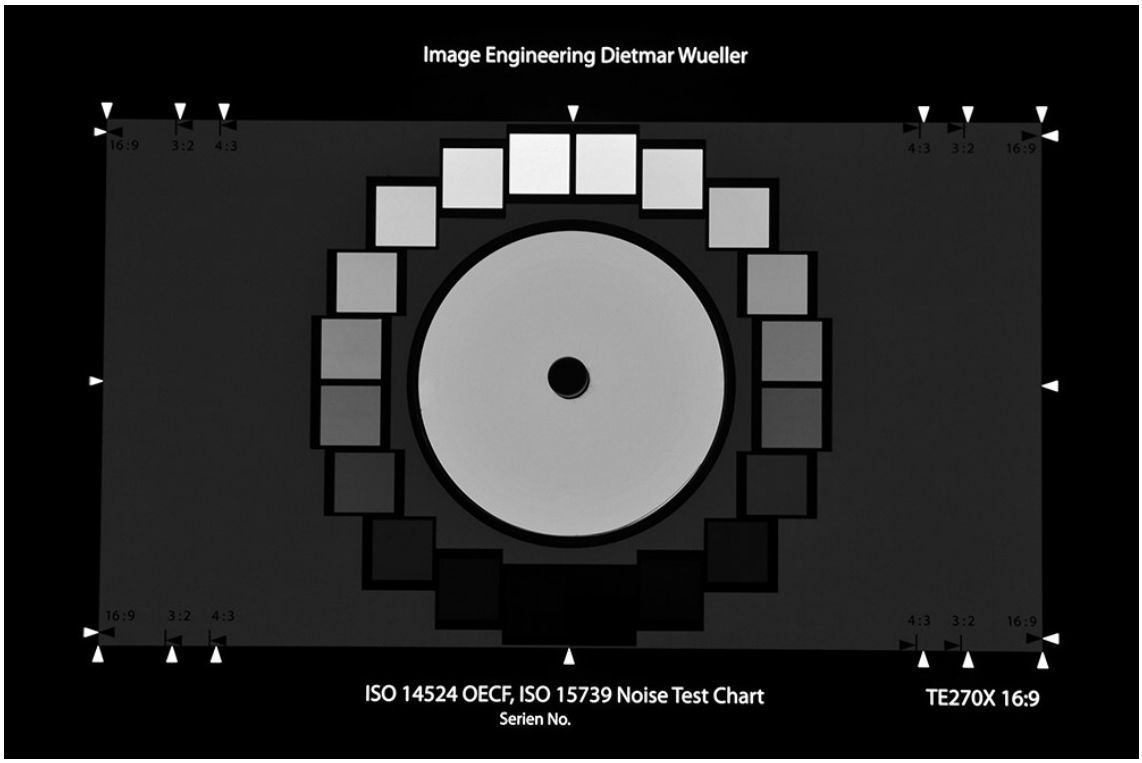

Figure 3.8. Noise test chart from Image Engineering. [17]

Image is exposed so that only the brightest patch is fully saturated. At least eight images are captured and those are averaged. Ideally a patch from the chart should be completely flat after averaging the images, meaning that all the pixels in the patch has the same value. If a pixel in the patch differs compared to average value, it is considered as a fixed pattern noise. The SNR=10 tells in which illumination level there is still ten times more desired signal than noise. The noise is the standard deviation of the measured patch. This is commonly used method to compare sensors' noise levels.

The general formula to measure signal-to-noise ratio is following:

$$SNR = \frac{\mu_{sig}}{\sigma_{sig}}$$

where

SNR = signal-to-noise ratio

μ_{sig} = the average signal of the image

σ_{sig} = the standard deviation of the signal.

Resolution / MTF - Modulation Transfer Function

In resolution test the resolution of the image is measured both from the corners and from the center of the image using sinusoidal Siemens star chart. This is vital, because lens system always introduces aberrations to the image and especially to the corners. These

aberrations decrease the resolution of the image. Image Engineering provides charts having either 9 or 25 Siemens stars being able to measure resolution for cameras up to 180 megapixels [18].

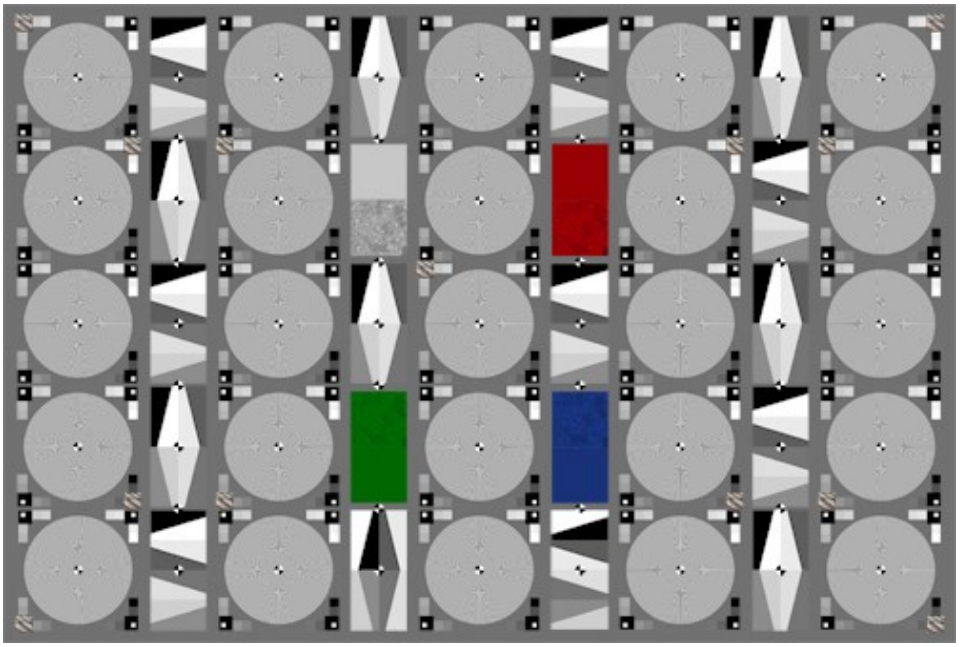

Figure 3.9. Lens resolution test chart TE268 from Image Engineering. [19]

The test chart above has 25 sinusoidal modulated Siemens stars and 16 slanted-edges at four different contrasts to measure resolution of a camera. The sinusoidal patterns are not that sensitive to sharpening than low contrast slanted-edges but are more sensitive to saturation because of high contrast. From the chart the modulation transfer function (MTF) and limiting resolution are measured from each of the Siemens stars.

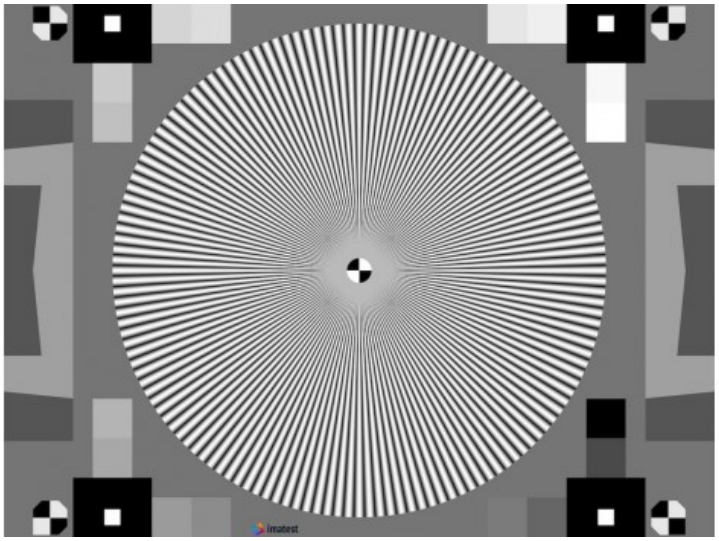

Figure 3.10. Sinusoidal Siemens star target captured by camera. [20]

From above image can be seen that in the outer part of the Siemens star the sinusoidal ring is well visible but when going more towards the center of the chart all there is left is gray. The MTF calculates the contrast of the sinusoidal ring. The MTF20 is the spatial frequency where MTF is 20 % of its low frequency value. This is used to measure the limiting resolution of the DUT [20]. Another outcome of the Siemens star is the Nyquist frequency (f_N). It is the MTF at 0.5 normalized spatial frequency i.e. half of the sampling rate. The figure below shows the result of the measurement.

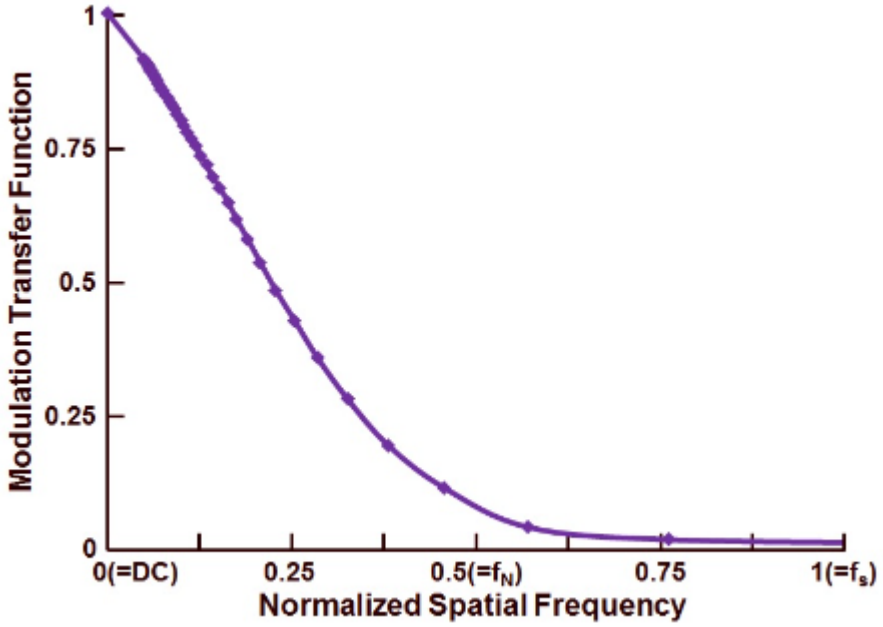

Figure 3.11. MTF measurement graph of an example camera. [21]

In the figure above, one can see the MTF curve drawn from dots. The dots are the measurement points measured from the sinusoidal rings. In this case the f_N is really low as well as the MTF20. The result combines the sensor resolution and the lens system performance.

Defect pixels / Blemish

Defect pixel is a pixel that doesn't perform as designed but either is stuck to empty or fully saturated all the time or the response differs too much compared to average pixel. Single defect pixels are easy to fix by software, but bigger clusters are more problematic. In RGB sensors clusters are often considered only if defect pixels are located next to each other but also in the same color channel. This means that is two defect pixels are right next to each other, but one is on red channel and another one is on blue channel, that is not a cluster but instead two single defect pixels. These two cases are presented visually in figure below.

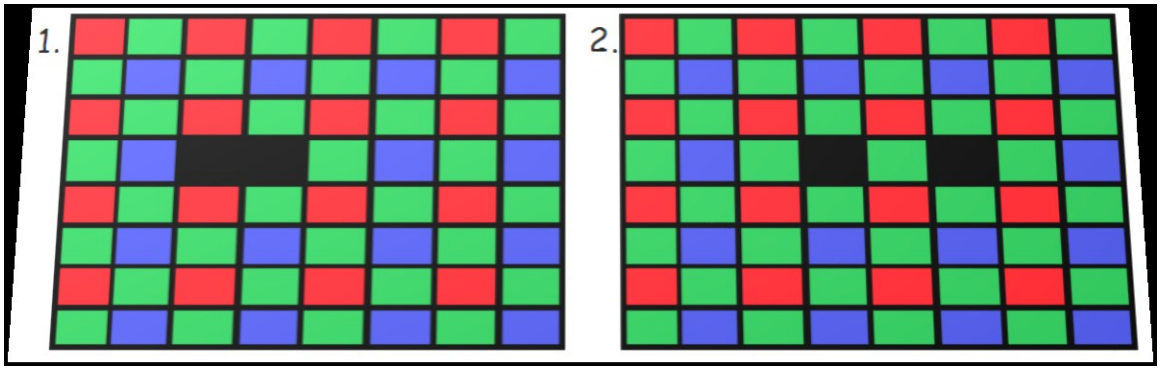

Figure 3.12. An example of defect pixels as two single defects (1.) and a cluster (2.).

In the first case of figure above the two black defect pixels are next to each other but in different color channel, green and blue. This is the two single defect case. In the second case, the black defects are apart from each other by one pixel but on a same row. That makes both defects to be in same color channel and that causes them to be a cluster. The sensor manufacturer often has a specification which says how many single, double or bigger clusters are allowed to be visible on the sensor. Often sensors get rejected, if it has a cluster with more than two defects.

Defect pixels can be calculated in two phases; only with bare sensor or together with its lens system. In first case, one can find defects that comes only from the sensor while in the latter case defects caused by the lens system are found as well. Sometimes it is hard to detect if there is dust on a lens or on top of the sensor or if the defect is in the hardware. The dust is problematic since even if the defect can be fixed by software, the dust particle can move over the time and cause defect in another location.

Black level

Black level measurement measures the level of pixels' capacity when no light is received into a pixel. Sensor's pixels don't stay "fully empty" when the pixel is reset, but instead some noise signal is introduced by dark current. Dark current basically means that electrons are created over time without light hitting to the pixel. Pixels captures electrons and are counted as a signal. Because of this, when the pixel value is converted to digital number in analog-to-digital converter (ADC), its value is not 0 but something else. Normally pixel's black level is set to certain level above zero, for example 64 (on 10-bit system), on sensor, but there are various parameters that affects to it. Most notable parameter that affects to dark current is temperature but also gain and exposure time influences to is. The longer the exposure time is or the higher the gain is, the higher the dark current is. Black level might vary spatially mainly because of sensor/camera module assembly so different parts of the sensor heats more than others. That is why black level needs to be characterized and corrected. Simultaneously white defect pixels can be found if any is present. Black level needs to be measured not only in different temperatures but also with different

exposure times and gains. Some higher end sensors have nowadays optical black pixels, which means that some rows of the sensor are covered so that no light can hit the pixels ever. These pixels give nicely very accurate black level data for every frame and it can be substracted from the real image. In this case only spatial variation needs to be taken care of.

Saturation

When defining a sensor's dynamic range, one must consider not only the black level of the sensor but also the saturation level. When too much light rays hits a pixel of the sensor, the pixel eventually reaches its saturation. After this point the pixel has reached its so called full well capacity and its value doesn't change anymore during the exposure. In digital sensors, this means that the pixel has stored the maximum amount of charge it can store. This is tried to avoid in photography since if the pixel gets saturated, most of its information is lost and that affects a lot for example to colors.

In the saturation test the sensor's all the pixels are tested if those reach the set saturation point. For example, sensor might have 10-bit ADC and often the saturation point is set to be its maximum value, 1023. If some pixels don't reach to that, those are considered as defect pixels. With some ADC settings, the saturation level might be lower than 1023. In that case the ADC needs some tuning which is exactly the result one wants to find from this test.

Lens uniformity

Lens uniformity (or shading, vignetting) measures a uniformity of an image i.e. how much illumination drops at the edges of the image. The phenomenon comes from physic laws that the incoming light through a lens is not equal in the center of an image and in the corners. It can be said that the wider the angle of the lens is the worse is the uniformity. Amount of light in the corners can drop significantly compared to the light coming to the center. This phenomenon is compensated by an increasing gain towards image corners. A set of images are captured from a gray, evenly illuminated test chart and the illumination difference between the center of the image and the rest of the image is calculated. This way a matrix of uniformity is created. The matrix can be whatever size is considered to be suitable for the case and it is then interpolated over the image.

Sensor linearity

Sensor linearity measures pixel linearity, meaning how linearly the sensor's exposure grows when the exposure time is grown. Linearity can be measured in percentages of nonlinearity, which is defined as:

$$Nonlinearity\ (\%) = \frac{D}{IN} * 100 \qquad [22]$$

where

Nonlinearity (%) is the percentage of nonlinearity
D is the maximum input deviation
IN is the maximum input.

In the figure below is shown an example of a measured curve, ideal fitted curve and the maximum error.

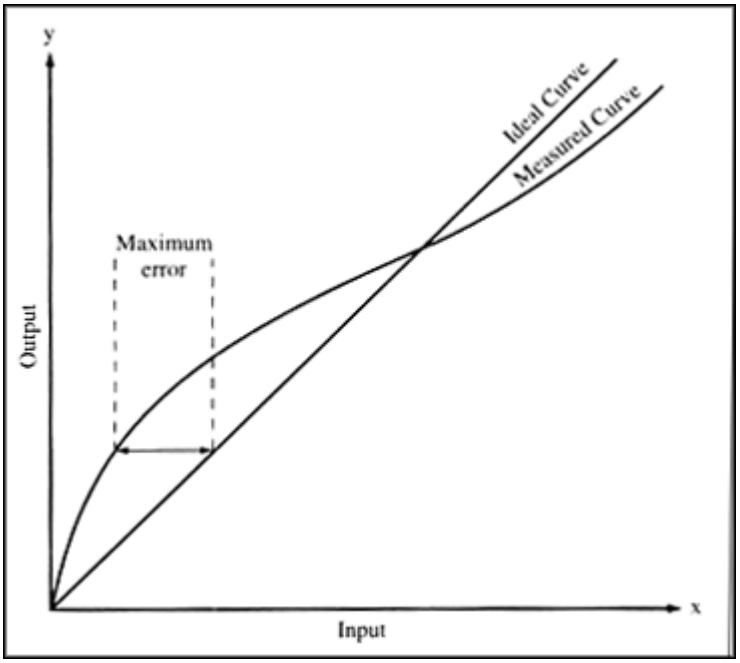

Figure 3.13. Nonlinearity of a sensor. [22]

The measured line shown in the figure above can be measured for example by illuminating the imaging sensor with a static light source and by capturing a set of images with different exposure time. Exposure times should start from so low that the signal read from the sensor is near or at the black level and eventually reaches pixel saturation with the longest exposure time.

Color performance

Color performance is one of the biggest issues in digital imaging and it takes a lot of time to tune the colors to look natural. In camera module validation developers are often interested how the infrared filter cuts the high frequencies of the incoming light and how are the pixels of RGB color channels receiving light in various color temperatures of illumination. The final colors are then fixed in image processing part and camera module's target is to provide as good raw data as possible.

X-Rite has provided widely used test chart for colors over 40 years. The ColorChecker Classic chart is presented below.

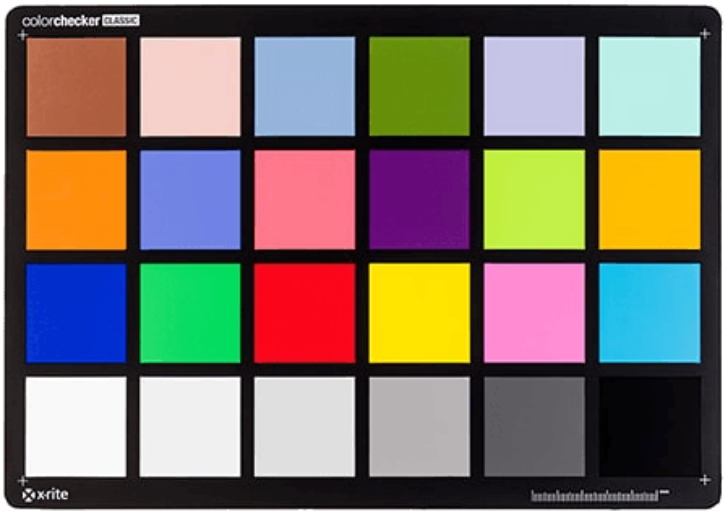

Figure 3.14. The ColorChecker Classic color test chart by X-Rite. [23]

The chart has 18 color patches and six gray patches. From the patches one can calculate the difference between captured color compared to ideal one. The chart is still widely used even tough more advanced charts are also becoming more popular. For example, similar chart but with 64 or 128 different colors.

3.2.3 Electromagnetic compatibility

In electromagnetic compatibility tests the camera module's performance is tested while feeding various disturbance signals into its power lines. Main frequencies are GSM, 3G, 4G, WLAN, GPS which are most common in every environment and these antennas are present in the smart phones and thus can interfere with the camera module. Main concern is to avoid visible row noise in image and that is exactly what is measured during the test. While the affected signal is feed to power line, an image is captured, and the amount of row noise is calculated of it. The signal is swept over the range of interest and the amount of row noise is calculated. The results can be plotted as a graph which shows the vulnerable areas which needs attention.

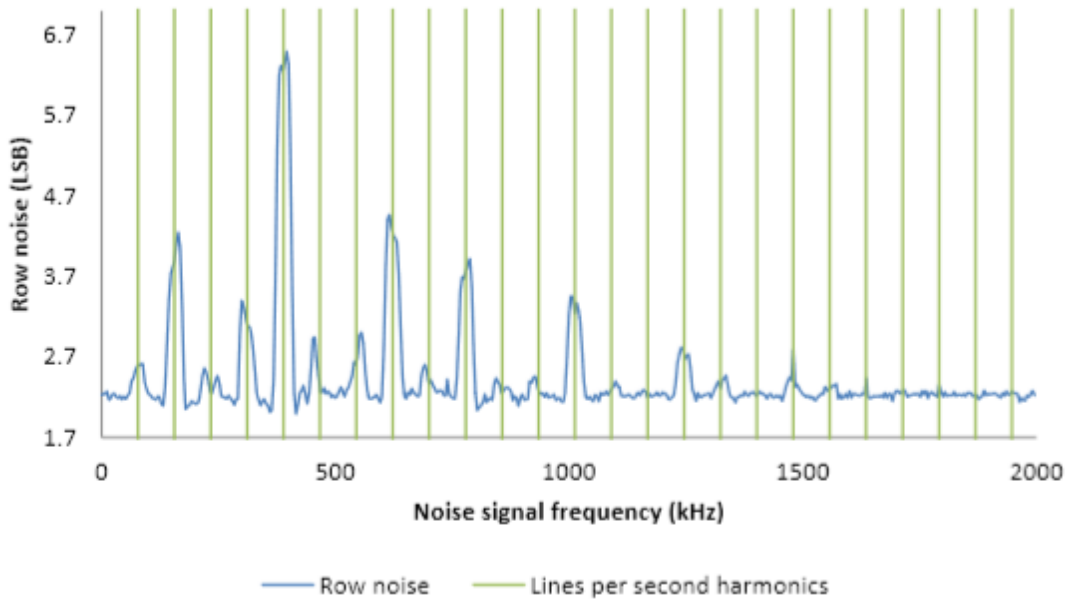

Figure 3.15. Row noise and line frequency harmonics. [24]

There are two ways to fix problems that becomes visible in this test. One can add more shielding to the camera module surroundings, or the camera module's internal clocks can be adjusted to get rid of these vulnerable areas. This latter case is a very nice example of an automation need because the test has many parameters to be changed independently, running the test takes quite a long time and the results can be calculated automatically. More info about electromagnetic compatibility can be read from Verification of camera module conducted immunity, Master of Science Thesis by Jussi Mikkonen [24].

3.2.4 Reporting results

Reports are one of the most important matters in the camera module validation. From every test, there must be a report saved for future evaluation. All the needed information must be seen from the report, for example which sample is tested, who did the testing, what was the exact settings of the sample module when tested and when the test was done. All the information is extremely important when comparing two test reports. If there is something different in the results it must be sure that the test setup was exactly same and only after that one can continue to investigate what is the difference between those two tested samples.

To be able to rely on test results which came from different locations, aligning the test environments must be done. It is done on inter-laboratory comparison (ILC). On ILC the same set of DUTs are tested in all laboratories and reports are generated. That way one can be certain that test results are comparable.

3.3 Test instruments

To be able to test a camera module in PC environment, a proper test board is needed. There are a few models available on the market which can be used but only few of those fills the requirements when high speed camera serial interfaces are required. Next two different types of test boards are presented. Atra Vision Inc. is a Canadian company that develops and sells Scooby2 platform which is made for MIPI and SMIA performance evaluation. Scooby2 supports CSI-2 camera interface and SMIA compliant image sensors [25]. Graphin Co., Ltd is a Japanese company which also develops and sells frame grabbers. Graphin has developed a GPirates series that also supports CSI-2 interface. Latest tester in the series is GPLAB-1500-8U. In this chapter both Scooby2 and latest GPirates are introduced and compared to each other.

3.3.1 Atra Vision Scooby2+

Scooby2+ is a modular platform for image sensor analysis. It has four-lane CSI-2 interface and FireWire interface to PC. CSI-2 interface supports 1.27 Gbit/s per lane. Scooby2+ is controlled and powered via FireWire connection. This single wire design makes it very convenient to use without unnecessary cable hassle. The camera module is attached on top of the device, with a camera adapter board which has 32 pins in two rows. The device is presented in Figure 3.16.

Figure 3.16. Atra Vision Scooby2+ MIPI test board.

3.3.2 Graphin GPirates

Graphin has developed a frame grabber that supports up to eight lanes in CSI-2 interface and speed up to 1.5 Gbit/s per lane. It has USB3 PC interface which enables much faster data sending from test board to PC. GPirates devices are more like production test devices. GPirates is not packed in a box like Scooby2 is but instead it is just a stack of circuit boards. In Figure 3.17 Graphin's test board is presented.

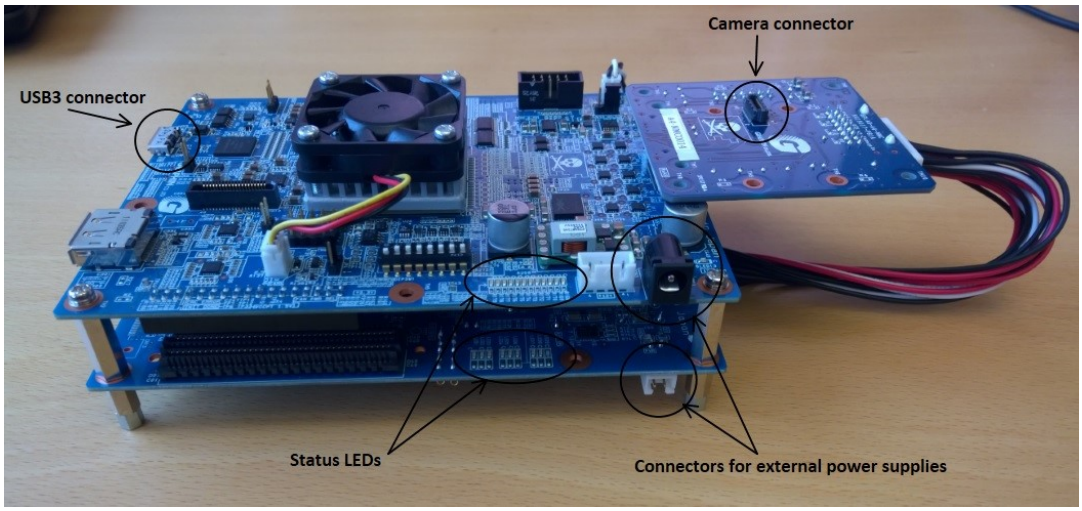

Figure 3.17. Graphin GPirates GPLAB-1500-8U test board.

The bottom board is a power board and all power lines come from there to the camera module. The upped board, the one with a fan is the main board. It contains USB3 PC interface, FPGAs, CCI and CSI lines to the camera board. Camera board is on the top right in the figure. It has a connector to the main board and another connector for the power board. A camera connector can be seen on the upper side of the camera board. Camera module is attached with a specific connector board to the camera board. Both the main board and the power board needs own power supply and has own connectors. Both boards have own status LEDs. Power board's LEDs indicates currently powered lines. Amount of available power lines is four and all of those are programmable.

3.3.3 Other instruments

To be able to measure and test camera modules, also other instruments are needed. Many of those instruments are PC controllable and hence possible to automate. There are tens of different test equipment and it is not meaningful to introduce or even list all here. But to give some impression of used devices, two very different instruments are shown here.

Many of the IQ measurements are done in certain illumination. To be able to illuminate a test chart right, a proper light source is vital. There are a few manufacturers that supplies that kind of PC controllable light sources. Image Engineering's (IE) LE7 is one that kind of light source, shown in figure below.

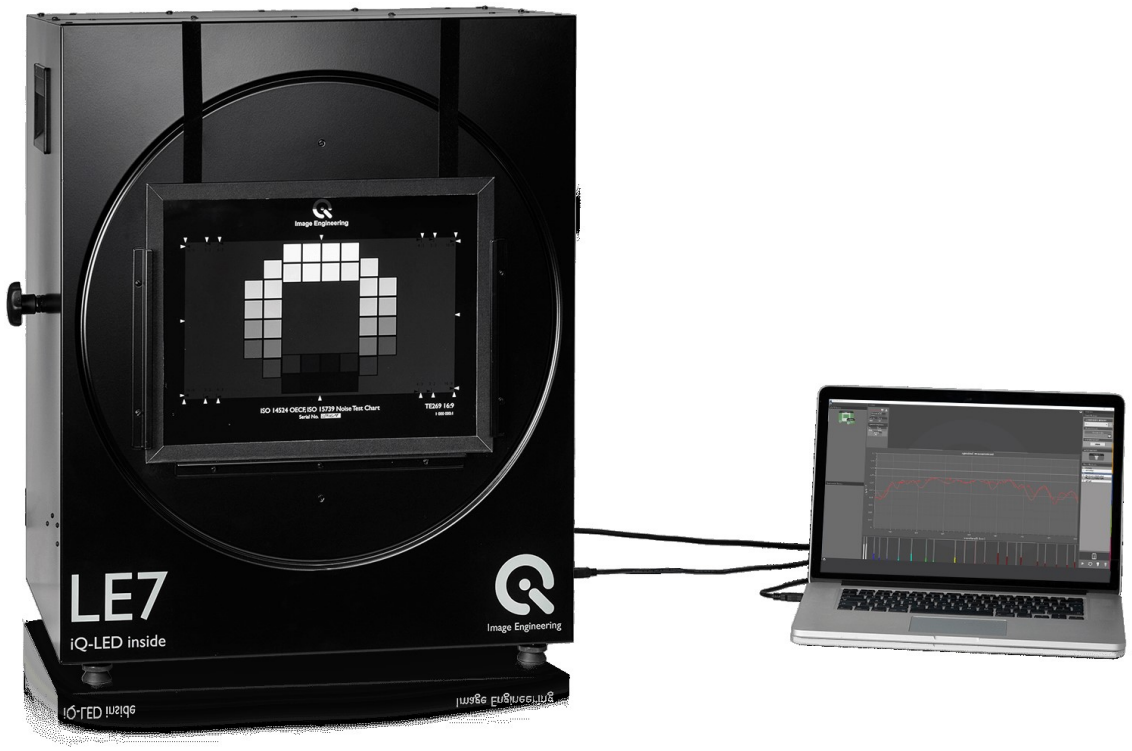

Figure 3.18. Image Engineering's LE7 light source [26].

LE7 can be connected to PC via USB and controlled using IE's GUI application or from a custom-made application using an application programming interface (API) they provide. The light source contains two to six iQ-LED multi-channel LED light panels with 22 different LED channels. Each of the LEDs are adjustable by software. The light source has also a spectrometer inbuild, so it can generate very accurate standard light sources with matching spectrums [26]. LE7 is a light source only for transparent test charts. If one wants to use reflective test charts, other type of light sources must be used, and uniformity of the illumination must be taken care of.

Camera modules are needed to test also in different ambient temperatures to characterize the modules in full temperature range of possible usage. To reach from -20 up to 70 Celsius degree ambient temperature, camera module is needed to put into a temperature test chamber, like one shown below.

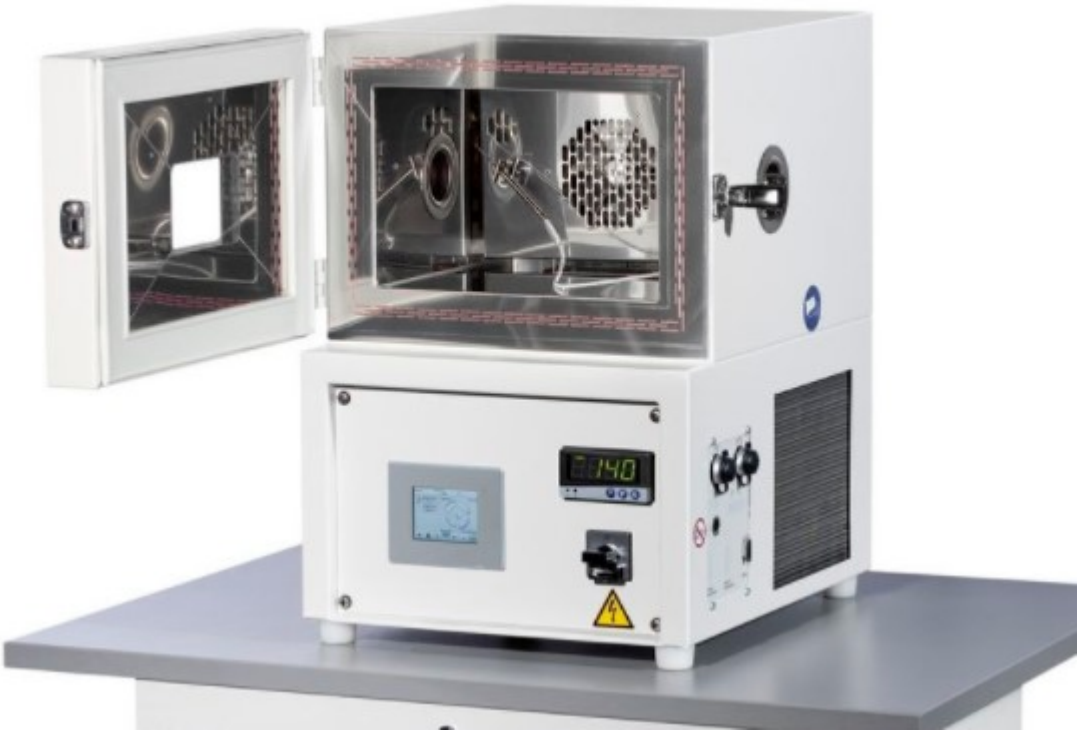

Figure 3.19. A temperature test chamber made by Vötsch [27].

Chamber can be also controlled from PC, so automatization is relatively easy. It uses old school serial port interface which nowadays can be converted to USB using converter. Basic usage is to put the DUT in to the chamber and set the desired temperature and poll the temperature until it is reached. Then execute the tests, for example the black level test and change the temperature to next one.

3.4 Test automation software development

There is not that many test automation software available in market since often companies needs quite unique test automation and hence it seems to be easier to build it from the scratch than try to integrate existing automation tool to company's own tools. In most of the cases there is no public documentation available from systems which are not available in the market. When designing a test automation software, understanding of the DUT is the most vital. The DUT defines what is needed to test. Environment and user defines how the tests are done. In software designing the environment dictates the way software should work. For example, a tool used in research and development (R&D) is used very differently than one in a production line. In production, the test software needs to be 100 % tested and fulfill defined standards. This is because products tested in production are then sold to customers, so the products must be top quality. On the other hand, in R&D the test software can have an easier release process with more features which might not

be fully tested yet. There must be a clear border between production and R&D test software so that R&D software never goes to production without a proper release process and validation.

The need of automated software tool comes often from a user that has been doing certain tests manually for some time. The user finds out that some parts of the tests are repeated over and over again either with same parameters or with multiple parameter sets. This kind of test is a great starting point of test tool automatization since the process steps are often clear and the variable parameters can be listed easily. Often it is seen that next step is to write a command line script that calls certain functions with the different parameters. This is good enough if the test case is small and is not going to get much bigger in the future. But as devices gets more complicated over time, also tests begin to be more complicated, long and result dependent. Amount of scripts for different combinations of tests and hardware gets too big so cleverer tool is needed to handle all that.

3.4.1 Existing test automation systems

When looking for available test automation systems on the market, there is one product that keeps popping up, LabVIEW. It is National Instruments' powerful development environment mainly for test and factory automation development. It supports a wide range of different instruments used in hardware testing. LabVIEW uses visual programming language called "G" which enables user to make software by building and connecting graphical blocks to each other by wires. An example of LabVIEW block diagram is shown in Figure 3.20.

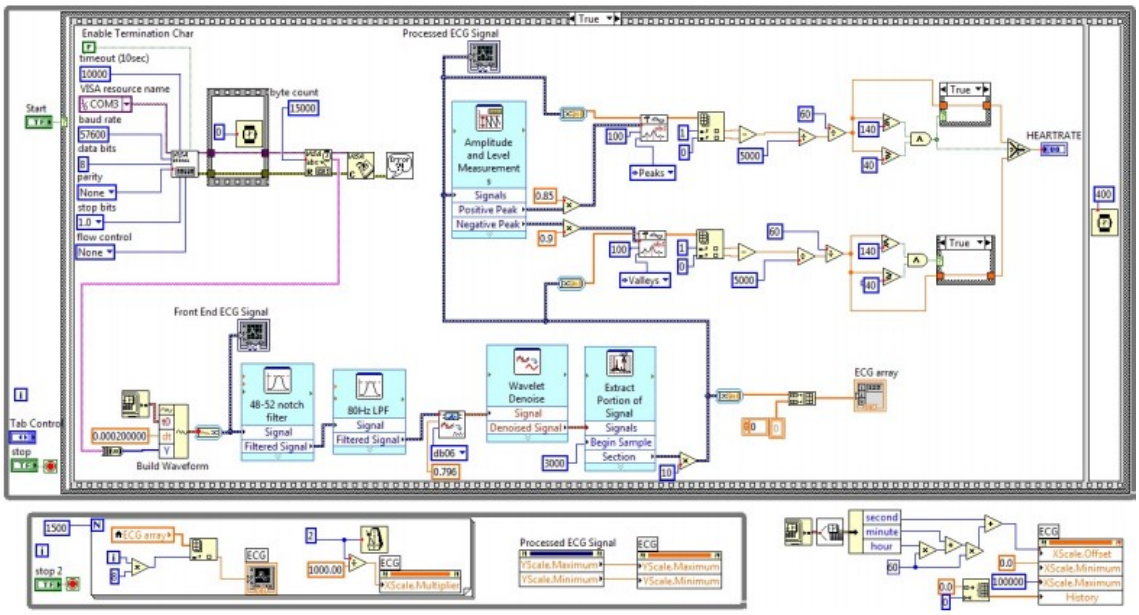

Figure 3.20. Example of LabVIEW block diagram [28]

From the figure above can be seen how blocks gets data in as inputs, or as National Instrument calls, controls. When all controls come in to a block, its code can be executed. A block has also an output, called indicator which is the result of a block. Combining

multiple blocks together one can build complex programs from scratch using only graphical user interface. LabVIEW has libraries which contains readymade blocks for certain tasks. If user wants to do more complex operations in one block, user can create new block and make it do whatever is needed.

3.4.2 Chosen method and alternative approaches

LabVIEW might have been a great alternative to use instead of MIP because it is also really flexible system with quite good support of test and measurement instruments. Decision was anyway to further develop existing inhouse software. The main reason was the already existing integration to support test hardware and camera modules which wouldn't have had readymade support in LabVIEW. Also, if LabVIEW would have been chosen, it would require buying multiple licenses because the system is wanted to use also by camera module suppliers. With company's own software there wouldn't be any issues related to licenses.

As the main dilemma with MIP was how to automate a software that has only graphical user interface and no console or scripting features. The only feasible solution to the problem was to develop a scripting framework on top of MIP's GUI. This enables efficient use of existing library portfolio and to enable the power of scripting in sense of functions, loops, variables etc. tools needed for reusable automated test case building. In the next chapter the whole scripting environment is described in detail.

4 GENERIC SCRIPTING FRAMEWORK

When considering usability of the MIP in camera module testing the biggest downside has been the lack of automation regarding to controlling the camera. So far almost all the camera controlling has been done manually from MIP's graphical user interface. For example, in Conducted Immunity test, there was already available automated testing which automatically changes frequency from a function generator controller, measures a row noise from an image and reports the conducted immunity. Only thing missing was the automatic camera control since one had to configure the camera module before starting each of the tests. And if, and what is usually done, one wants to test same test using multiple camera settings, the camera settings must be changed manually and then repeat the test. That leads to conclusion that testing is semi-automatic since there is a need for manual work between test sequences. This again leads to situation where the testing is done in the shortest way where the camera parameter sets are cut to minimum. This happens because doing the full test with wide range of parameter sets takes too long and has too many steps for human to execute. A generic scripting framework presents a solution to this problem by introducing a possibility to script among other things also these camera parameter sets for better automatization.

With the scripting framework one can control not only the camera but also all the other test devices and instruments within one script file. Script syntax is very high level and thus easy to understand and write also only with moderate programming expertise. Anyway, a basic understanding of programming is required. In this chapter, the generic scripting framework is defined, and its implementation is introduced.

4.1 Advantages and requirements

Building scripting on top of existing software enables an option to use existing algorithms if separated "core" project was developed when the original plugin was made. That way developing script plugins in MIP is fast and efficient. If the core project was not developed to the original plugin, it must be separated to be able to use same algorithm without copying the code. Otherwise algorithm should be rewritten and that generates easily errors and different behavior when using scripting instead of the original plugin. Also, algorithm maintenance is a lot easier when there is only one implementation of the algorithm. The main idea is that there is no difference in test results whether the test is done with script automation or manually.

When using scripting most of the human errors can be isolated and thus the results are more reliable. That is because in readymade script, all the parameters are defined so the user doesn't have to manually select those each time the test is executed. On the other hand, the script automation must be done so that user can easily change certain parameters without need to read whole script file through. This is because every camera module type

has own setup of parameters because not all camera modules has same hardware components. For example, the number of power lines may vary from one to four lines and each of the used lines must be tested.

So, the solution must be highly flexible, easily modifiable and understandable. One must be able to add new features when for example new instruments are taken into use. With these requirements, the generic scripting framework was developed on top of MIP.

4.2 Architecture

Architecture of the scripting on top of MIP can be divided in three parts. First part is a Script Parser plugin, which parses and validates the script file and sends one command line at the time to other script plugins. Script Parser-plugin's type is source, so the execution of the script is started from this plugin. Second part is all filter typed script plugins, which each has own specific task, for example control of an instrument. Third part is a Script Reporter plugin, which collects test results and finally generates a test report. In the figure below, a very basic script playground is presented.

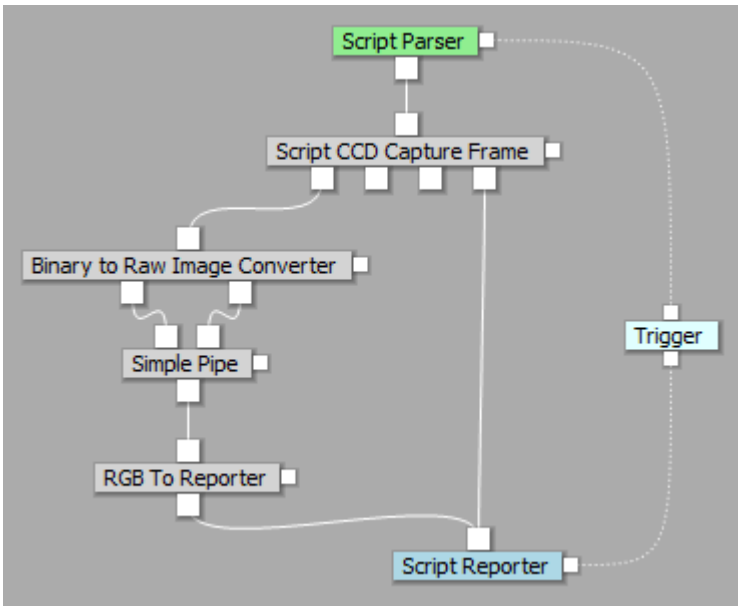

Figure 4.1. A basic script playground.

As can be seen from the figure above, Script Parser has only one output, which type is Script Packet. Script Packet object contains three parts: a command to be executed, a list of measurements or other outputs where a script plugin can add measurements to, and a list of errors where script plugins may add exceptions if those occurs during the execution of the command. Script Packet is carried through all the script plugins and only those plugins that recognizes the first word from the command as their keyword takes the packet and executes it. Other ones just pass it through. Usually there is just one plugin that executes one command, but it is not strictly limited to one. So, basically there could be more than one plugin that obeys a single command. In that case, the plugins will execute same

command in order they receive the command on the playground. When the packet has gone through all the script plugins, it finally reaches to the Script Reporter -plugin. Script Reporter is the last plugin in the script chain as it collects all the Script Packets together and generates reports from those. After receiving the packet, Script Reporter triggers Script Parser to execute the next command.

4.3 Implementation

In this chapter, all the relevant parts of the generic scripting framework implementation are presented. The implementation mainly concentrates on exporting all relevant UI features of a set of plugins to be used from the script. As most of the test cases were possible to execute manually from the UI, this was the simplest way to start the automatization.

4.3.1 Parser

Script Parser has three different states: read file, execute and finish. In read file state, a script-file is read into the PC's random-access memory, it is parsed line by line and syntax error checks are done. The procedure when parsing the script is as following: Read all lines from the script-file and go through all lines. From every line; collect tags, remove comments and remove extra whitespaces from the beginning and from the end of a line as well as if there are multiple spaces between words. Let user fill the tags in UI. After these steps, all the loops and lists are opened to executable code. This means purely copying the contents of a loop as many times as there are rounds and filling the variables to the code. This method was chosen because it is good to "compile" the script before execution to get rid of possible syntax errors in the code before starting the test. Some tests might take several hours to run through and most probably the test would get ruined if some of the cases were not executed in the right way because of typing error in the script.

Unfortunately, above mentioned syntax error checking only applies to variables and how lists and loops are created. Every script filter plugin has its own syntax and the parser doesn't know that. This could be improved by implementing filter plugin metadata, which could manifest its syntax and possible parameters.

After the "compilation" the parser goes to second state, execution. Each of the lines are executed separately, line by line. The parser creates a Script Packet object and adds the executable line into it. The line is split to parts and stored to Script Packet as a vector of strings. The parser doesn't do any conversion but instead outputs each parameter as a string. The filter plugin, which then executes the line, will handle the conversion if necessary. This enables use of more general parser which means that whenever a new filter plugin is introduced, the parser will remain the same.

The GUI of the parser looks as in figure below.

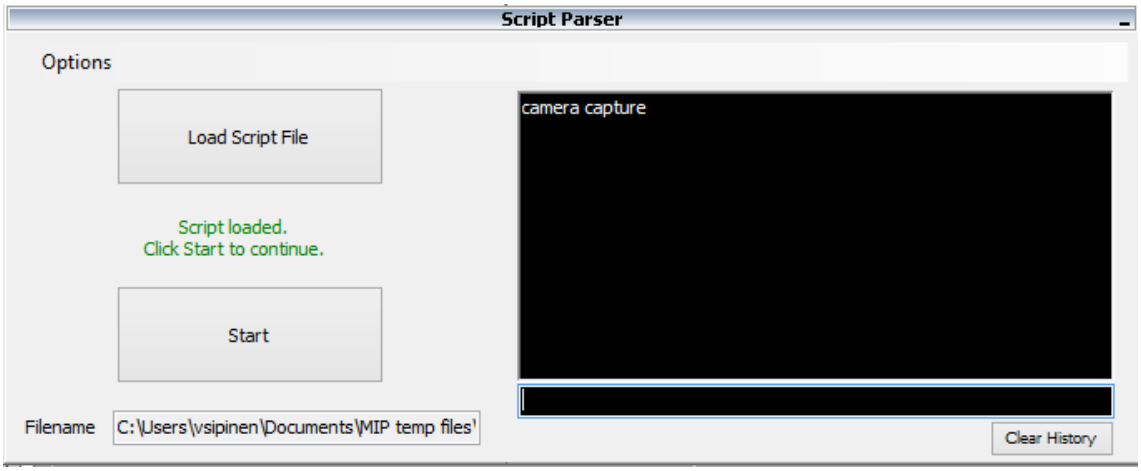

Figure 4.2. Parser plugin's graphical user interface.

The GUI has two sides. On the left side is the default view, with two buttons, options and filename label. The upper button loads the script from a file and does the syntax check. The lower button starts the execution. The right side of the GUI can be enabled from the options. This part is for ad-hoc testing or script development. It contains a console where one can run single commands through the built plugin chain.

4.3.2 Tags

User may use tags in a script to change parameters in the script without modifying the script itself. Tags are set to the script so that any word can be replaced with the tag. When user fills the tag in UI, it is then replaced with user's input. Tag's syntax is like <--tagname--> and "tagname" can be set to whatever, for example <--settings file path-->. This makes use of the script more manageable between multiple testers, because for example the settings file path can be set on each PC to use its own path regardless of the file structure. Same tag can be used as many times as needed in the script and same text will be replaced to each of those. Also, the number of different tags are not limited. When user fills tags in UI, filled fields are saved to <script-name>.tags file where filename is same as the script file. This enables automatic tag loading to UI when the user next time loads same script file. User may modify tags as needed. An example of filled tags in UI can be seen in Figure 4.3

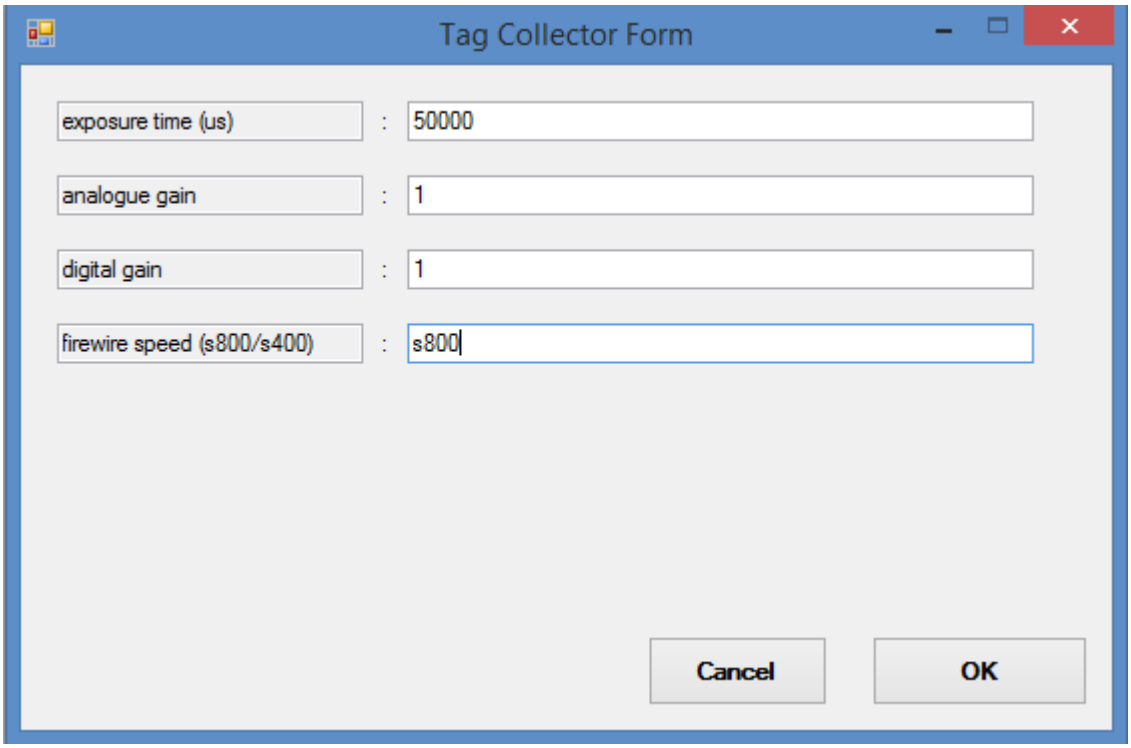

Figure 4.3. Filled tags in a form.

In the example above the settings of a camera module are stored as tags which enables easy modification. Here the exposure and both analog and digital gain can be modified as well as frame grabber interface speed as some testers might require slower speed to function properly. In the script, the exposure tag is presented as <--exposure time (us)--> so actually the tag name straight defines name of the tag on UI as well. This simplifies the entire process as no extra metadata is needed.

4.3.3 Script Trigger

In scripting, there is every now and then a need to start some other process, do it sometime and then get back to original process. For example, run an auto focus algorithm after every 5 capture frames. Script Trigger makes it possible to start another plugin chain in the middle of execution. Two plugin chains are presented below to illustrate that.

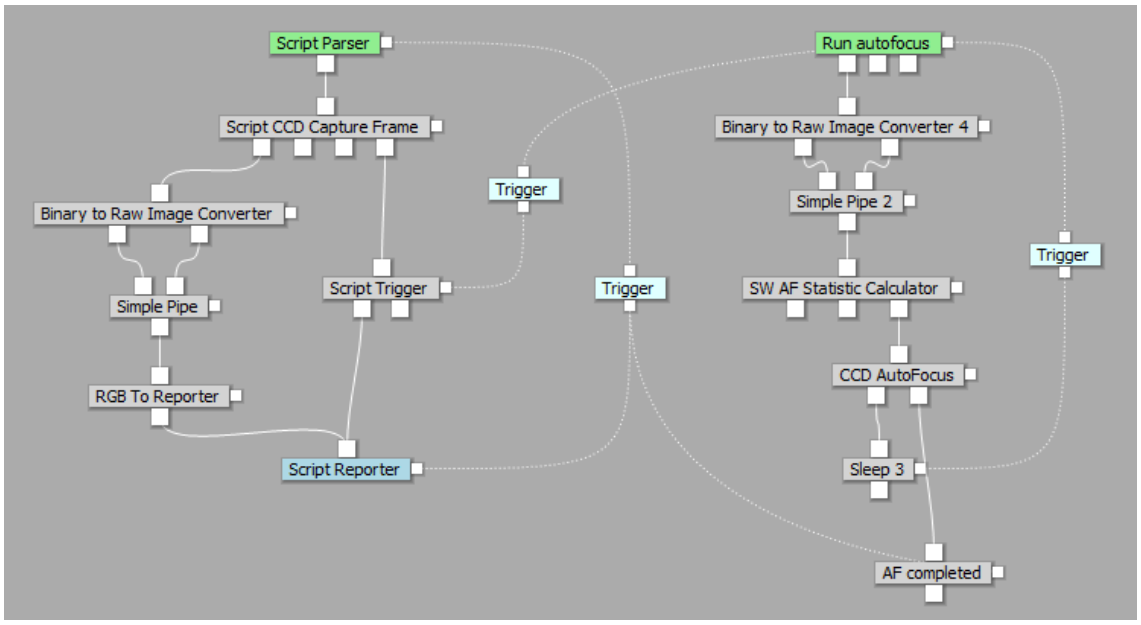

Figure 4.4. Auto focus example using scripting.

The left plugin chain in the figure above is the main chain that captures test images. The Script Trigger plugin starts the auto focus chain on the left when it gets its keyword as a parameter. The keyword is defined in the plugin's GUI as in figure below.

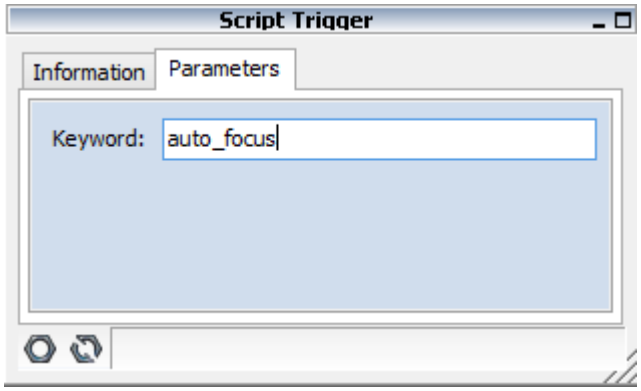

Figure 4.5. Script Trigger plugin's GUI and the keyword parameter.

Once the Script Trigger receives the keyword, it starts the chain visible on the left side of the Figure 4.5. When the auto focus chain has reach to its end, it triggers again Script Parser and the script continues its own execution. Downside of this solution is that errors occurred in the auto focus chain are not caught by Script Reporter, which is definitely something to be changed in the future.

4.3.4 Script language

The script language is a list of commands, one per line. Each command is passed through all the plugins one at the time. All script filter plugins have own keyword, which specifies if the plugin takes the command for execution. If the keyword doesn't match plugin's one, the plugin passes it through to next one without further processing. After the keyword

the rest of the command can include whatever parameters as long as the plugin can understand it. When developing new plugins, the developer can decide what kind of syntax is to be used with the plugin. Idea of the basic syntax is like:

Keyword command1 [command2 ...] (parameter1 [parameter2 ...])

So, after the keyword there is a first command which can have more subcommands like command2 and command3, as many commands as needed. After commands, there can be as many parameters as needed. A couple of example commands are shown below:

camera capture

The first example command consists of two words: camera and capture. Camera is the keyword for camera controller plugin and capture is the command. This line captures an image using a connected camera module. The command could also have single parameter, number of images to be captured. If the number is specified in the command, camera will capture that number of images. In that case, the command would be like:

camera capture 5

Second example has multiple commands:

camera set vdig 1.9

With the command above, the digital voltage (vdig) of the camera is set to 1.9 V. In that case, the set command tells that the voltage must be set before powering on the camera, or that the camera must be reset before the setting is applied. Thus, all the commands that has the “set”-command in front of, must be set the same way.

Even though the voltage is a floating number, the plugins gets it as a string. All parts of the command line are treated as strings and each plugin must handle type casting of the parameters. It enables light script parsing and easy developing of the script plugins because all type handling is handled when data is used. This brings us to the error handling. Because the script parsing is done light way, parsing the errors from the parameters is left to the filter plugins. That is why the Script Packet has own error field, where the plugins can add errors which occurred in parsing or execution. Errors has own severity parameter that lets a filter plugin abort the script execution in case of fatal parameter error. Fatal parameter error can be for example if parameter needs to be float but it cannot be converted to float. Less severe errors might be for example if camera is unable to capture an image because of off limit camera parameter. That case would be logged into the log file as well as to the report, but execution of the script would continue normally.

4.3.5 Script-plugin development

Scripting in MIP is designed to be easily extendable with new plugins and features while maintaining backward compatibility. That means one may develop new plugins that uses scripting on the fly and no changes are needed to the platform or Script Parser -plugin. Every machine or instrument type has its own script-plugin. Each script-plugin defines its own syntax, only limitations are that the first word must contain the plugin's keyword and the syntax must contain at least two parts e.g. *camera capture*, where "camera" is the keyword and "capture" is the action for the plugin.

Script plugin must have exactly one input, an object with type of Script Packet. That is because every script line is passed through all the plugins in the plugin chain. A plugin is run only when all the inputs are ready, i.e. some other plugin's output data has reached to every input. So, if the plugin would have multiple inputs but the current line in the script would not be for that plugin, the plugin would not get all the inputs ready and would not ever run. This would lead to deadlock and thus is not allowed.

The script packet's type is Script Packet and it contains elements defined in IScript-Packet interface below:

```
class IScriptPacket {
 public:
    IList<string> GetScriptCommand();
    void AddError(PluginError error);
    void AddOutput(ScriptOutput output);
    string ToString();
    bool Trigger { get; set; }
};
```

Script Packet interface is quite simple and that is basically everything what is needed when implementing a script-plugin. When the Script Parser parses a script, it adds a line to script command list. That list contains all parts of the line: keyword, action and parameters. The script-plugin gets the list by calling GetScriptCommand method. Once the script-plugin has executed the action, it can add results to the output list by using AddOutput method. Results are packed to Script Output object which contains caption for the result and a list of results. This list can have several different types of results, including:

- integers
- floating points
- strings
- lists
- tables (class)
- figures (class)
- bitmaps.

Results are added to the list as objects. Unrecognized types are converted to strings and printed to report. Same practice goes with errors. In case of error in script-plugin, an error can be added to Script Packet by using AddError method. The error is added as PluginError which can contain additional details of the error as well as exception, if occurred. Errors will be added to the report. Depending on error severity, some errors are just added to the report, but execution of the script continues. Severe errors, for example loss of camera connection, stops also script execution.

There are still two other methods, ToString and Trigger. ToString converts the command list to string for easier printing. Trigger is a Boolean which is used for informing the Script Report plugin if there are still lines of script to be executed, so it will trigger the Script Parser plugin again. In case of severe error, a script-plugin may change the trigger to false to be able to stop script execution.

4.3.6 Reporting

Automated testing is nothing without a proper reporting. One can test automatically many things but if there are no results available, one would think that the tests are irrelevant. So, generating a report from the tests is vital. Many times, actually the report generation is the biggest effort in testing. One needs to collect test parameters and test results from each of the test, then add those to e.g. Excel and create graphs and tables out of it. The generic scripting framework tries to help also on that and introduces automated reporting. It lets user to define in the script, what is added to the report in each test. The measurements can be added for example as a graph or table. An example graph is shown below.

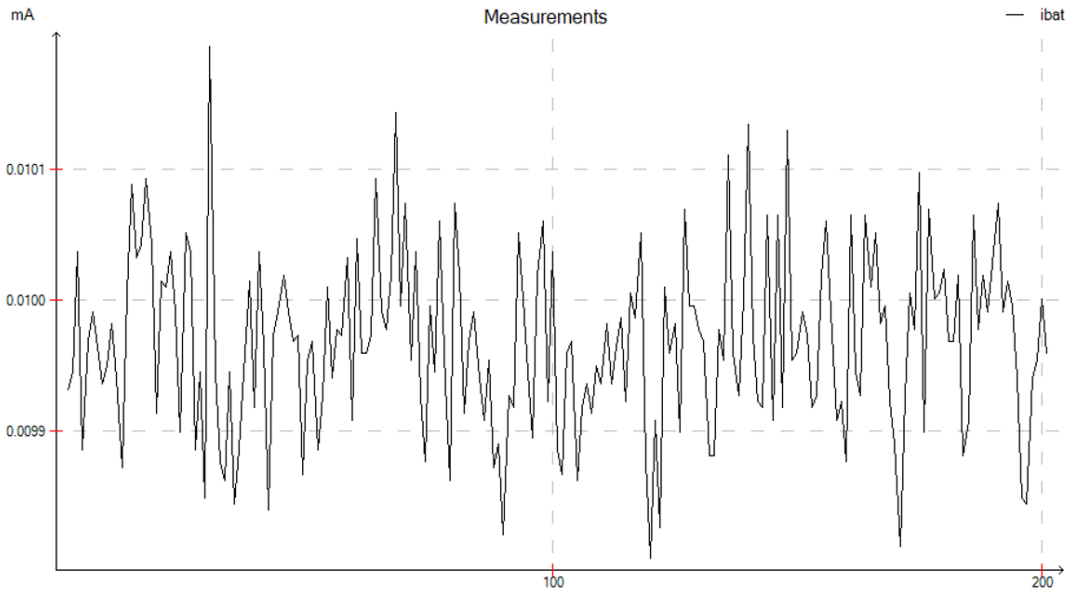

Figure 4.6. Graph of a measured power consumption from a camera module.

The graph presents measured power consumption (current) of a main power line from a camera module. On y-axis units are in milli-amperes and on x-axis measure count. Measurements are done in certain frequency, for example 1000 measures per second. From the graph there is also min, max and average values presented as a table.

	Average	Max	Min
ibat	0.01	0.0102	0.0098

Figure 4.7. Min, max and average values from a power consumption measurement.

The values presented in the table above gives a good overview for reader of the report. In addition to these values the graph gives valued information from power consumption. Often, when measured power consumption over several frames, the frame figure is visible in the graph as can be seen from the following figure.

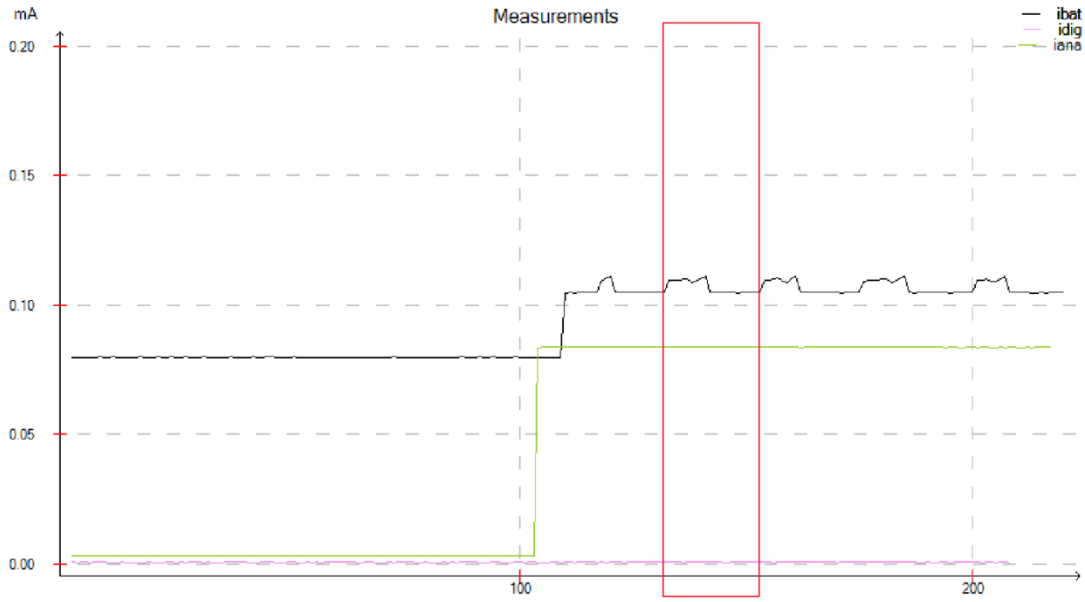

Figure 4.8. Power consumption after stream start.

Peaks in the frame figure can reveal issues in hardware configuration which may lead to too high-power consumption. This is highly valuable information and is tested using several different camera module parameters like exposure time, gains, different clock speeds etc. Scripting makes automatically graphs from each of the cases so comparing results is easy and efficient.

In addition of the reporting, also a full log file is saved separately, so one can check what parameters were used in each of the test cases if something suspicious pops up in the report.

The report created by the scripting contains a front page, an optional summary page and unlimited amount of separated measurement pages. Each measurement page has a title and possibility to add multiple graphs, tables, images or plain text. All objects are collected to the page and distributed evenly once the page is “closed”. The report is saved

as PDF format for the most convenient file sharing. PDF documents are well compressed, portable and easy to send for example via email.

Script Reporter plugin also handles errors what may have occurred during script line execution. It adds occurred errors to the report and depending on error severity either continues the script execution by triggering Script Parser for next line or stops the execution and saves the report. The GUI of the reporter is presented below.

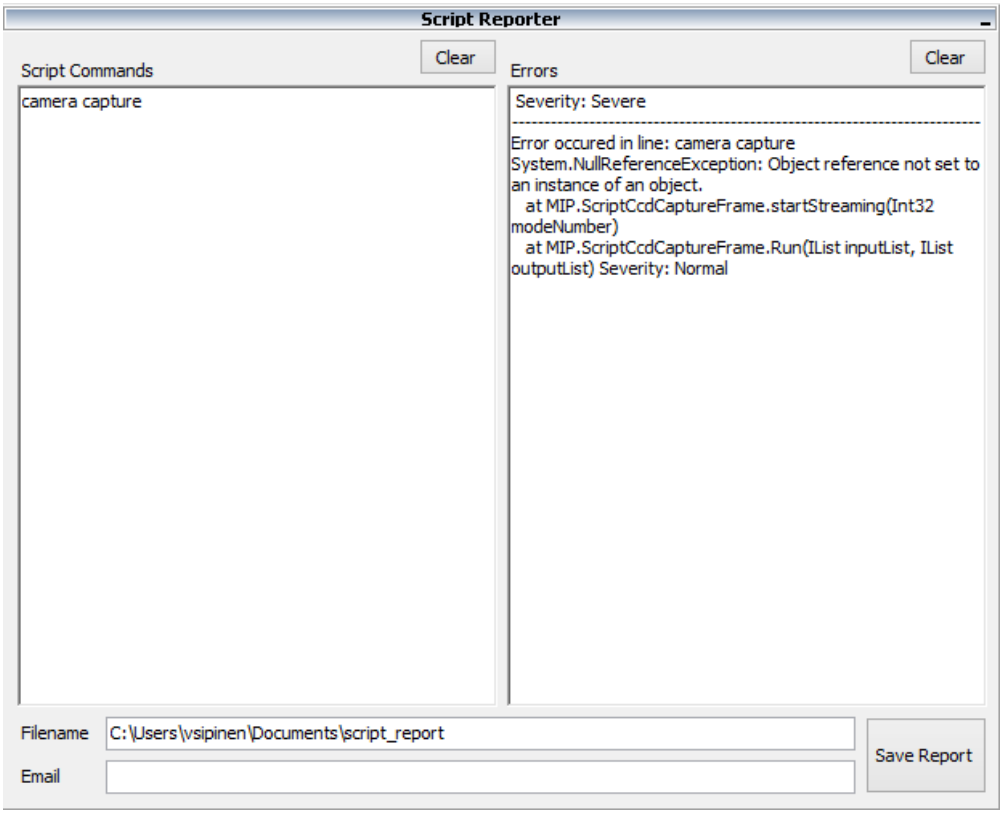

Figure 4.9. Reporter plugin's graphical user interface.

On the left side one can see the executed commands in a list and on the right side all the possible errors are listed. The report can be also saved manually by filling the filename and pressing save report button. User can get an email notification about the finished script execution so there is no need for waiting the result in the lab or to poll unfinished test.

5 EVALUATION OF THE TEST SYSTEM

To evaluate how the newly developed tool works in practice, it is good to compare it to how tests were done previously. In this evaluation the current consumption measurement is examined in old manual way and with the new automated test tool.

In mobile products where the device is powered by a battery, the power consumption is one of the most important matter to be optimized. When overall power consumption is limited to very low, importance of the power consumption of a single part of the device is emphasized. The current consumption of the camera module is measured to be able to calculate the total power consumption. Measurements are made in multiple conditions and with several different settings to identify the maximum and the average power consumption.

5.1 Manual testing

Until now the power consumption test has been a manual test which has taken several days to execute fully. The test contains leakage current measurement, hardware and software stand-by current measurement and measurement when camera is streaming. These states are tested in various conditions by changing voltages in power lines, with different external clock frequencies, with different combination of actuator states and with different sensor settings. In each of the test cases an engineer sets the parameters and reads the current consumption from the power source. The value is written to the test table including minimum, maximum and average consumption. For example, when totally over a hundred test cases are tested in three different temperatures to 10 camera module samples, it is quite clear that there is a huge need for automation to get rid of that huge human workload.

5.2 Automated testing

The current consumption measurement is done by generating a script and a playground into MIP. The script defines all settings meaning the different camera module parameters where the current is measured and what power lines are measured. Different types of script can be made to do the testing as needed in different phase of the camera module development project. For example, more accurate and more test cases including script can be run for the first sample version and later only most critical test cases can be executed. Also, if the current consumption measurement is done in different temperatures, camera module can be placed to a temperature chamber and the chamber can be controlled from the script.

Another great thing in automated testing with scripts is that there is no need for engineer to fill test reports because the script reporter plugin does it automatically. That means

when a test has finished, the report is also ready to be delivered for example to the project manager.

When designing new automated test, also new script needs to be generated. Generating the script file and especially testing it can be somewhat time consuming. General guideline is to keep scripts or functions relatively simple for easier maintenance and combine multiple scripts for bigger test system.

5.3 Test system comparison

Following short example explains concretely how the scripting works.

```
1. #Lens positions
2. camera mode 0
3. camera measurescan start ibat
4. loop var lpos_ 0:3:1023
5.         camera lens lpos_
6. endloop
7. camera measurescan stop
8. reporter newpage "Lens positions"
```

Script snapshot above is used for measuring the battery powerline, ibat, while camera module's lens system is moved from one extreme position to another. First, on line 2, the camera is stopped from streaming. Then the ampere meter, which is integrated to the frame grabber, is turned on to measure ibat. In this case, the lens system is controlled via 10-bit register, so values from 0 to 1023 with step of 3 are gone through in a loop and set as the camera's lens position. After the loop the current measurement is stopped, and this also outputs the measurement data to the reporter. Lastly the page in the report is created with caption "Lens positions". This script outputs a report page with following graph and table.

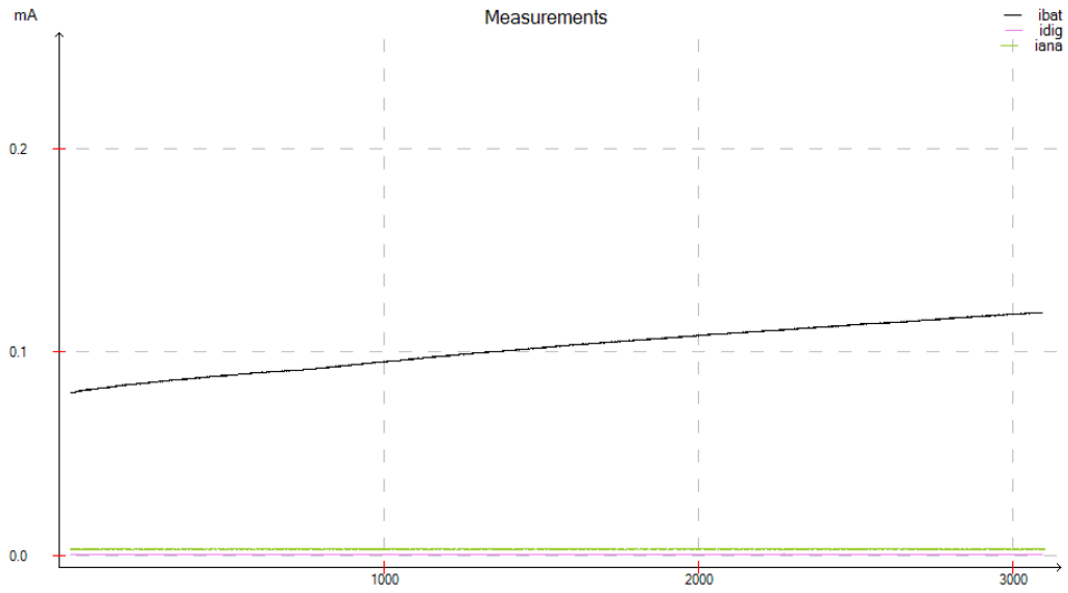

Figure 5.1. Current consumption of a lens sweep.

The graph above shows how the current rises when the lens system is moved from rest position to another extreme position, i.e. macro. On Y-axis are milliamperes and on X-axis number of measurement. In this case, the total number of measurements is a bit over 3000 so almost ten measurements per lens position. In the test the lens was moved from 0 to 1023 with step size of 3 meaning that total number of the test cases were 341.

	Average	Max	Min
ibat	0.1018	0.1195	0.0799
idig	0.0007	0.0009	0.0005
iana	0.0032	0.0034	0.003

Figure 5.2. Measured current thresholds from three powerlines.

The figure above shows all powerlines and those used current. Total power consumption can be easily calculated when we know the used voltage by using well known formula

$$P = UI,$$

where
P = power (W)
U = voltage (V)
I = current (A).

Often the current consumption is enough if the voltages are fixed.

The scripting collects automatically data points to a graph and calculates average, max and min values. By doing this with scripting, running the test takes only a couple of seconds, whereas doing the test manually is far more complex and time consuming. One must set the lens system to each of the positions one at the time, then read the value from a power source or multimeter and write the value to a table. Once all points are measured, one must still make the graph and calculate the averages etc. and fill these to the report. Doing manually this test will take even some hours, is quite sensitive for errors and most likely some corners will be cut, and test will have a lot less data points measured.

When comparing this relatively simple measurement task done in manual and automated way, one can note the following. The test system instruments are the same, so no new equipment purchase is needed when moving towards test automation. One must create the script in the beginning and test it through. This can take up to a work day including training. Running the measurements takes just a few seconds compared to manual multi-hour test. Afterwards the results are available right away, without any data processing compared to manual data gathering and processing. The automated test outputs a lot of data points, which can be averaged to get rid of any noise present in the measurements. This can't be done with manual way, one can only trust what the multimeter displays. The current peaks are lost in manual mode which are very interesting and caught in automatic mode. The peaks could be detected in manual mode by using an oscilloscope but then the setup and measurement execution would get even more complex and slow.

There are of course some doubts with fully automated tests as well. For example, if there comes an error during the test, it might be challenging to find the reason or what causes it if it is in the system. Most of the errors comes when the user initializes the system with wrong parameters, so following the instructions is most vital thing to do. More advanced error handling helps but developing such is very complex and takes time. Anyway, error handling is easier than in the manual mode where if the user types an error to the document, there is no way to find out later that it was an error and not a real measurement.

6 CONCLUSIONS

In the camera module validation, the number of test cases has been rising constantly as the used sensors and mechanics are getting more complicated and other accessories, like optical image stabilization, are being added into the camera modules used in smart phones. This thesis describes one solution which helps to overcome this massive pile of test cases bringing a way to automate these tests. The automation helps to validate camera modules more reliably by enabling effortless way to repeat tests and get credible results. It also speeds up the testing by an order of magnitude and at the same time reduces the change of human error to minimum.

Keeping up with the future requirements in fast development of new products, it is clear that often repeated tests must be automated as fully as possible. A lot of time and money is wasted when these tests are executed manually. Not only the spent time but also one might eventually get bored when repeating same tests over and over again with manual test's slow phase. Carelessness is one of the main reasons errors occurs in the tests.

As the goal of this thesis was to design and implement a test platform for a camera module validation, the example use cases has shown benefits of the tool as well as that it can be used for real product validation tests. The tool is convenient to use and easy to expand with new plugins to support any PC controllable instrument in the market. The automated reporting does create useful reports with all the related information packed to an easy-to-use PDF document.

The main idea of the generic test platform was to have a platform that is easy to continue developing and to be able to use existing parts of MIP software. The scripting tool became easy to use and many of the test instruments got already transformed to support scripting. This thesis proves that also in camera module validation the automatization can bring a lot of benefits in matter of speeding up testing and reducing the change of errors in measurements. Both of these benefits are great for reducing unnecessary costs.

6.1 Future improvements

As in every project, there is always room for improvements in the future. As a last chapter, let's look at the improvement opportunities found during this project but left out because of timely manners.

Automated report generation is a great start, but delivery of the report to ones that are interested of it is needed. As the Script Reporter does send a notification after the test is done, obvious next step would be to attach the report to the email. That way the owner can get the results right after the measurement and the test station can be in some other location for example in a laboratory in another floor or building.

In current consumption measurement, the sample rate was not as high as it could be. As now the Scooby 2 could measure current in roughly 2kHz frequency, the optimal frequency would be at least 10 kHz or even 100 kHz to be able to capture also the shortest peaks of the current consumption. This is a hard challenge, since this most likely will require some external measurement device. Using Scooby 2 as a frame grabber and a measurement device is a big benefit as it makes the test setup so much simpler. When developing the future frame grabbers, that could be taken into account to get more precise accuracy.

What comes to the script language, one thing to improve would be better syntax checking. The syntax check could be improved by implementing a filter plugin metadata, which could manifest its syntax and possible parameters. That way all the script lines could be checked for syntax errors already when parsing the file. This would prevent error cases like having a typing error in the late part of the script parameters which would lead to an error after most of the script is already executed. Also, a script development GUI with syntax highlighting and batch generator would make the script generation much more convenient and simple for non-programmers to do.

REFERENCES

[1] Statista, "Number of smartphones sold to end users worldwide from 2007 to 2016 (in million units)," 2017. [Online]. Available: https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/. [Accessed 28 3 2017].

[2] A. Chowdhury, R. Darveaux, J. Tome, R. Schoonejongen, M. Reifel, A. De Guzman, S. Soon Park, Y. Woo Kim and H. Wook Kim, "Challenges of Megapixel Camera Module Assembly and Test," in *Proceedings Electronic Components and Technology*, 2005.

[3] V. Garousi and F. Elberzhager, "Test Automation: Not Just for Test Execution," in *IEEE Software, vol 34*, 2017, pp. 90-96.

[4] R. Anbunathan and A. Basu, "Automation framework for test script generation for Android mobile," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, Bangalore, 2017.

[5] Gamma, Helm, Johnson and Vlissides, "Elements of reusable Object-Oriented," in *Design Patterns*, USA, Addison-Wesley, 1994, pp. 124 - 130.

[6] A. Knuutila, MIP Developer Document, Tampere: Nokia, 2014.

[7] Microsoft Devices Team, "The magic camera ingredients inside the Nokia Lumia 1020," Microsoft, 20 September 2013. [Online]. Available: https://blogs.windows.com/devices/2013/09/20/the-magic-camera-ingredients-inside-the-nokia-lumia-1020/#xAr6uQJqLos05hyB.97. [Accessed 09 May 2017].

[8] K. F, M. W and Z. H, "A MEMS-based variable micro-lens system," Department of Microsystems Engineering-IMTEK, University of Freiburg, Germany, 31 5 2006. [Online]. Available: http://stacks.iop.org/1464-4258/8/i=7/a=S06. [Accessed 7 1 2018].

[9] PoLight, "PoLight AS - Technology & Products - The TLens Products," PoLight, 2018. [Online]. Available: http://www.polight.com/technology-and-products/the-tlens-products/default.aspx. [Accessed 7 1 2018].

[10] Sony Corporation, "Sony Announces a New Type 1/2.6 22.5 Megapixel Exmor RS™, the Industry's First*1 Stacked CMOS Image Sensor with Built-in Hybrid Autofocus and 3-Axis Electronic Image Stabilization," 2016. [Online]. Available: https://www.sony.net/SonyInfo/News/Press/201602/16-013E/index.html. [Accessed 29 3 2017].

[11] J. Purcher, "Apple's iPhone 6-Plus Optical Image Stabilization Invention Comes to Light," Patently Apple, 15 7 2015. [Online]. Available: http://www.patentlyapple.com/patently-apple/2015/07/apples-iphone-6-plus-

optical-image-stabilization-invention-comes-to-light.html. [Accessed 21 11 2017].

[12] Nokia ans ST, SMIA 1.0 Introduction and overview, 2004.

[13] Nokia and ST, SMIA 1.0 Functional Specification, 2004.

[14] MIPI Alliance, Inc., "Camera Interface Specifications," 2014. [Online]. Available: http://mipi.org/specifications/camera-interface. [Accessed 11 June 2014].

[15] MIPI Alliance, Inc., "MIPI Alliance Specification for Camera Serial Interface 2 (CSI-2)," 2009. [Online]. Available: http://electronix.ru/forum/index.php?act=Attach&type=post&id=67362. [Accessed 2 June 2014].

[16] International Organization for Standardization, ISO 15739:2013 Photography -- Electronic still-picture imaging -- Noise measurements, ISO/TC 42, 2013.

[17] Image Engineering, "TE270X," Image Engineering, 2017. [Online]. Available: https://www.image-engineering.de/products/charts/all/405-te270x. [Accessed 20 11 2017].

[18] Image Engineering, "Resolution measurement for cameras with up to 180 megapixels," Image Engineering, 2014. [Online]. Available: http://www.image-engineering.de/component/content/article/70-whats-new/news/671-resolution-measurement-for-cameras-with-up-to-180-megapixels-image-engineering-s-testcharts-now-even-more-efficient. [Accessed 15 09 2014].

[19] Image Engineering, "TE268," Image Engineering, 2017. [Online]. Available: https://www.image-engineering.de/products/charts/all/584-te268x. [Accessed 21 11 2017].

[20] Imatest, "Sinusoidal Siemens Star Target - imatest," 11 11 2015. [Online]. Available: http://www.imatest.com/2015/11/sinusoidal-siemens-star-target/. [Accessed 28 11 2017].

[21] Harvest Imaging Blog, "How to Measure Modulation Transfer Function (1)," 2014. [Online]. Available: http://harvestimaging.com/blog/?p=1294. [Accessed 28 11 2017].

[22] National Instruments Corporation, "Sensor Terminology," 2017. [Online]. Available: http://www.ni.com/white-paper/14860/en/. [Accessed 1 11 2017].

[23] X-Rite, "ColorChecker Classic," X-Rite, 2017. [Online]. Available: http://www.xrite.com/categories/calibration-profiling/colorchecker-classic. [Accessed 28 11 2017].

[24] J. Mikkonen, Verification of camera module conducted immunity, Tampere: TUT, 2013.

[25] Atra Vision Inc., "Scooby2 Platform for Image Sensor Analysis," Atra Vision Inc., 2011. [Online]. Available: http://atravision.com/scooby/scooby2.php. [Accessed 8 August 2014].

[26] Image Engineering, "LE7," 2017. [Online]. Available: https://www.image-engineering.de/products/equipment/illumination-devices/378-le7. [Accessed 24 10 2017].

[27] Direct Industry, "Temperature test chamber / compact - VT minis series - Vötsch Industrietechnik," 2017. [Online]. Available: http://www.directindustry.com/prod/voetsch-industrietechnik/product-16219-424280.html. [Accessed 24 10 2017].

[28] H. G. R. Tan, ”Creating a Bluetooth ECG Monitoring System Based on NI LabVIEW,” UCSI University, [Online]. Available: http://sine.ni.com/cs/app/doc/p/id/cs-14830. [Haettu 1 10 2015].