OULUN AMMATTIKORKEAKOULU

Teemu Luukkonen

SOFTWARE DEVELOPMENT FOR A RASPBERRY PI RF SWITCH CONTROLLER

SOFTWARE DEVELOPMENT FOR A RASPBERRY PI RF SWITCH CONTROLLER

Teemu Luukkonen
Bachelor's Thesis
Spring 2020
Information Technology
Oulu University of Applied Sciences

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, laite- ja tuotesuunnittelu

Tekijä: Teemu Luukkonen
Opinnäytetyön nimi: Ohjelmiston kehitys Raspberry Pillä ohjattavalle RF-signaalien ohjausyksikölle
Työn ohjaaja: Kari Jyrkkä
Työn valmistumislukukausi ja -vuosi: Kevät 2020 Sivumäärä: 27

Opinnäytetyön aiheena oli kehittää ohjelmisto radiosignaalien ohjausyksikköprototyyppiä varten. Ohjausyksikön laitteisto on yrityksen sisäinen ratkaisu. Työn tilasi Nokia Networks Oyj.

Ohjausyksikkö on tarkoitettu käytettäväksi 5G-tukiasematuotteiden testauksessa. Yhdessä ohjelmisto ja ohjausyksikkö mahdollistavat usean lähetin-vastaanotinantennin yhtäaikaisen testauksen, sekä testattavana olevan tuotteen radiosignaalien ohjauksen mittalaitteille. Ohjelmistokokonaisuus koostuu PC:llä käytössä olevasta ajurista, joka kommunikoi Raspberry Pille implementoidun serverin kanssa. PC:lle toteutettiin myös graafinen käyttöliittymä, jolla ajuria hyödyntäen pystytään ohjaamaan ohjausyksikköä manuaalisesti.

Tuloksena saatiin toimiva ohjelmistoratkaisu, jolla pystytään ohjamaan RF-ohjausyksikköprototyypillä olevia RF-kytkimiä graafista käyttöliittymää operoimalla, kuten myös erillistä testausohjelmistoa hyödyntämällä.

Asiasanat: RF switch, 5G, DUT, Raspberry Pi

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Option of Device and Product Design

Author: Teemu Luukkonen
Title of the thesis: Software Development for a Raspberry Pi RF Switch Controller
Supervisor: Kari Jyrkkä
Term and year of completion: Spring 2020 Number of pages: 27

The objective of the thesis was to develop software for the RF switch unit prototype, including server, driver and graphical user interface implementation. The hardware is an in-house RF signal switching solution. The work was commissioned by Nokia Networks Oyj.

The RF switch unit is designed to be used in a 5G base station testing environment. The unit is capable of routing simultaneously multiple RF signal paths between devices under test and measurement instruments. Raspberry Pi acts as a server and it communicates with the driver implemented on the PC. The graphical user interface utilizes the driver to manually control the RF switch unit.

The result of the thesis project is a working software solution for the RF switch unit prototype. The RF switch unit can be controlled using both software and the graphical user interface.

Keywords: RF switch, 5G, DUT, Raspberry Pi

PREFACE

I started working as a trainee at Nokia in March 2019. During the summer 2019, I had a great opportunity to work in an interesting project. It was natural to also finish my Bachelor's degree at Nokia.

I would like to thank Antti Puolitaival and Jani Arhipoff for this opportunity and for offering a few potential topics for the thesis work, as well as Kari Jyrkkä and Kaija Posio for providing feedback during the thesis.

29.4.2020
Teemu Luukkonen

CONTENTS

ABBREVIATIONS

DLL	Dynamic Link Library
DUT	Device Under Test
GPIO	General-purpose input/output
GUI	Graphical user interface
IDE	Integrated development environment
I/O	Input / Output
I2C	Inter-Integrated Circuit
JSON	JavaScript Object Notation
REST	Representational State Transfer
RF	Radio Frequency
TAP	Test Automation Platform
TRX	Transceiver
WPF	Windows Presentation Foundation

1 INTRODUCTION

As telecommunication technologies evolve and increase in complexity, requirements in production testing change. More complex 5G radio solutions make testing more time consuming and require new kind of testing equipment. Production test software and the platforms need to be improved to make the testing less time consuming and affordable. For example, when comparing older base stations with 2, 4 or 8 radio transceivers (TRX) and the newer base stations with 32 or 64 TRXs, a test time for the base station transceivers can be multiplied. This is a problem especially when you do not want to compromise on the test coverage. The 5G technology brings new kinds of requirements for testing and new efficient solutions are crucial to keep the testing costs down.

The objective of this thesis work was to develop software for the RF switch unit prototype. The work included the development of software driver for a PC, a graphical user interface (GUI) and server implementation for Raspberry Pi. The RF switch hardware design is not in the scope of this thesis work.

Nokia is a global information technology company and it designs hardware and software solutions for the telecommunication infrastructure. Nokia has over 100,000 employees working all around the world. Today, Nokia’s focus is on the implementation and development of the 5G network solutions. Together software and RF switch unit enable to simultaneously route multiple RF signals from DUT to various measurement instruments and vice versa. The purpose is to develop an extensive support for 5G radio testing, while reducing costs and the need for expensive third-party instruments.

2 SYSTEM OVERVIEW

In this chapter the building blocks and requirements of the RF switch system are described. A more detailed explanation of the software components is in the following chapters.

2.1 System architecture

The RF switch system consists of three main components: a client, server and the RF switch unit. The PC driver and the GUI are the client component of the system. Raspberry Pi is the server component and it acts as a broker between the PC and the third component RF switch unit. The server implements a simple web API to listen for requests received from the PC driver. Figure 1 shows a diagram of the RF switch unit system.

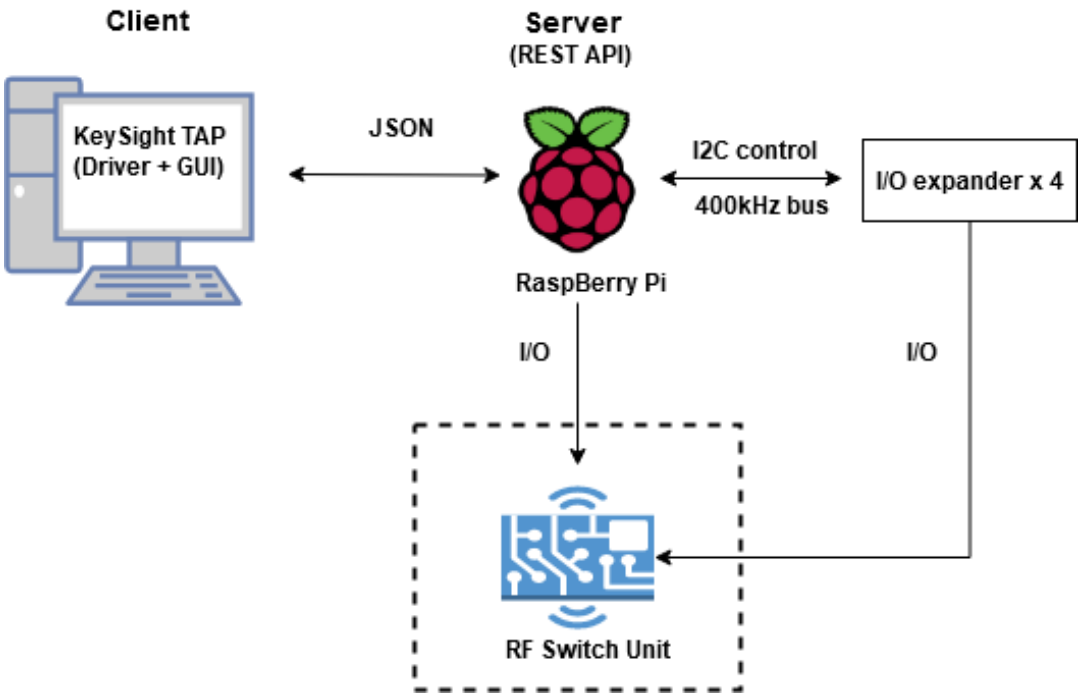

FIGURE 1. RF Switch Unit System Diagram

2.2 System requirements

The system is required to be compatible on the existing test platforms and the integration with the test platforms need to be straightforward. The Raspberry Pi server needed to be a generic solution, so it is not dependent on any platform. The Raspberry Pi server should only receive commands from a client and be able to control the I2C bus and I/O ports. To keep the server functionality as simple as possible, all the logic must be implemented to the PC driver.

2.3 Tools

A PC driver and a graphical user interface (GUI) were developed using the C# programming language with Microsoft Visual Studio 2017. Both the driver and GUI are developed as extension plugins for the Keysight Test Automation Platform (TAP).

The server was developed using Python with JetBrains PyCharm IDE and the Python server was installed on the Raspberry Pi Model 3 B+.

2.4 Keysight Test Automation Platform

The Keysight Test Automation Platform is a software platform for test automation development. TAP can be used as an extensible test sequencer platform and it allows to execute and develop automated test sequences. TAP allows developers to build a custom instrument and DUT plugins, test steps and other custom tools as extension for its base architecture. (1.)

TAP is a core part of this thesis work because the driver and the graphical user interface were built on top of it. The custom plugins are compiled as DLLs and TAP automatically loads the plugins if they are located in the same execution path.

2.5 RF switch

The RF switch unit hardware consists of a several separate RF switches that are mounted together to create a custom switch matrix. The RF switch unit enables routing radio frequency signals between multiple inputs and outputs. The RF

switches are controlled by a software and they form the connections between measurement instruments and other signal conditioners. Additionally, the unit can be used to directly route RF signals from the DUTs transmitter to another DUT or back to its own receiver. Figure 2 shows an example of RF switch with 4 inputs/outputs. The system includes a few different types of RF switches with different number of possible signal paths to be routed.

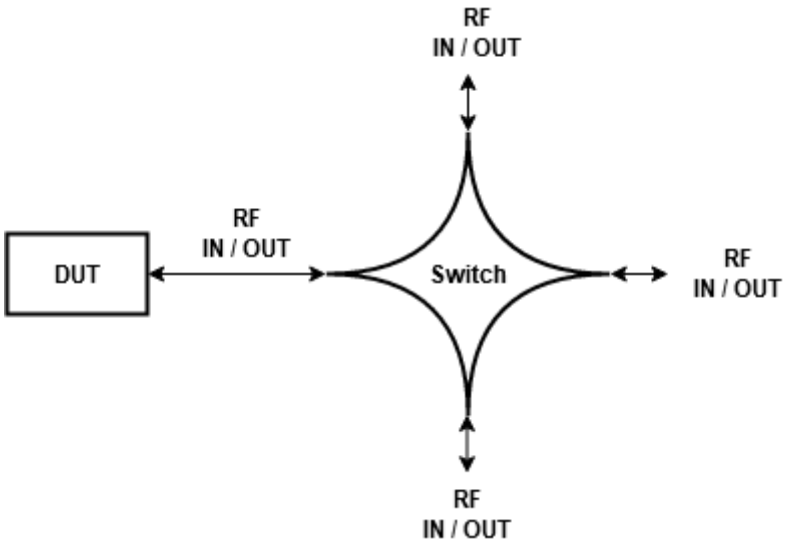

FIGURE 2. RF switch

2.6 Raspberry Pi

The RF switch system required an external controller with I2C and GPIO interfaces. Since a normal PC does not offer the said interfaces, the solution was to use the single-board computer Raspberry Pi. As a default, Raspberry Pi runs the Linux operating system with built-in I2C and GPIO interfaces. The server implementation can be done with Raspberry Pi as easily as with the normal computer. All the required interfaces can be controlled without any external components. The Raspberry Pi is already in use on some of the internal test platforms, therefore it does not bring any extra costs.

3 RF SWITCH DRIVER IMPLEMENTATION

In this chapter, the development process and issues that were encountered during the development are described. Core functionalities, technologies and components are explained.

3.1 Driver development process

The individual RF switches of the RF switch unit are controlled by I/O signals. Since Raspberry Pi has a limited amount of I/Os, a separate I/O extension board was necessary. A co-worker, who was responsible for the hardware, designed the I/O extension board, which can be directly inserted on top of Raspberry Pi. Before starting the software implementation, it was necessary to study the extension board hardware specification. The extension board included four I/O expander chips, which each provide 16 I/O signals.

After the familiarization with the hardware, it was possible to start the driver software design. Since the driver is going to be part of the existing test environment, the driver was developed as an instrument plugin for the Keysight TAP. In this way the driver can be easily used in the test automation development. The driver needs to manage the signal routing logic without a physical connection to the RF switch unit, so the system required a communication protocol between the driver and the Raspberry Pi server. It was decided that the communication between the PC and Raspberry Pi is done with JSON requests over HTTP. There was also discussion of using a plain TCP socket which can be less heavy solution. The main reason of using HTTP was the fact that the JSON was widely used in the other internal software solutions and probably in the future cloud environments. Therefore, it was useful to learn using HTTP and JSON but it also makes the transmitted data easily readable.

3.2 Driver overview

The driver handles the signal routing logic of the system. The driver receives routing commands from an external test step or from a graphical user interface. Each routing command represents the state of a single RF switch. A single RF

measurement signal path can be routed through one or more RF switches. The driver converts the received routing commands to I2C and GPIO commands, which are then sent to the Raspberry Pi server (FIGURE 3.).

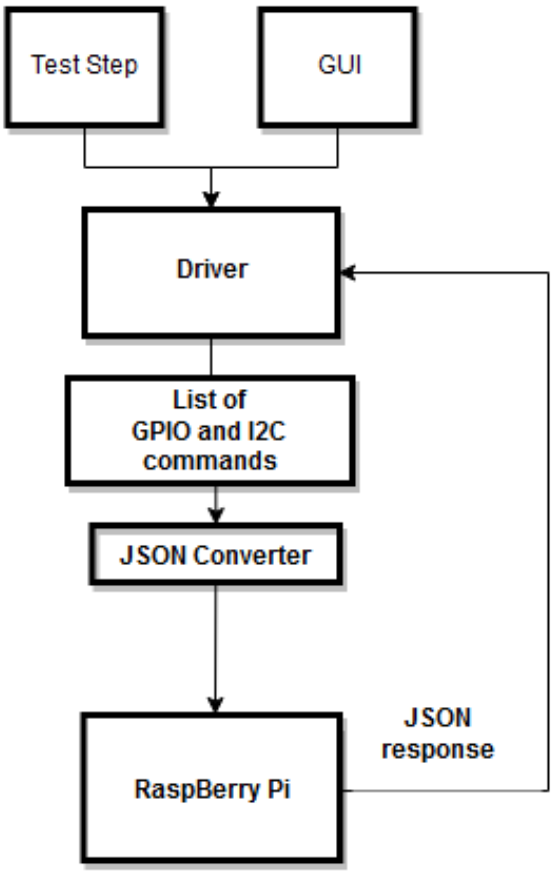

FIGURE 3. Driver functionality overview

3.3 I2C bus

The Inter-Integrated Circuit is serial communication protocol between a master and a slave device. It is an effective two-wire interface for connecting multiple electronic devices together. The I2C hardware controller is pre-integrated in most of the microcontrollers.

The I2C interface uses SCL (serial clock) and SDA (serial data) lines for the communication. In simplicity, the master device generates a start condition to reserve the bus. Then, the address of the slave device is requested and data is read or

written to the register. After data transmission is finished, the master generates the stop condition and releases the bus to be used by another device. (2.)

In the thesis work Raspberry Pi is considered as the master device on the I2C bus, while I/O expander chips are the slave devices.

3.4 GPIO

A general-purpose input/output is a standard interface used to connect microcontrollers to other electronic components. GPIO is a digital signal pin that does not have a pre-defined function. Its functionality can be controlled by software. These pins can be configured to be used to either receive inputs or to provide outputs to the other components. (3.)

Raspberry Pi delivers a set of GPIO pins. For example, a GPIO pin can be used to detect a push of a button or to control external switches, as in this thesis work.

3.5 Routing commands

The I/O expander board contains total of four I/O expander devices. They each have two 8-bit output registers, which are connected to RF switch control inputs. Registers are divided into three, four and five -bit groups. Each group presents input pins of a single RF switch. RF switch control inputs are connected to the expander I/Os or directly to the Raspberry Pi I/Os.

A route for the switch is selected with different bit combinations. In figure 4 there is an example of switch with three control inputs and some of the possible states.

Route 1	Route 2	Route 3
001	010	011

FIGURE 4. RF switch route combinations

To set a route for a single switch, two separate commands might be needed because bits of each register are shared with multiple RF switches. This leads to situations where parts of the control bits are located in another register address.

Figure 5, where sections marked in blue represent a RF switch, shows that the control bits are divided into two separate register addresses.

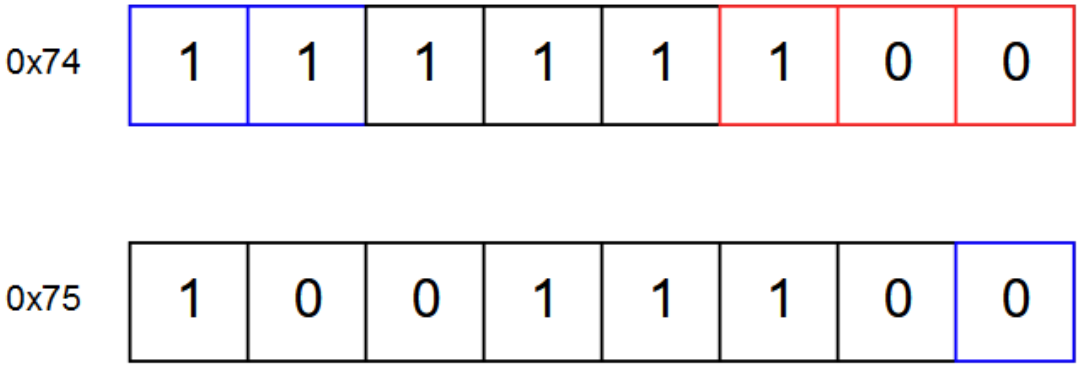

FIGURE 5. I/O expander registers

There is also a case where one or more of the RF switch control inputs are directly connected to the Raspberry Pi GPIO port. This means that these switches need both I2C and GPIO commands to set the desired route. Figure 6 illustrates the RF switch with a 5-bit control input where one of the bits is controlled directly from the Raspberry Pi I/O port.

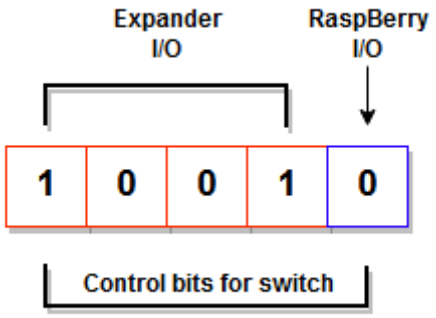

FIGURE 6. Switch with 5-bit control input

3.6 Driver functionality

The driver requests a target register value from the server. The server returns a JSON response, which contains the register address and the current register value. The register value is then modified with a new value. Since registers are

shared with more than one switch, it is necessary to ensure that the current state of the other switches is not changed.

The register needs to be masked with the route command in the correct position. In code, every switch object holds the required register information:

- Device address
- Register address
- Offset from the least significant bit
- Number of bits to write to register(s)

Figure 7 shows the method that takes needed parameters as inputs and modifies the register.

```
public string ModifyRegisterValue(byte newVal, byte currentRegValue, byte maskBitsCount, byte offset)
{
    byte maskBits = 0;

    for (var i = 0; i < maskBitsCount; i++)
        maskBits = (byte)((maskBits << 1) | 1);

    var mask = maskBits << offset;
    var output = (currentRegValue & ~mask) | (newVal << offset);
    if (output > 255) throw new Exception(message: "Output value greater than byte");

    return $"0x{output:X}";
}
```

FIGURE 7. Register value modification method

3.7 JSON Converter

Before sending the I2C and GPIO control commands to the server, the commands need to be converted to a format that the server can process. The driver aggregates a list of commands, which are then converted to a JSON format. In figure 8 there is an example of class constructor, which takes the I2C device, address, register and data as parameters to initialize the object. Other control commands are constructed in the same style.

```
public WriteI2C(string address, string register, string data)
{
    this.address = address;
    this.register = register;
    this.data = data;
}
```

FIGURE 8. JSON object constructor

A list of objects is then serialized to a JSON format, using the Json.NET framework (FIGURE 9.).

```
public static string JsonConverter(List<object> payload) =>
    JsonConvert.SerializeObject(payload, Formatting.Indented);
```

FIGURE 9. JSON converter method

In figure 10 there is an example of the human readable JSON request which is then sent to the server.


```
[
  {
    "address": "0x74",
    "register": "0x07",
    "value": "0xF1"
  },
  {
    "address": "0x75",
    "register": "0x06",
    "value": "0x05"
  }
]
```


FIGURE 10. JSON request body

4 GRAPHICAL USER INTERFACE

The graphical user interface (GUI) was developed to manually test the prototype of the RF switch unit. The objective was to develop a simple and easy to use graphical user interface.

The GUI was developed with the Windows Presentation Foundation (WPF) UI framework. The GUI was not developed as a standalone desktop application, rather it was developed on top of the Keysight Test Automation Platform (TAP) as a dockable panel. Implementing the GUI as a TAP plugin makes it easy to be deployed on the different tester platforms which are using the TAP. In this way the GUI is conveniently embedded to the existing test platform. Development of the GUI was a good chance to implement and learn the basics of the Model-View-ViewModel (MVVM) pattern, although it does not bring major benefits in such a simple application.

4.1 Windows Presentation Foundation

The Windows Presentation foundation (WPF) is a UI development framework developed by Microsoft. The WPF framework library includes various GUI elements, such as textboxes, grids, labels and other basic GUI elements. The WPF framework is part of the .NET and it enables to develop applications using a declarative style. Normally, the Extensible Application Markup Language (XAML) is used to develop the appearance of the application and code-behind to implement the behavior. (4.)

4.2 MVVM

The Model-View-ViewModel pattern is a development pattern which is used to separate the user interface from the logic of the software. The model, the view and the view model are the key components in the MVVM pattern. The separation of the view and the logic makes the software easier to be deployed on another platform or operating system. Implementing the MVVM pattern enables the user interface developers to work independently with the view, while other developers can work on the model and view model logic. This logic separation also makes

the software easier to be unit tested because the logic is not tightly coupled to the view objects. There are also other design patterns that are used to develop WPF applications, but they are not discussed in the document. (5.) Figure 11 illustrates the relationship between components in MVVM pattern.

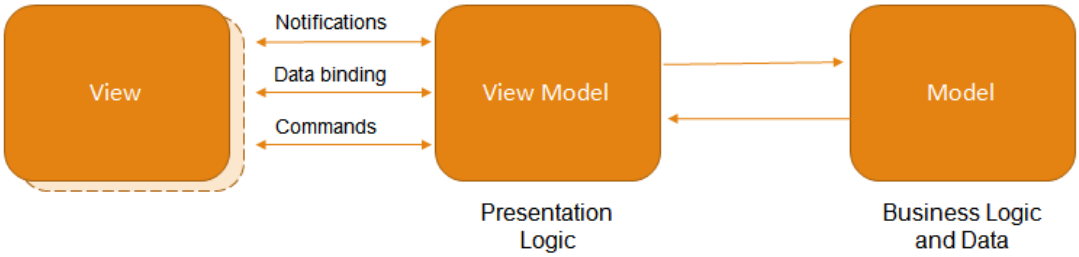

FIGURE 11. Relationship between MVVM components (6.)

4.3 GUI development

It was known that the GUI can be developed directly on top of the Keysight TAP but it required some time for the research to find out how it can be done. Keysight did not offer comprehensive instructions for the GUI panel development. The most useful resource was to study the single code example which comes with the TAP installation package.

After all, the GUI development was quite straightforward. There was an interface which needed to be implemented to integrate the custom GUI with the TAP. Developing the GUI on top of the TAP did not differ much from the regular desktop application development. In this manner it was possible to use other TAP interfaces conveniently, such as the driver interface explained in chapter 3.

4.4 GUI features

The graphical user interface seen in the figure 12 is the software tool for the manual control. Each RF switch can be separately set to the wanted state on click of a button. The user can quickly route a desired signal path from the known input to an output. The GUI software utilizes the driver described in chapter 3.

Row labels represent the name of a single RF switch. Column labels represent the names of the different routes. All the available routes of the RF switch are

color coded in the hardware specification. The same color coding was applied to the GUI. Therefore, it is easy to select the correct route. The switches without color coding in the lower section of the GUI panel are a different type of RF switches. Switches of this type have the same purpose but are specified in a separate table.

Only the necessary features were implemented to the GUI, since the GUI is only useful during the development of the RF switch unit prototype. The functionalities of the graphical user interface are:

- Button to check the server status
- Button to open the connection resource between the server and the PC
- Button to reset the GUI buttons and to reset the RF switches to a default state
- Button to close the connection resource between the server
- Buttons to select a route of the individual RF switches

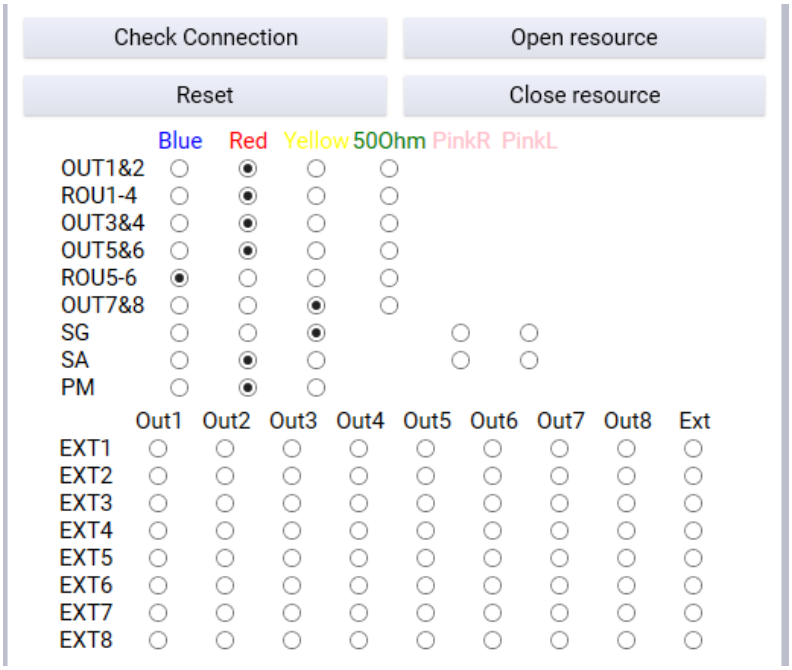

FIGURE 12. RF switch unit GUI

5 SERVER

In this chapter server components and functionalities are described.

5.1 Server background

The server was designed as a general solution for Raspberry Pi I2C and GPIO interfaces. It means that the Raspberry Pi server software is not tightly coupled with the RF switch driver, rather it can also be used in other development projects. The server is responsible for handling the incoming JSON serialized commands. It can only handle JSON requests but in the future it can be modified to accept requests also in different types of data formats.

Raspberry Pi is equipped with the I/O expander extension board where the I2C controllable I/O expander devices are mounted. The default I2C interface of Raspberry Pi is used to communicate with the I/O expanders.

The server API is implemented with the Flask web framework. Python was selected as a programming language because it is well supported also in Linux environments and it offers many open source frameworks and tools.

5.2 Flask framework

Flask is a Python micro web framework. It offers tools, technologies and libraries for fast and easy deployment of web services, applications and APIs. (7.)

Flask was used in this project because it is a lightweight framework, easily scalable and it makes the server deployment fast.

5.3 REST APIs

The Application Programming Interface (API) is an interface which allows applications and services to communicate with each other. For example, a server receives data from the application, processes the received data and sends a response back in a readable format. (8.)

The Representational State Transfer (REST) is one of the most popular types of APIs. REST takes advantage of existing communication protocols. It can be used with many different protocols, typically HTTP for web APIs. A REST API defines a set of functions so that users can perform requests and receive responses via the HTTP protocol. (8.)

5.4 Server endpoints

The server is implemented with an API that uses JSON over HTTP to communicate with the client. The server provides a set of endpoints for different actions that the client can access. In figure 13 there is a code example of endpoint which is mapped to function so that it reads the requested register and sends a success or an error response back to the client. The Flask framework enables to easily map URL paths to the functions with the route() decorator. The server has a separate function and endpoint for the different actions.

```
@app.route('/i2c_read', methods=['POST'])
def read_i2c():
```

FIGURE 13. Example server endpoint

5.5 Server functionality

The server listens to commands and accepts the requests only in a JSON format. The server deserializes the received JSON requests and uses that data to form the I2C and the GPIO commands. The commands are executed using the SMBus and RPi.GPIO libraries. A status message of the read or write operation is returned to the client.

5.6 Error handling

An exception or an HTTP error is handled in a desired way. Figure 14 shows an example of an HTTP error function which returns the error code and message to the client so that the client can handle the error correctly. The example error message seen in figure 14 could be improved by returning a more detailed error message. The Flask framework errorHandler() decorator is used to register specific HTTP error status code to a function. Other than the HTTP exception messages

are also returned to the client for further handling. When an I2C or a GPIO command execution fails during a write or read operation, the client gets a custom I2C or GPIO error message.

```
@app.errorhandler(400)  # JSON response for user.
def not_found(error):
    return jsonify(ERROR="Bad request, 400")
```

FIGURE 14. Bad request handler

6 CONCLUSION

The objective of the thesis was to develop a software driver, server and graphical user interface for the RF switch unit prototype. The system is used to route RF signals between DUTs and measurement instruments. The RF switch system aims to cut down the costs of the new test platforms and decrease the need for the expensive third-party solutions. The RF switch unit is planned to be used in the Nokia's test engineering development and production line environment.

Several different tools and technologies were used in the project. Some of the used technologies were already familiar for the author, so it was possible to utilize the previous experience. However, a lot of new skills were learned, such as the WPF framework and the MVVM pattern used in the graphical user interface development. Overall, the author's programming skills have improved during the implementation of a whole software architecture, including the server, the graphical user interface and the driver. The project was a good way to learn the plugin development for the Keysight TAP, since it will be used in the future job assignments.

At the beginning of the thesis project it was necessary to examine the RF switch unit hardware specification. The hardware specification was almost already done before the start of the project and the first RF switch prototype was available quickly. The software implementation was completed in time and met the most important predefined requirements. The driver/server combination was utilizable by the Keysight Test Automation platform. During the development process, the graphical user interface was useful for the hardware testing and debugging.

The software can be later improved and developed further. The feature that was discussed also at the beginning of the thesis associates with the improved routing logic. The discussed new feature was an algorithm for the driver that would dynamically find the wanted RF signal path. With this feature the user would only need to know the signal input point and the output point. The algorithm implemented would find the shortest route and form a list of the routing commands. Currently, the possible routes need to be predefined in an external file or switches

need to be individually controlled with the GUI. If the RF switch unit hardware is modified and new switches are added, the algorithm would make the software maintenance easier. If more time had been spent, the software could have been also improved by implementing the unit tests for the three software components.

REFERENCES

1. Keysight Test Automation Platform. Date Of retrieval 20.12.2019. https://www.keysight.com/upload/cmc_upload/All/TapDeveloperGuide.pdf

2. I2C bus, interface and protocol. Date of retrieval 22.12.2019. https://i2c.info

3. GPIO Electrical Specifications. Date of retrieval 6.1.2020. http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/gpio-pin-electrical-specifications

4. WPF overview. Date of retrieval 21.12.2019. https://docs.microsoft.com/en-us/dotnet/framework/wpf/introduction-to-wpf

5. The MVVM Pattern 2012. Date of retrieval 10.12.2019. https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)?redirectedfrom=MSDN

6. MVVM Framework. Date of retrieval 10.12.2019. https://documentation.devexpress.com/WPF/15112/MVVM-Framework

7. Full Stack Python. Date of retrieval 6.1 2020. https://www.fullstackpython.com/flask.html

8. MuleSoft. What is an API. Date of retrieval 6.1.2020. https://www.mulesoft.com/resources/api/what-is-an-api