Paavo Siitonen

Ultra Low Cost Test Platform

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme in Electrical and Automation Engineering

Bachelor's Thesis

31 May 2022

Abstract

Author:	Paavo Siitonen
Title:	Ultra Low Cost Test Platform
Number of Pages:	35 pages + 3 appendices
Date:	31 May 2022
Degree:	Bachelor of Engineering
Degree Programme:	Degree Programme in Electrical and Automation Engineering
Professional Major:	Electronics
Supervisors:	Heikki Valmu, Senior Lecturer

The topic of this work was to design and implement an inexpensive test system for production testing of printed circuit boards. The system is designed to be manufactured simply and easily using commercially off-the-shelf products. Ready-made expansion boards and component are utilized to make the manufacturing of the system easy, fast and as cheap as possible.

The work focused on the development of electronic hardware for the system. Software development and mechanical design were only partially within the scope of this work. If the system enters production phase, an external provider may be used to design the final version of the software. However, to verify the functionality, a test software was developed, intended to be as close to the final version as possible.

The work resulted in a test system based on Raspberry Pi that can be used to test simple circuit boards that can have a few dozen test points. The system was manufactured with a purpose-built Python program at the Lohja plant of Enics Finland.

Keywords: In-circuit testing, Functional testing, Test fixture, Raspberry Pi

Tiivistelmä

Tekijä:	Paavo Siitonen
Opinnäytetyön nimi:	Ultra Low Cost Test Platform
Sivumäärä:	35 sivua + 3 liitettä
Päivämäärä:	31.5.2022
Tutkinto-ohjelma:	Sähkö- ja automaatiotekniikka
Pääaine:	Elektroniikka
Valvojat:	Yliopettaja Heikki Valmu
	Janne Kuusivaara, Test Development Manager at Enics

Työn aiheena oli suunnitella ja toteuttaa halpa testijärjestelmä painettujen piirilevyjen tuotantotestaukseen. Järjestelmä on tarkoitettu valmistettavaksi yksinkertaisesti ja helposti käyttäen markkinoilta valmiina saatavia tuotteita. Valmiiden komponenttien käytöllä on tarkoitus tehdä järjestelmän valmistamisesta niin helppoa, halpaa ja nopeaa kuin mahdollista.

Työ keskittyi elektronisen laitteiston kehittämiseen järjestelmään. Testausohjelmiston kehittäminen ja mekaaninen suunnittelu kuului vain osittain työn aihealueeseen. Jos järjestelmä etenee tuotantovaiheeseen, käytetään mahdollisesti ulkopuolista tahoa ohjelmiston suunnitteluun. Järjestelmän toiminnallisuuden toteamiseksi kehitettiin kuitenkin ohjelmisto, joka mahdollisimman pitkälle vastaa lopullista versiota.

Työn tuloksena valmistui Raspberry Pi:lle perustuva testausjärjestelmä, jolla voidaan testata yksinkertaisia piirilevyjä, joissa voi olla joitain kymmeniä testipisteitä. Järjestelmä valmistettiin ja testattiin tarkoitukseen kehitetyllä Python-ohjelmalla Enics Finlandin tehtaalla.

Asiasanat: elektroniikan testaus, testausjärjestelmä, Raspberry Pi, piirilevy

Contents

List of Abbreviations

AOI:	Automated Optical Inspection
ATE:	Automated Test Equipment
AXI:	Automated X-ray Inspection
BGA:	Ball Grid Array
COTS:	Commercially off the shelf.
DUT:	Device Under Test
FCT:	Functional testing
GPIO:	General-purpose input/output
HAT:	Hardware Attached on Top.
ICT:	In-circuit testing.
MES:	Manufacturing Execution System
PCB:	Printed Circuit Board
SCPI:	Standard Commands for Programmable Instruments
SPI:	Serial Peripheral Interface
UUT:	Unit Under Test

1 Introduction

Testing of electronics typically involves testing printed circuit boards (PCB). PCBs are expected to function according to their design parameters with no errors or failures. Poor designing or bad quality can cause a PCB to malfunction or stop working which can be costly if it happens with a finished product. Therefore PCB testing has to be done as a part of the manufacturing process, so that possible defects can be detected as early as possible.

The topic of this thesis is developing an ultra low cost test platform for testing PCBs of electronic devices. The goal was to develop a system that is very cheap and can be easily put together. To achieve this aim, first the components for the platform were chosen. In particular, the aim was to find affordable, ready-made parts from the market. Since the platform would be built on Raspberry Pi, especial attention was put on finding suitable Hardware Attached on Top (HAT). HATs are expansion boards that can be connected to Raspberry Pi by attaching them on top of the set of 40 GPIO pins included in the Pi.

2 PCB Testing Methods

Inspection and testing ensure high quality in the manufacturing of PCBs and electronic devices [1]. Each PCB must be inspected and tested based on its specific design and performance specifications. PCBs are often complex with hundreds of components and thousands of solder joints. Because of this, the manufacturers of products that use PCBs have instituted inspection and testing procedures that ensure the quality of the products. These procedures discover faulty boards for rework and create a feedback process that ensures continuous improvement.

There are three main methods for testing electronics. which are visual inspection, in-circuit testing (ICT) and functional testing (FCT)[2]. These will be discussed below in chronological order. The most common inspection methods are automated optical inspection (AOI) and automated X-ray inspection (AXI).

2.1 Visual Inspection

2.1.1 Automated Optical Inspection

Automated optical inspection improves inspection systems by limiting human error component in the process. In an AOI system, like in figure 1, one or more cameras autonomously scan the board. The system takes images and compares them to what the board is supposed to look like.

Figure1. Automated optical inspection machine.

AOI can detect both problems that can cause the board to malfunction, like open or short circuits or missing components and quality issues like nodules, stains, scratches or skewed components. AOI systems can perform the inspection process faster and more accurately than a human inspector. The disadvantage of AOI is that it is limited to the line of sight.

2.1.2 Automated X-ray Inspection

Automated X-ray inspection uses X-rays as its source instead of visible light. In AXI systems, X-rays are transmitted through the object and a detector records an image of the X-rays. These are then either converted to visible light and imaged or detected with an X-ray sensor array. The produced image can then be compared with features that the board is expected to have. AXI is often used where AOI cannot be used; For example, in inspecting BGA

(Ball Grid Array) packages, where the connections are under the chip, out of the line of sight of AOI.

2.2 In-Circuit Testing

There are several testing methods for testing a manufactured PCB. Of these in-circuit testing and functional testing are the most common. In-circuit testing means an internal measurement of the circuit board. The PCB is in passive state, which means that no voltage is connected to the board. Testing of internal structures or workings of an application is also known as white-box testing.

ICT is based on a large number of test points, which are placed onto PCBs for testing purposes by the manufacturers [2]. In this way the components can be measured in passive state, which reduces the rework required at functional testing. Common defects that an in-circuit tester detects include solder shorts, disconnected solder joints and missing or poorly soldered components.

2.2.1 Flying Probe

The test device Figure 2 is a flying probe tester. Flying probe machine has needles that go around the circuit board and take measurements between test points. It does not need a test fixture, but it is slower than using a bed of nails tester. The advantage of flying probe testing is that it places less mechanical strain on the PCB being tested.

Figure 2. Flying probe.

2.2.2 A Bed of Nails Test Fixture

In a bed of nails text fixture a bed of nails tester is put into a fixture and takes the required measurements. A PCB is aligned and pressed against the bed-of-nails tester so that test points make a contact with spring loaded connectors. The measurements are made through these probes, which are connected to the test devices.

2.3 Functional Testing

In functional testing actual power is applied to the circuit board. FCT validates that all the functions in the circuit board work according to specifications, for example that a voltage regulator puts out the correct voltage. It makes sure that the board can be integrated into the full product that is being manufactured, and it will function as it is supposed to.

The main benefit of FCT as well as ICT is that they save time and money when circuit boards are found faulty. When the defects are discovered at this phase of

the manufacturing process, the boards can be reworked or scrapped without disassembling the whole product which can be difficult and expensive.

2.4 Boundary-Scan

Boundary scan is an in-built architecture, where tests can be performed without using external probes. Boundary scan utilizes JTAG protocol and it can simulate e.g. a voltage signal to a pin of the component without being active and without programming with the software desired by the customer. Boundary scan reduces the amount and complexity of equipment required for testing, because using it eliminates the need for external probes and expensive test fixtures. Because of this, the use of boundary scan is expected to grow in the in the future at the expense of in-circuit testing.

3 Design of an In-Circuit Test System

3.1 Basic Automated Test System

A basic automated test system described in figure 3 consists of test controller, test equipment, switching control and circuitry and the device under test (DUT)[3]. Switching control and switching circuitry are usually put inside a test fixture. The test controller is connected to the switching control, which controls the switching circuitry. The switching circuit connects the test equipment and instrumentation to the test points in the DUT.

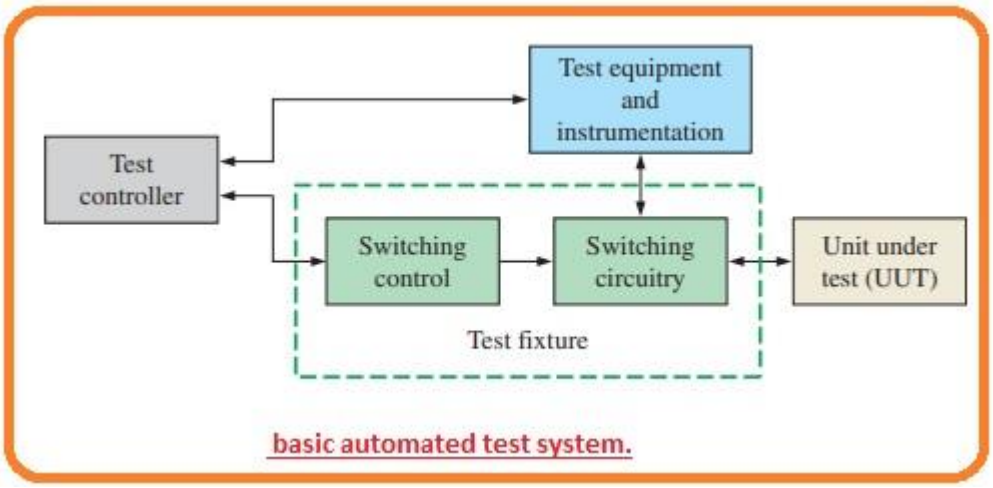

Figure 3. A basic automated test system [3].

Figure 4 is a typical in-circuit test fixture. In this platform the controller and parts of the test equipment and instrumentation as well as the switching control and circuitry are formed by Raspberry Pi and expansion boards. They will all be attached together inside the text fixture. On the outside of the test fixture are the DUT and the power source, as well as a touch display or keyboard and mouse with which to operate the system.

Figure 4. In-circuit test fixture.

3.2 Composition of an In-Circuit Test Fixture

In this work the demonstration product used does not have test points and therefore connectors and wiring will be used to connect with the DUT. However, usually in testing electronic products a bed of nails text fixture is used. Bed of nails test fixture acts as a press with which to press the circuit board against spring-loaded test probes called pogo pins. Figure 5 shows a selection of pogo pins. They get their name from their resemblance to pogo stick. Pushing down

the pin creates contact between the different parts of the probe, *plunger* and *barrel.*

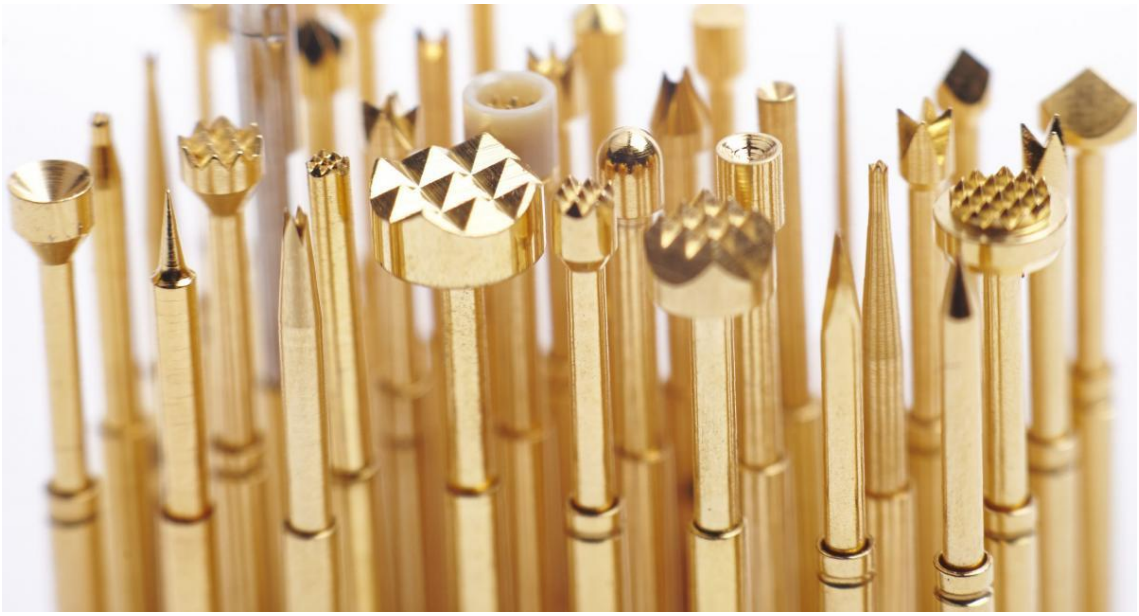

Figure 5. Pogo pins.

The fixture base, which contains the pressing mechanism, is reusable for different products. Inside the base is the adapter that connects the test devices to the product. The adapter contains three different plates, pressure plate, moving plate and probe plate. These will have to be customized for each different circuit board. The pressure plate is usually placed on the cover of the test fixture. When the cover is closed, the pins in the pressure plate push down against the circuit board that is being tested on the moving plate as can be seen in figure 6.

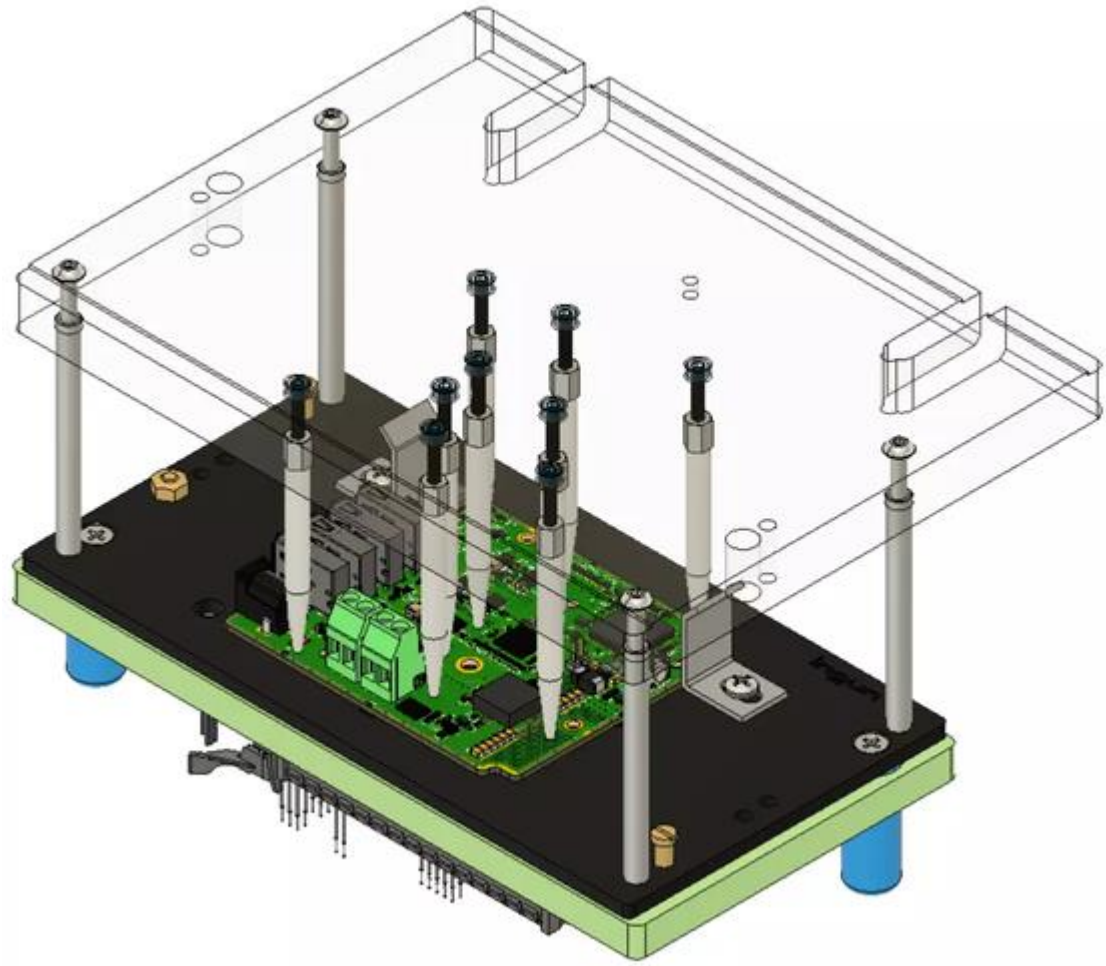

Figure 6. Pins of the pressure plate pushing against the DUT.

On the opposite side the pogo pins make a contact with test points of the DUT. Inside the pogo pins a contact is created inside when the springs are pushed down. If the product has a lot of test points, pushing down all the corresponding pogo pins requires a lot of force, often hundreds of Newtons. If a lot of force is applied to one corner of the PCB, it causes the board to bend and may cause serious damage and failures on the product. The moving plate supports the circuit board from bottom to prevent it from bending and being damaged.

The probe plate is where the pogo pins and the connectors are mounted into. It has holes machined into it such that the probes get contact with the test points in the DUT as can be seen in figure 7. Below the probe plate the pogo pins are attached into a custom PCB. This circuit board is connected into tester receiver

with signal wires and acts as an interface between the product and the instrumentation that is being used. If there is data that needs to be loaded into components of the device that is being tested, a hardware programmer is needed.

Figure 7. Pogo pins on the probe plate.

3.3 Software

There are several software solutions available for running automated test system. These range from simple code written for example in C or C++ to a graphical UI using LabVIEW. Labview is a graphical programming environment for designing test, measurement and control systems[4].

One of the most commonly used test software is TestStand by National Instruments. TestStand is high-level test executive software that allows direct call to modules that are in diverse languages[5]. It has two modes: a development mode for defining the testing processes or sequences and a run-time mode for executing the test sequences and generating reports of the tests.

4 Ultra Low Cost Test Platform Specifications

The test platform was intended be built according to initial specifications, that were provided by Enics Oy. These were divided in five parts and included requirements for processor board, for measurements and controls, for test sequences and software interfaces, for user interface and for mechanical requirements. Some of the most essential requirements are described in this chapter. The complete list of specifications for the platform can be found in appendix 1. Mechanical design was not really part of the work, so from those requirements only the need of the processor board and IO interfaces to have casing was something that had to be kept in mind.

4.1 Processor Board

The recommended device for control unit is Raspberry Pi 4. The processor board has to have at least minimum two USB 3.0 ports and 4 GPIO lines. The board must be capable of 1Gps Ethernet connection and have a wifi interface. It must also support USB 2D code scanner and have a light operating system. All of these requirements are met in Raspberry Pi 4 B, which was at the time the latest version of the device.

4.2 Measurements and Controls

Specifications for measurements and controls were some of the most important requirements for the platform, because finding appropriate measurement- and control devices was the core part of the work. Once suitable devices were found, the rest of the project was relatively straightforward. According to the specifications the platform would have to have at least three relay outputs, as well as minimum three input- and three output channels. The system should have at least one 0 V to 30 V digital voltmeter measurement channel with 2% accuracy. The power supply should be at least 30 V / 1 A, and it should be controllable through Ethernet or USB.

4.3 Test Sequences and Software Interfaces

Only some of the remaining specifications were relevant for the project, because the main focus of this thesis was electronic hardware. Once the electronic hardware part of the work was mostly finished, an attempt was made to get the software as ready as possible, even if it would not be used in the final version of the platform. Because of this, the specifications for the software became more important as time progressed, but there was not anywhere near enough time to get the software in condition where it would fulfill all of the requirements.

One of the requirements for test sequence and software interface was that the test program should be implemented using a well-known language like Python, HTML and XML. The Platform should also have an interface for a J-Link programming device and support for Enics Manufacturing Execution System (MES). Local test results would have to be stored using Enics TDC xml or ATML format and the software should have revision management. Additionally the test sequence would have to support serial number reading and would have to able to communicate with the power supply unit.

4.4 User interface

The user interface specifications required that there should be three colour indication for operator with green meaning OK or pass, yellow meaning that the test program is running and red being for fail. Touch interface was also a requirement and additionally the user interface was to have a start button. It was initially thought that all of these requirements could be met by controlling the test program from a touch display by pressing a button that would change colour according to the state of the program.

Another requirement for the user interface was that the platform should have the possibility to add external display with HDMI connection. Test software should also have a debug mode and in it tell the user the test step, test result, did the

test pass or fail and the limits for the ongoing test step. Debug information would have to be real time.

5 Components

5.1 Processor Board

The test controller unit for the platform is Raspberry Pi 4 model B. Raspberry Pi is a series of small single-board computers (SBCs) developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom[6].
Raspberry Pi has everything that is needed to connect a mouse, a keyboard, a monitor and internet in it and can therefore be used like a PC. It runs on a Linux-based operating system called Raspbian.

There are three series of raspberry Pi with several generations of each of them. First there is the original Raspberry Pi which was released in 2012. Another series is Raspberry Pi zero, which was released in 2015. This is smaller in size and has less I/O and GPIO capabilities. In January 2021 another series called Raspberry Pi Pico was released, which is a very small microcontroller and based on a single microcontroller chip. The version used in this project, Raspberry Pi 4 Model B in figure 8, belongs into the original main series and it was released in June 2019.

Figure 8. Raspberry Pi 4 B.

Component shortage caused problems in acquiring the touch display. Initially PiTFT 3.5” touchscreen was to be used, but the delivery date of this was pushed further and further by the distributor, and it could not be acquired in time. The new official Raspberry Pi display, which is a 7” touchscreen display, was considered, but eventually testing the touch interface was postponed for further development after the thesis was finished. Raspberry Pi has two slots for HDMI cable so an extra display can be added.

5.2 Power Supply

According to the initial specifications the platform needs to have an Ethernet- or USB- controlled, minimum 30V/1A power supply. For this Tenma 72-2710 Bench Power Supply in figure 9 was chosen. It is USB-controlled and it costs a little over 100 € [7]. Raspberry Pi comes has its own 5.1V/3A official power supply unit, which is also used in the platform.

Figure 9. TENMA 72-2710 power supply.

5.3 Flashing

The specifications require a J-Link device to be used as a Flash programmer. The cheapest device for commercial use is J-Link Base Compact in Figure 10 at around 400 €. The demonstration product that was chosen to be tested does not have any components that need programming. Therefore the hardware programming device was not needed it the demonstration test.

Figure 10. J-Link Base Compact hardware programmer by Segger.

5.4 IO Control and Measurements

DAQC2plate by Pi-Plates was the expansion board chosen for data acquisition. DAQC2plate is an expansion board meant to be used with raspberry Pi. The board in figure 11 has 8 analog and 8 digital inputs, as well as 8 open collector outputs [8]. Up to 8 DAQC2plate boards can be stacked together to get more input- and output channels.

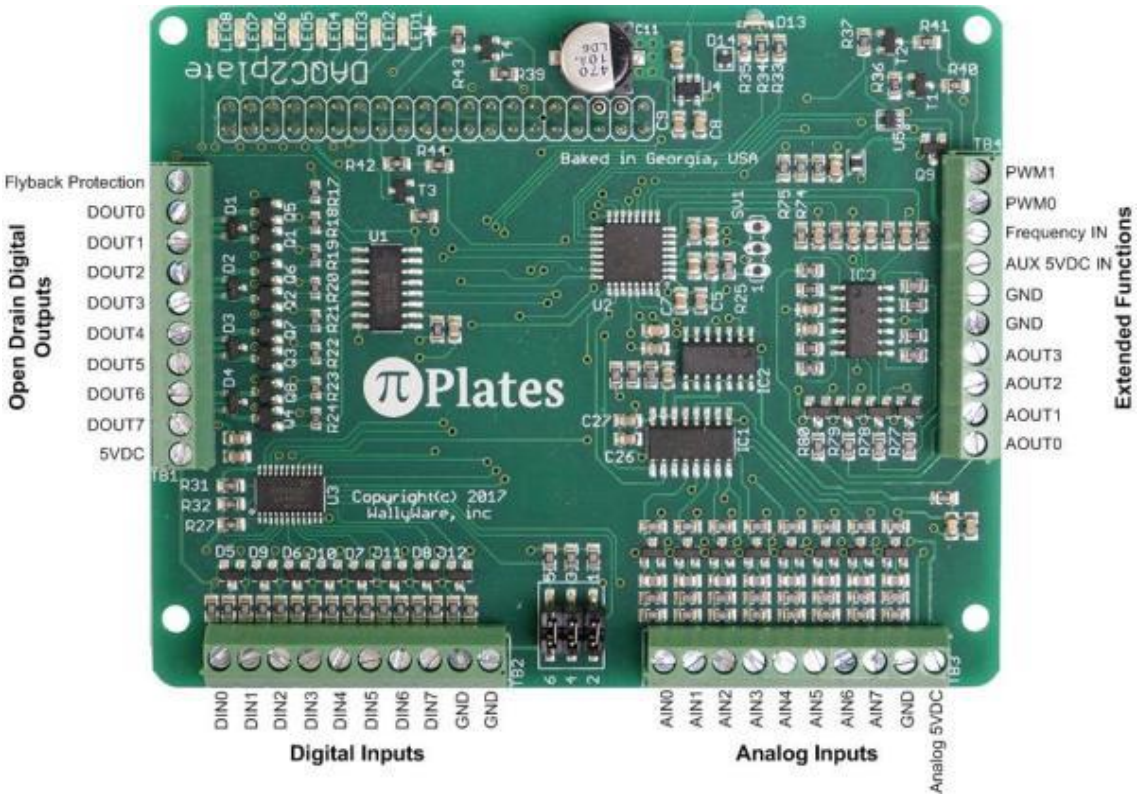

Figure 11. DAQC2plate by Pi-Plates

The measurement accuracy of analog input channels is 0.1% and they have -12 V to +12 V measurement range. This means that the 30 V measurement channel with 2% accuracy required by the specifications can be comfortably implemented using a voltage divider. A divide-down ratio of 3 would still give 0.3% accuracy, which is well within the margin of error.

5.5 Relay Outputs

The platform is required to have minimum three relay outputs. To provide these it was decided to use RELAYplate, which is also offered by Pi-Plates. The board in figure 12 has 7 relays that are rated to switch 1 A at 120 volts AC or 30 volts DC [9].

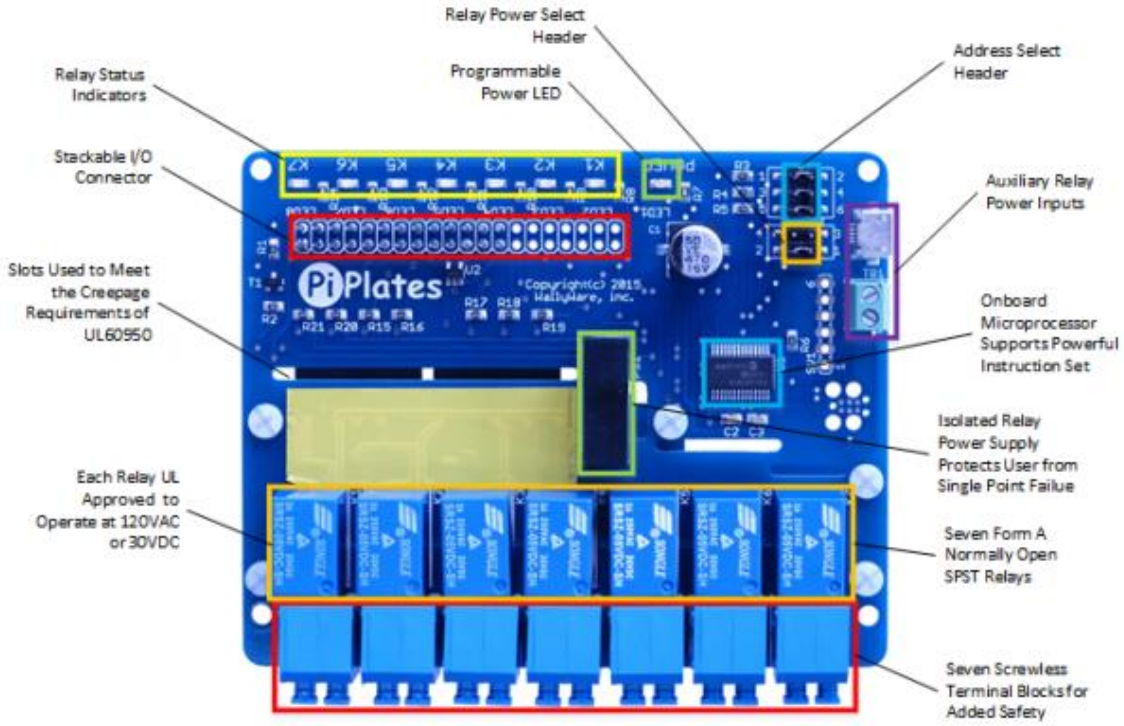

Figure 12. RELAYplate

5.6 Bar Code Scanner

The system also must support a USB 2 D bar code scanner to be able to read serial numbers. Figure 13 shows the device that was chosen. This was Opticon C37 and it was chosen because it was the cheapest scanner found in the market that was still practical to use.

Figure 13. Opticon C37 bar code scanner.

6 Implementation

Then the components were put together. Figure 14 shows the general composition of the platform in the demonstration test. The platform was built at the Lohja factory of Enics Finland. All the electronical components that were used were commercially off-the-shelf products. For mechanical design, the casing box was also ordered from a distributor, but the bottom plate and the cover were manufactured in the factory.

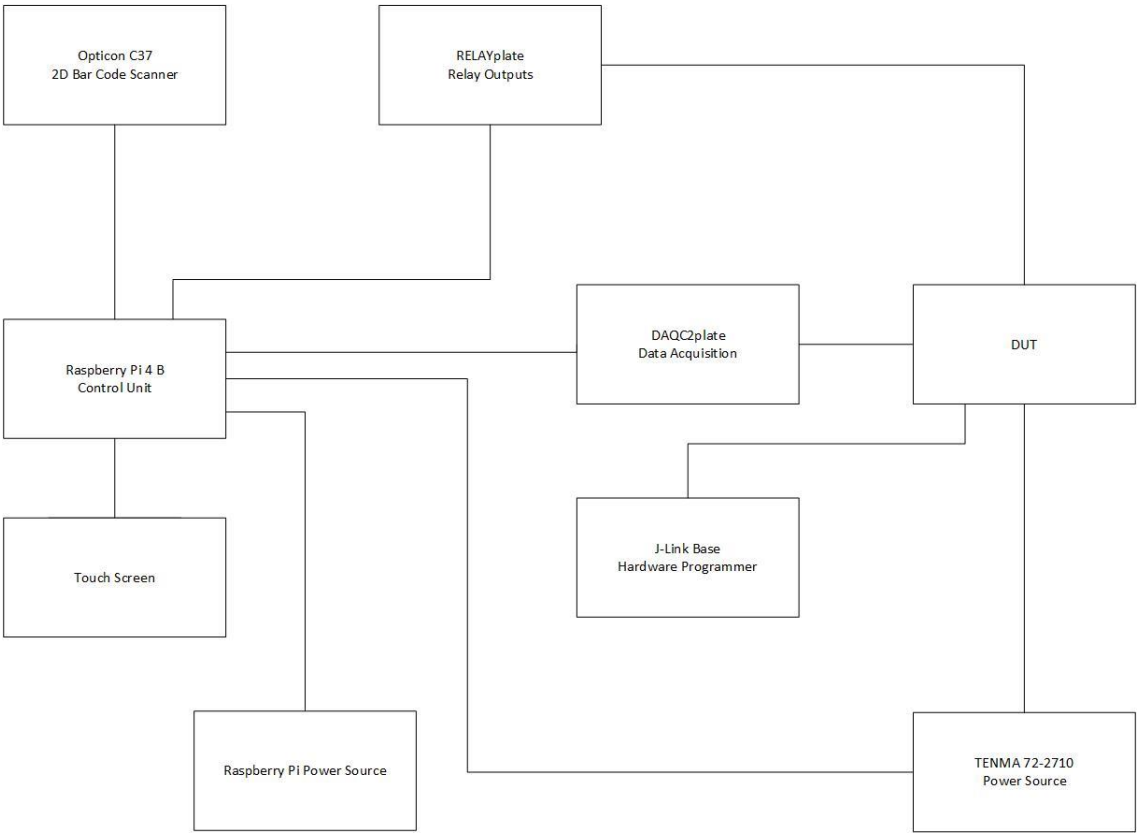

Figure 14. General composition of the platform

6.1 The Product

It was decided that for a product demonstration, a relay multiplexer board would be used as the DUT. This board, which is seen in figure 15, is designed and manufactured by Enics Finland. Because the product doesn't have test points, functional testing with wires and connectors will be done instead of ICT-test using a bed of nails. A J-Link device will not be needed with this product, because there are no components that need to be programmed.

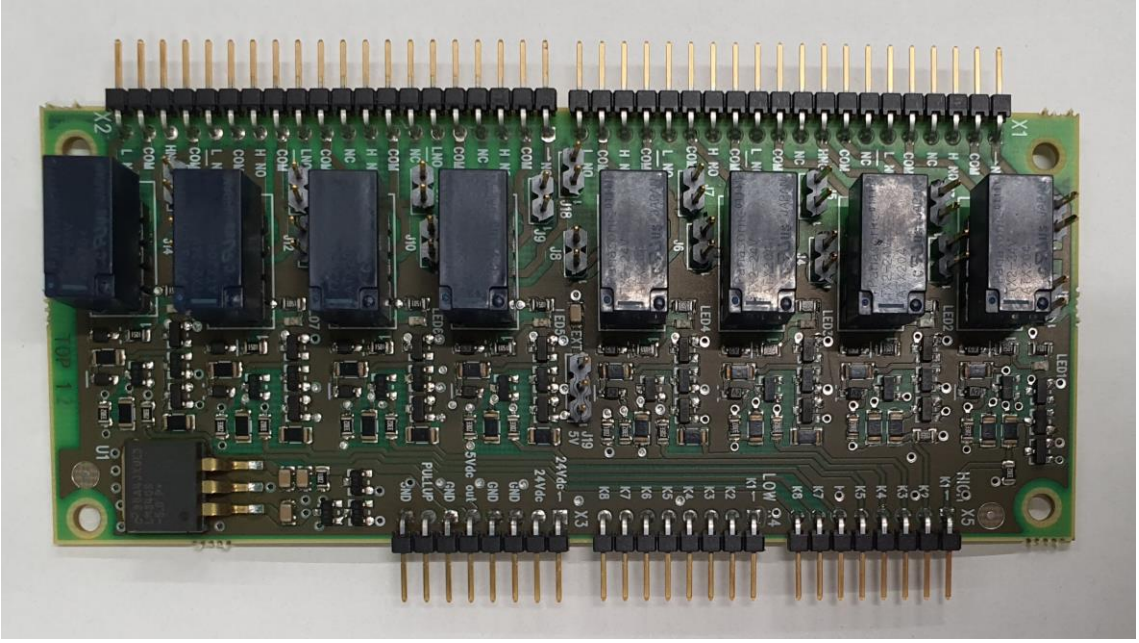

Figure 15. Relay multiplexer board by Enics Finland

The board contains 8 latching dual coil relays that require 24 volts to be operated. The board has first to be powered with a 24 V supply. In addition, a control voltage of 3.3-24 V is used for switching the relays. The relays are switched on and off in two ways: by feeding a control voltage into active high terminal or connecting active low terminal to the ground. It is necessary to have these two operation modes, because the outputs of different data acquisition I/O devices may be initially set either high or low.

The board has 64 pins. On the other side of the board there are 40 pins which act as the in– and out channels of the relays. On the other side there are three blocks of 8 pins. The first two of these contain the active high- and active low controls for the relays K1 through K8. The remaining 8 pins contain 24 V supply and ground pins. There is also a pullup-pin, to which a 3.3-24 V control voltage must be connected when wanting to use the active low controls. In addition, the board has a regulator that converts 24 V DC into 5 volts DC and feeds this into a 5 V DC out- pin.

6.2 Power Supply

Raspberry Pi is powered by its own 5 V / 1 A power supply. The power for the DUT is provided by Tenma 72-2710, which can be controlled through a USB interface with Standard Commands for Programmable Instruments commands (SCPI). SCPI is a standard for syntax and commands designed for controlling programmable test and measurement devices [10]. In testing this platform only two commands are needed:

- VSET:1:24 sets the output voltage at 24 V DC
- OUT1 turns the output on
- OUT0 turns the output off.

6.3 Control Signals to the DUT

The relays in the DUT have two coils. Each relay therefore has two output channels. To ascertain that the relays are functioning it is only necessary to know whether the contacts are open or closed. This can be done with digital inputs, but since one DAQ board only has eight of them, also analog inputs will be used. The functioning of the relays has to be checked both in case of active low- control and active high control.

The test specification requires that the test platform has at least three relay outputs, for which the RELAYplate daughterboard was included in the system. To demonstrate the operation of the relay daughterboard, testing the active high control of the DUT is done utilising a relay in RELAYplate. The active low control will instead be tested by pulling the pins low using GPIO pin 14 in the Raspberry Pi. Schematic part one in figure 16 describes the inputs to active low and active high controls, which are marked X4 and X5, respectively.

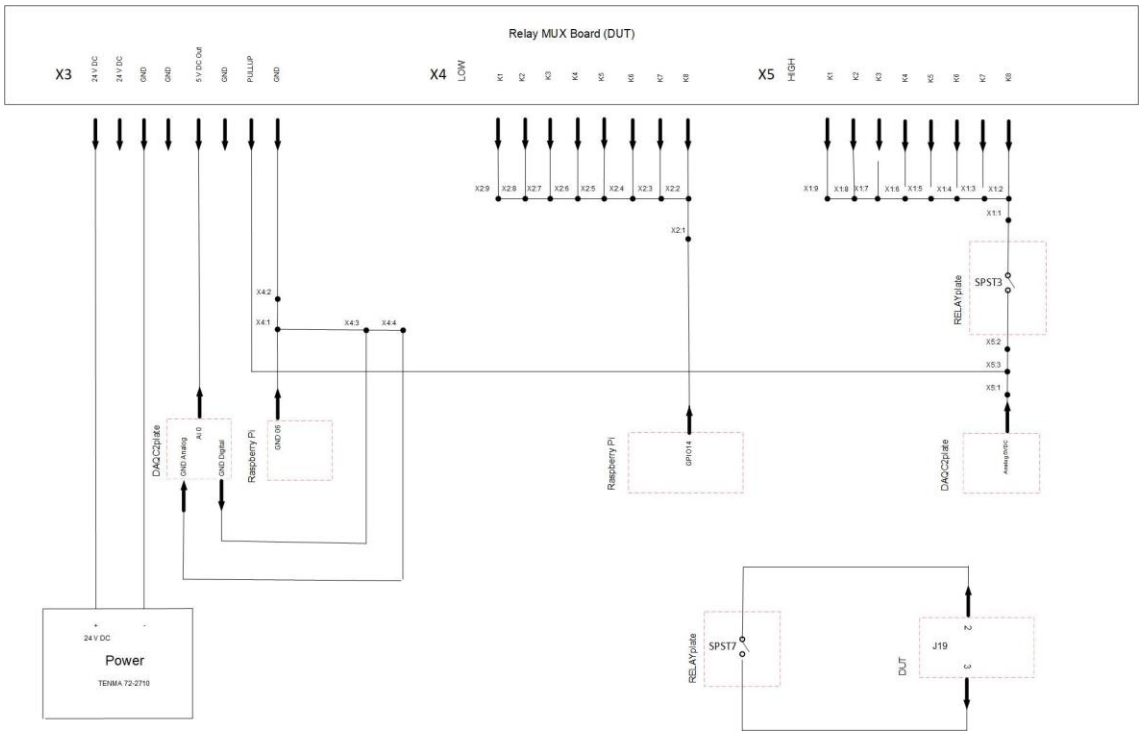

Figure 16. Schematic part one.

To test the active high controls the relays in the DUT will be controlled with RELAYplate. The pins of the active high control are in this way connected to 5 V pin in the DAQC2plate and they are switched on and off with relay 3 of the RELAYplate expansion board. One relay in the RELAYplate will also be used to connect together pins of jumper J19 on the DUT. This is used to connect the pullup into the active low control of the DUT, which pulls the control pins high.

6.4 Measuring the Inputs

A signal will be input to the “in” contacts of the relays in the DUT. The “out” contacts of the relays will then be measured using the data acquisition board to see whether the relay is in “on” or “off” state. One of the analog input pins of the DAQC2plate must be reserved for measuring the output of the voltage regulator. This is described in the schematic in figure 16. Therefore there are left 7 analog input- and 8 digital input pins and there are not enough input channels to measure all the out contacts of the relays at the same time with only one data acquisition board. This could be done in multiple ways, for example, switching test points

with relays and software code would give more analog input channels. However, to test the relays it is not necessary to test all the input- and output channels of every relay on the DUT. It is more straightforward and cost effective to link contacts in the relays together, so that fewer measurements will be needed, but the operation of the relays can still be tested.

Therefore, as described in the schematic in figure 17, the “out” contacts of relays K1, K2, K5 and K6 will be linked together and only the last output in them is measured using analog input channels. In the case of relays K3, K4, K7 and K8 every “out” contact will be measured by using the 8 digital input channels. When writing the code for the test sequence it has to be kept in mind that floating analog input pins will read approximately 1.38 V instead of zero.

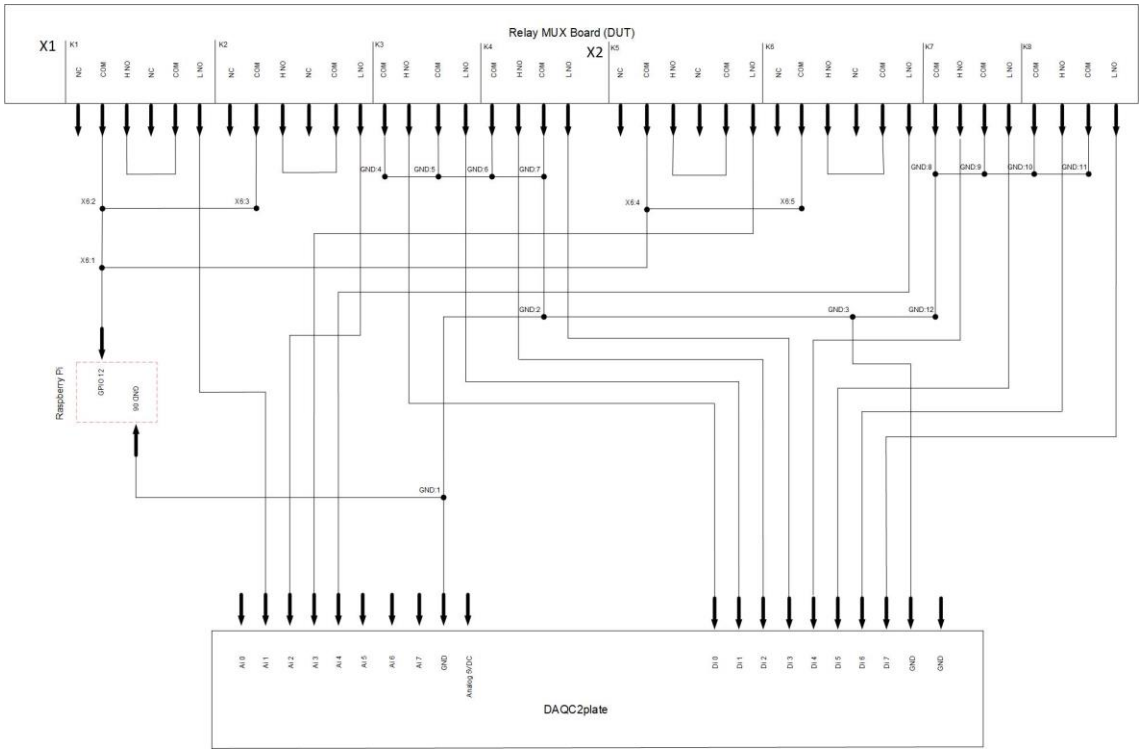

Figure 17. Schematic part two.

7 Testing

7.1 Test Specifications

The demonstration test of the platform consists of three parts. Below are the test specifications for functional testing of the DUT. The specifications consist of 49 functional tests divided into three sections. First the operation of voltage regulator is verified and then the relays of the DUT are tested in off- and on state using both active high- and active low control. In the test specification tests are numbered starting from functional test 1. The specifations contain the test point where the measurement is made, the channel where the measurement is made in the platform, the value expected for passing the test wit the margin of error and the test method.

7.1.1 U1 – Voltage Regulator

Table 1 describes the first test of the test sequence. The voltage regulator is tested by feeding 24 V from the DC power source into 24 V pin in the Relay MUX board. The test point is the 5 V DC out-pin in the DUT. The output is then measured with the analog input pin 0 in the DAQC2plate.

Table 1. Test for verifying that the regulator is functioning properly.

ID	Test point	Measurement channel	Limit value	Test Method
	- Switch on external 24V supply (Relayplate K-1?)			
FCT1	X3_5Vdc out	DAQC2plate_Ai1	5000 mV ±5%	Voltage measurement

To test the relays, GPIO 12 in Raspberry Pi is switched high, and it feeds an input voltage of 3.3 V to the “in” contacts, or COM-pins, of the relays K1, K2, K5 and K6. In the case of the rest of the relays, which are measured with digital input channels of the DAQC2plate, the COM pins will be connected to ground. Relay

K7 of the RELAYplate is also switched on before FCT2 and it acts as a jumper to pull the control pins of the DUT high as already mentioned.

7.1.2 X4 – Active low control

Tests numbered as FCT2-FCT13 in table 2 are the first part of testing the active low control of the DUT. These tests are for the measurement of the relays in the board in off state. To keep the active low control pins high, GPIO 14 in Raspberry Pi is switched high. Analog input channels Ai1-Ai4 of the DAQC2plate are used to measure the “out” contacts, or L NO-pins, of relays K1, K2, K5 and K6 in the DUT. As already stated earlier, the digital input channels Di0-Di7 of the DAQC2plate will be used for measuring relays K3, K4, K7 and K8 and they will measure every out contact of the relays. Because the “in” contacts are connected to the ground, the reading is inverse and the digital input pins return 0 in “on” state and 1 in “off” state.

When there is no voltage connected to the analog input channels, they are in a floating state and read approximately 1.375 V with slight variance. This is not a problem, because the output in the off state does not actually need to be measured accurately. It is enough to know that the contact is not in a high state. Margin of error in the software must be 10 % to each direction, slightly higher than in the other cases.

Table 2. Off-state tests for relays K1-K8 with active low control.

ID	Test point	Measurement channel	Limit value	Test Method
- Switch on external 24V supply - Switch on GPIO12 on Raspberry Pi - Switch on GPIO14 on Raspberry Pi - Switch on relay K7 in RELAYplate				
FCT2	X1_3, X1_6	DAQC2plate_Ai1	1375 mV ±10%	Voltage measurement
FCT3	X1_9, X1_12	DAQC2plate_Ai2	1375 mV ±10%	Voltage measurement
FCT4	X1_14	DAQC2plate_Di0	1	0=on, 1=off

FCT5	X1_16	DAQC2plate_Di1	1	0=on, 1=off
FCT6	X1_18	DAQC2plate_Di2	1	0=on, 1=off
FCT7	X1_20	DAQC2plate_Di3	1	0=on, 1=off
FCT8	X2_3, X2_6	DAQC2plate_Ai3	1375 mV ±10%	Voltage measurement
FCT9	X2_9, X2_12	DAQC2plate_Ai4	1375 mV ±10%	Voltage measurement
FCT10	X2_14	DAQC2plate_Di4	1	0=on, 1=off
FCT11	X2_16	DAQC2plate_Di5	1	0=on, 1=off
FCT12	X2_18	DAQC2plate_Di6	1	0=on, 1=off
FCT13	X2_20	DAQC2plate_Di7	1	0=on, 1=off

The next set of tests in table 3, numbered FCT14-FCT25, form the second part of testing the active low control. In these tests the “out” contacts are measured while the relays in the DUT are switched on. This is done by switching Raspberry Pi GPIO 14 low. Then the “out” contacts of the relays are measured again. This time to pass the test, the analog measurements are expected to return 3.3 V with a 5 % margin of error in each direction. The digital input pins are expected to return zero.

Table 3. On-state tests for relays K1-K8 with active low control.

ID	Test point	Test description	Limit value	Test Method
- Switch off GPIO14 on Raspberry Pi				
FCT14	X1_3, X1_6	DAQC2plate_Ai1	3300 mV ±5%	Voltage measurement
FCT15	X1_9, X1_12	DAQC2plate_Ai2	3300 mV ±5%	Voltage measurement
FCT16	X1_14	DAQC2plate_Di0	0	0=on, 1=off
FCT17	X1_16	DAQC2plate_Di1	0	0=on, 1=off
FCT18	X1_18	DAQC2plate_Di2	0	0=on, 1=off
FCT19	X1_20	DAQC2plate_Di3	0	0=on, 1=off
FCT20	X2_3, X2_6	DAQC2plate_Ai3	3300 mV ±5%	Voltage measurement
FCT21	X2_9, X2_12	DAQC2plate_Ai4	3300 mV ±5%	Voltage measurement
FCT22	X2_14	DAQC2plate_Di4	0	0=on, 1=off

FCT23	X2_16	DAQC2plate_Di5	0	0=on, 1=off
FCT24	X2_18	DAQC2plate_Di6	0	0=on, 1=off
FCT25	X2_20	DAQC2plate_Di7	0	0=on, 1=off

7.1.3 X5 – Active high control

Table 4 describes the first part of testing the active high control. At the start of active high tests the jumper relay K7 in RELAYplate must be switched off because both active high- and active low controls cannot be used at the same time. Naturally, when measuring the “out” contacts while the relays of the DUT are in “off” state the same measurement results will be expected as in tests FCT2-FCT14.

Table 4. Off-state tests for relays K1-K8 with active high control.

ID	Test point	Test description	Limit value	Test Method
-	Switch off relay K7 in RELAYplate			
FCT26	X1_3, X1_6	DAQC2plate_Ai1	1375 mV ±10%	Voltage measurement
FCT27	X1_9, X1_12	DAQC2plate_Ai2	1375 mV ±10%	Voltage measurement
FCT28	X1_14	DAQC2plate_Di0	1	0=on, 1=off
FCT29	X1_16	DAQC2plate_Di1	1	0=on, 1=off
FCT30	X1_18	DAQC2plate_Di2	1	0=on, 1=off
FCT31	X1_20	DAQC2plate_Di3	1	0=on, 1=off
FCT32	X2_3, X2_6	DAQC2plate_Ai3	1375 mV ±10%	Voltage measurement
FCT33	X2_9, X2_12	DAQC2plate_Ai4	1375 mV ±10%	Voltage measurement
FCT34	X2_14	DAQC2plate_Di4	1	0=on, 1=off

FCT35	X2_16	DAQC2plate_Di5	1	0=on, 1=off
FCT36	X2_18	DAQC2plate_Di6	1	0=on, 1=off
FCT37	X2_20	DAQC2plate_Di7	1	0=on, 1=off

The last set of tests in table 5, numbered FCT38-FCT49, are for measuring the output of the relays while they are switched on using active high control. To do this, relay K3 in the RELAYplate is switched on, which will feed a “high” signal into the pins of active high control. For passing these tests, same results are expected from the measurements as in tests FCT14-FCT25. At the end of the tests the system is reset. This means that relay 3 in RELAYplate and GPIO 12 in Raspberry Pi are turned off. The voltage output of the external power source is turned off and the voltage is set to 0 V.

Table 5. On-state tests for relays K1-K8 with active high control.

ID	Test point	Test description	Limit value	Test Method
-	Switch on relay K3 on RELAYplate			
FCT38	X1_3, X1_6	DAQC2plate_Ai1	3300 mV ±5%	Voltage measurement
FCT39	X1_9, X1_12	DAQC2plate_Ai2	3300 mV ±5%	Voltage measurement
FCT40	X1_14	DAQC2plate_Di0	0	0=on, 1=off
FCT41	X1_16	DAQC2plate_Di1	0	0=on, 1=off
FCT42	X1_18	DAQC2plate_Di2	0	0=on, 1=off
FCT43	X1_20	DAQC2plate_Di3	0	0=on, 1=off
FCT44	X2_3, X2_6	DAQC2plate_Ai3	3300 mV ±5%	Voltage measurement
FCT45	X2_9, X2_12	DAQC2plate_Ai4	3300 mV ±5%	Voltage measurement
FCT46	X2_14	DAQC2plate_Di4	0	0=on, 1=off
FCT47	X2_16	DAQC2plate_Di5	0	0=on, 1=off
FCT48	X2_18	DAQC2plate_Di6	0	0=on, 1=off
FCT49	X2_20	DAQC2plate_Di7	0	0=on, 1=off
-	Switch off relay K3 on RELAYplate			
-	Switch off external 24V supply			
-	Switch off GPIO 12 on Raspberry Pi			

7.2 Test Sequence

The software that runs the tests defined in the test specifications is called the test sequence. The plan at the start of this project was that if the platform enters in production phase, an external provided could be used for developing the software. But testing the product requires software in any case and therefore a test software was developed that was as close to the final version as possible.

In the test specifications it was defined that the test sequence must be based on a well-known language like Python, HTML of XML. The latest version of Raspbian contains Thonny IDE which comes with built in Python and has a debugger in it. It was therefore a natural decision to use Python as the programming language for the test software.

Figure 18 is a flow chart representation of the test sequence. In the beginning of the program the limit values are read into an array from a separate text file. The code consists of 49 if-else conditional statements. In each of these a measurement made in a test point is compared to a predetermined limit value and then the result is written into a csv report file. To make debugging easier limit values, the value measured from the test point and the outcome are stored into variables, from where these can be checked in real time.

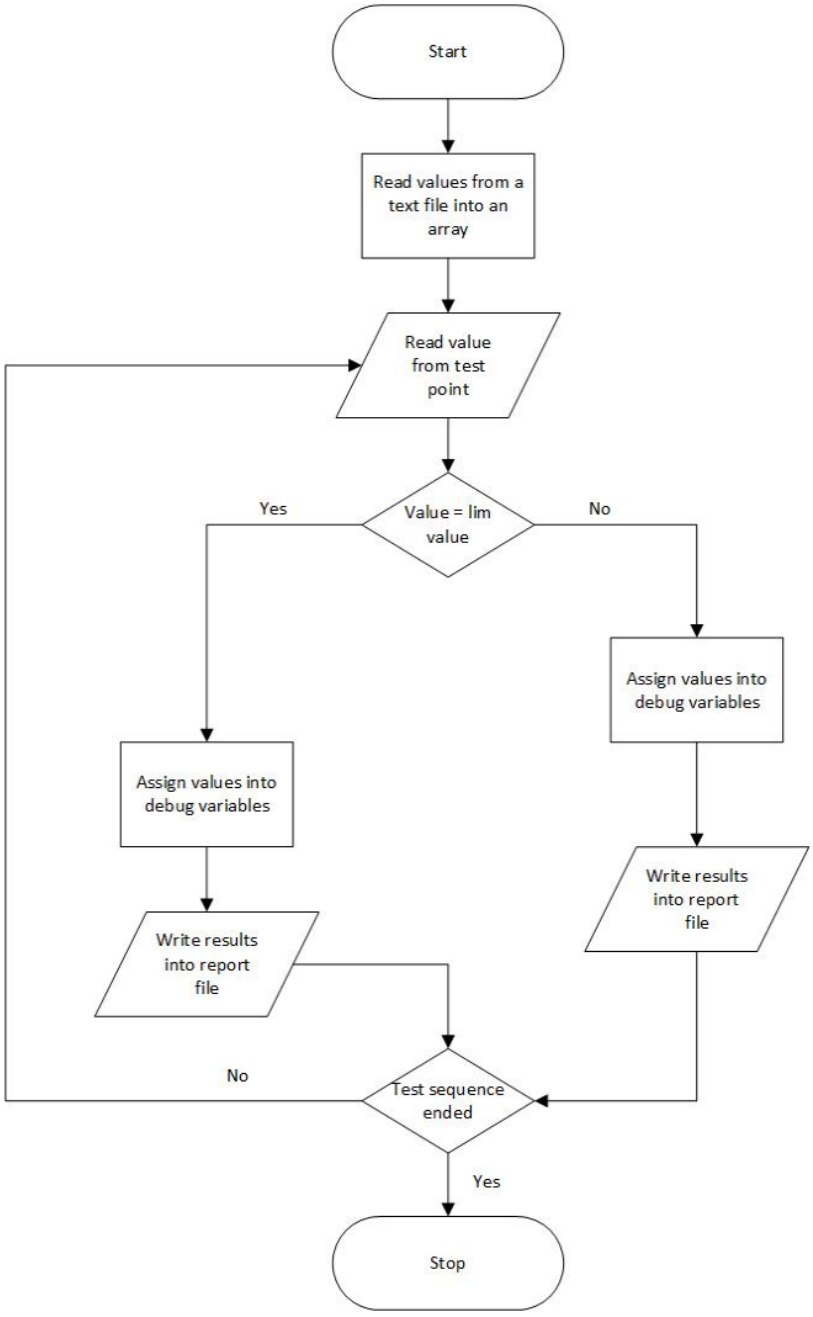

Figure 18. The test sequence.

Before writing a Python program for the platform, Pi-Plates Python libraries had to be installed. First it was checked that the system is up to date and upgraded by following two commands into command prompt:

```
sudo apt-get update
sudo apt-get upgrade
```

Then the libraries were installed with the following command:

```
sudo pip install pi-plates
```

The complete test program can be found in appendix 3. Figure 19 shows the start of program in Visual Studio Code. At the start the required libraries are imported and serial communication settings are done.

```
import csv
from time import sleep
from datetime import date
from datetime import datetime
import RPi.GPIO as GPIO
import piplates.RELAYplate as RELAY
import piplates.DAQC2plate as DAQC2
import sys
import serial
ser = serial.Serial(
    port='/dev/ttyACM0',
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=1
)
```

Figure 19. Start of the program in Visual Studio Code

Serial communication is used for communication with the Tenma power supply unit. Figure 20 contains the commands used for setting output voltage and current using SCPI commands. The commands have to be written using “bytes” type.

```
ser.write(bytes(b'VSET1:24.00'))    # Set the output voltage
sleep(0.5)
ser.write(bytes(b'ISET1:0.250'))    # Set the output current
sleep(0.5)
```

Figure 20. Serial communication with the power unit.

As seen in figure 21, reading the limit values into an array is done with a for loop by utilizing csv module from the Python Standard Library and using the reader object.

```
lim = []
with open('limits.txt', 'r') as fd:
    reader = csv.reader(fd)
    for row in reader:
        lim.append(row)
```

Figure 21. Making an array containing the limit values

In this way the limit values can be handled as elements of the array and the user does not have to touch the software when changing them. As an example, below in figure 22 you can see test number 40, which measures an “out” contact of relay K3 while the relay is set in “on” state with the active high control.

```
# FCT40(K3)
Fct40_Limits = "lo="+lim[39][1]+" hi="+lim[39][2]  # Show limit values in debugging
Fct40 = DAQC2.getDINbit(0, 0)                      # Read X1_14 with digital input 0 DAQC2plate
if Fct40 == float(lim[39][2]):
    Fct40_Result = "PASS"                          # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct40"+","+"X1_14"+","+str(Fct40)+","+"PASS"+"\n")
else:
    Fct40_Result = "FAIL"                          # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct40"+","+"X1_14"+","+str(Fct14)+","+"FAIL"+"\n")
```

Figure 22. Test 40 in the test sequence

At the start of test 40 the limit values are retrieved from array named “lim” as elements and stored into a variable named Fct40_Limits. The result of the measurement is stored into another variable named Fct40. Varible Fct40 is then compared with the expected limit value, which is again an element of the “lim” array. Finally the result of the test is stored in variable Fct40_Result and written into a report file. The test report can be found as appendix 2.

8 Conclusions

The purpose of this thesis was to see what a low-cost platform that was easy to manufacture would look like. Raspberry Pi was to be the control unit. The project started with choosing the components. This was the most time-consuming part of the project, largely due to limited experience with the topic. Once the parts for the platform had been chosen, the task became relatively straightforward

The prototype can be seen in figure 23 connected to the DUT on top of the cover of the casing. The result would be esthetically better if the wiring was simplified. Making only one measurement for each relay in the final version of the product would help to accomplish this.

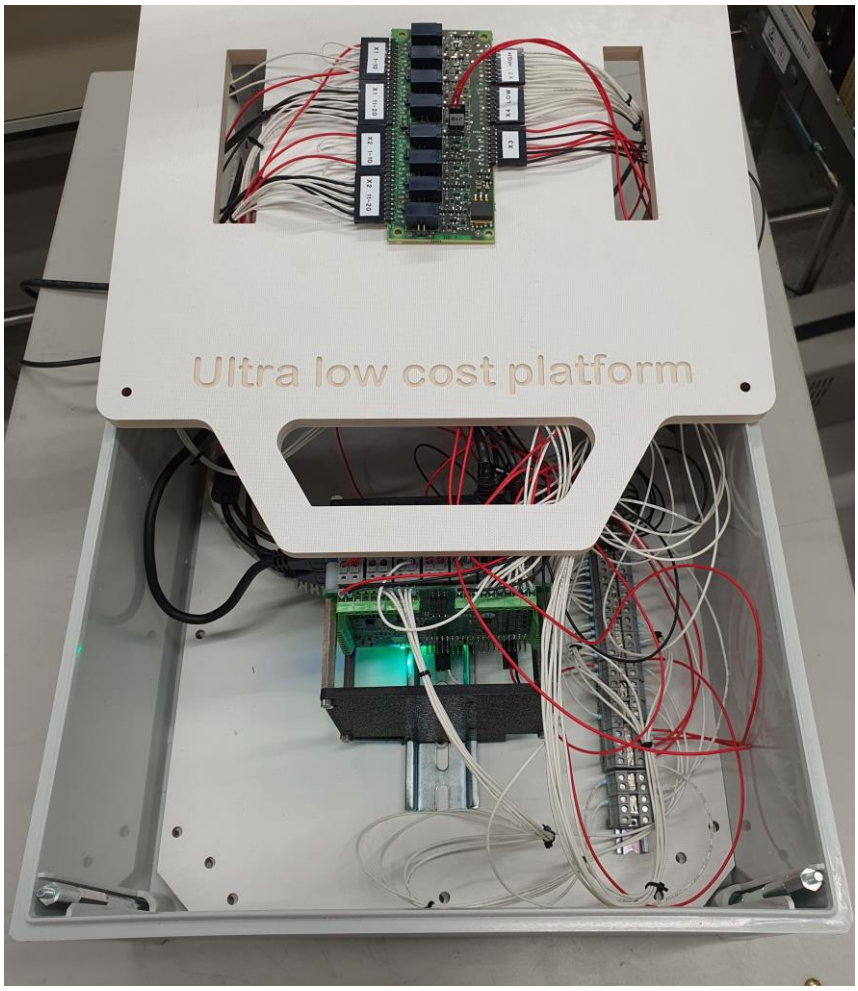

Figure 23. The prototype connected to the DUT.

The plan was to look for ready-made expansion boards for the controller. It turned out that there are plenty of cheap expansion boards available in the market which offer an adequate performance for testing many PCBs. E.g. the DAQC2plate used is stackable and altogether could offer 64 analog and 64 digital input channels. However, this is obviously not adequate for all products, since PCBs can have hundreds of test points. Additionally, when stacking multiple daughterboards on top of Raspberry Pi, the power requirements must be taken into account and the system might need auxiliary power supply. With stacking the price would also go up to some extent. Because ot this, the system would be optimal for a product with a few dozen test points at the most.

Developing the software turned out to be much simpler than was originally thought. However, it was not possible to finish the software with the time available. The test software is operable, but it does not have connection to the Enics MES system and it is missing version control. Also storing the test results in Enics TDC xml of ATML format and support for local languages are missing. These features were left for further development.

References

1. Houdek, Cal. Inspection and testing methods for PCBs [Internet]. An overview: Caltronics Design & Assembly Inc. [Online.] https://caltronicsdesign.com/wp-content/uploads/2016/11/Inspection-and-testing-methods-for-PCBs-an-overview.pdf accessed 29 May 2022.

2. The PCB Test Point and Its Importance to Circuit Board Manufacturing: Cadence PCB solutions [Online]. The PCB Test Point and Its Importance to Circuit Board Manufacturing (cadence.com) accessed 29 May 2022.

3. Floyd, Thomas. 2012. Electronic Devices Ninth Edition. Prentice Hall

4. What is LabVIEW?: National Instruments [Online]. What is LabVIEW? - NI accessed 29 May 2022.

5. What is TestStand: National Instruments [Online]. What is TestStand? - NI accessed 29 May 2022.

6. Raspberry Pi 4: Raspberry Pi Foundation [Online]. Buy a Raspberry Pi 4 Model B – Raspberry Pi accessed 29 May 2022.

7. Bench Power Supply; Programmable, 30V, 5A, 1CH, USB, RS232 [Online]. 2207204.pdf (farnell.com) accessed 9 May 2022.

8. DAQC2plate Users Guide: Pi-Plates [Online]. DAQC2plate Users Guide – Pi-Plates accessed 9 May 2022.

9. RELAYplate Users Guide: Pi-Plates [Online]. RELAYplate Users Guide – Pi-Plates accessed 9 May 2022.

10. SCPI, A Programming Language for Measurement Device: FILPAL Xpress [Online]. SCPI, A Programming Language for Measurement Device – FILPAL XPress (wordpress.com) accessed 9 May 2022.

Specifications for Ultra Low Cost Test Platform

1 Processor board

ID	Type	Description	Verification criteria
RQ-200	FUNC	Processor board shall have HDMI interface for external display	Platform review
RQ-201	FUNC	Processor board shall have minimum 2 USB 3.0 ports	Platform review
RQ-202	FUNC	Processor board shall have one SPI/I2C interface	Platform review
RQ-203	FUNC	Processor board shall have 4 GPIO lines	Platform review
RQ-204	FUNC	Processor board shall have minimum 1 Ethernet connection with 1Gbps	Platform review
RQ-205	FUNC	Processor board shall have possibility to add second Ethernet interface via USB	Platform review
RQ-206	FUNC	Processor board shall have Wifi interface. Wifi can be in module (recommended) or possibility add by external module.	Platform review
		Rasberry Pi 4 is recommended device	
RQ-208	FUNC	Processor board shall have light operating system (e.g. Ubuntu or Win10 IoT)	Platform review
RQ-209	FUNC	Processor board operating system shall be possiblity to update via internet connection	Platform review
RQ-210	FUNC	Processor board shall support USB 2D code scanner	Platform review

2 Measurements / controls

ID	Type	Description	Verification criteria
RQ-300	MEAS	Platform shall have minimum one DVM measurement channel, 0-30V / 2%	Platform review
RQ-301	MEAS	Platform shall have minimum 3 relay outputs	Platform review
RQ-302	MEAS	Platform shall have minimum 3 input channels	Platform review
RQ-303	MEAS	Platform shall have minimum 3 output channels	Platform review
RQ-304	MEAS	Platform shall have Ethernet (or USB) controlled power supply with minimum 30V/1A.	Platform review
RQ-305	MEAS	Platform IO devices shall be off-the-shelf	Platform review

3 Test sequenses / SW interfaces

ID	Type	Description	Verification criteria
RQ-400	SEQ	Platform shall have possiblity to manage sequences (products) via Ethernet connection	Demo review
RQ-401	SEQ	Test sequence shall based on well known language (e.g. Python, HTML, XML..) or UI for making test sequences shall be available	Platform review
RQ-402	SEQ	Platform shall have interfaces for GPIO, SPI	Demo review
RQ-403	SEQ	Platform shall have interface for J-link programming device. STM32 microcontrollers shall be supported.	Demo review
RQ-404	SEQ	Platform shall have interface for power supply unit.	Demo review
RQ-405	SEQ	Platform shall have local storage location for test results.	Demo review
RQ-406	SEQ	Local database format shall be documented.	Demo review
RQ-407	SEQ	Local data shall be remote readable	Demo review
RQ-408		Local test result shall be stored Enics TDC xml or ATML format.	Demo review
RQ-409	SEQ	Platform shall have interface for Enics data collection database.	Demo review
RQ-410	SEQ	Platform shall be remote configured	Demo review
RQ-411	SEQ	Platform shall have support for Enics MES.	Demo review
RQ-412	SEQ	Test sequence shall support serial number reading.	Demo review
RQ-413	SEQ	Test sequences shall have revision management.	Demo review
RQ-414	SEQ	Platform shall be able to operate off-line.	Demo review

4 User interface

ID	Type	Description	Verification criteria
RQ-500	UI	Platform shall three colour indicator for operator: GREEN- ok ; YELLOW- Operating, RED-fail	Platform review
RQ-501	UI	Platform UI shall have test start button	Platform review
RQ-502	UI	Platform display shall be with touch interface	Platform review
RQ-503	UI	Test software UI shall support touch interface	Platform review
RQ-504	UI	Test software UI shall have clear OK / FAIL information	Demo review
RQ-505	UI	Platform shall have possibility to add external display via HDMI	Platform review
RQ-506	UI	Test software UI shall be done according Enics visual guideline	Demo review
RQ-507	UI	Test software UI shall support local languages	Demo review
RQ-508	UI	Test Platform Software shall provide means to display Operator messages, instructions and questions in localized language	Demo review
RQ-509	UI	Test sequence shall be possible to run step by step (via remote interface)	Demo review
RQ-510	UI	Test software UI shall have debug mode. Debug UI shall have: Test step, result, ok/pass and limits information. Debug UI information shall be real time.	Demo review
RQ-511	UI	Operator identification shall be available. Operator information stored to test logs.	Demo review

5 Mechanical requirements

ID	Type	Description	Verification criteria
RQ-600	MEC	Basic Platform device (processor board + basic IO interfaces) shall have casing.	Demo review
RQ-601	MEC	Platform shall have selected test probe mechanics.	Platform review
RQ-602	MEC	Selected test probe mechanics shall be off-the-shelf	Platform review
RQ-603	MEC	Probe mechanics shall support product changes, adapter solution.	Platform review
RQ-604	MEC	Probe mechanics shall support basic platform device (RQ-600). Integrated to probe mechanics (Ingun, GPS, Enics)	Platform review

Test Report

Test Report	2022-05-30	09:29:05 AM
Relay MUX Board	54852891011	

Test ID	Test point	Value	Result
Fct1	X3 5VDC out	5.056	PASS
Fct2	X1_3 X1_6	1.345	PASS
Fct3	X1_9 X1_12	1.346	PASS
Fct4	X1_14	1	PASS
Fct5	X1_16	1	PASS
Fct6	X1_18	1	PASS
Fct7	X1_20	1	PASS
Fct8	X2_3 X2_6	1.332	PASS
Fct9	X2_9 X2_12	1.329	PASS
Fct10	X2_14	1	PASS
Fct11	X2_16	1	PASS
Fct12	X2_18	1	PASS
Fct13	X2_20	1	PASS

Tests for Relays K1-K8 in on-state with active low control

Test ID	Test point	Value	Result
Fct14	X1_3 X1_6	3.3	PASS
Fct15	X1_9 X1_12	3.3	PASS
Fct16	X1_14	0	PASS
Fct17	X1_16	0	PASS
Fct18	X1_18	0	PASS
Fct19	X1_20	0	PASS
Fct20	X2_3 X2_6 X2_9	3.3	PASS
Fct21	X2_12_14	3.3	PASS
Fct22	X2_14	0	PASS
Fct23	X2_16	0	PASS
Fct24	X2_18	0	PASS
Fct25	X2_20	0	PASS

Tests for Relays K1-K8 in off-state with active high control

Test ID	Test point	Value	Result
Fct26	X1_3 X1_6	1.342	PASS
Fct27	X1_9 X1_12	1.336	PASS
Fct28	X1_14	1	PASS
Fct29	X1_16	1	PASS
Fct30	X1_18	1	PASS
Fct31	X1_20	1	PASS
Fct32	X2_3 X2_6	1.319	PASS
Fct33	X2_9 X2_12	1.327	PASS
Fct34	X2_14	1	PASS
Fct35	X2_16	1	PASS
Fct36	X2_18	1	PASS
Fct37	X2_20	1	PASS

Tests for Relays K1-K8 in on-state with active high control

Test ID	Test point	Value	Result
Fct38	X1_3 X1_6	3.302	PASS
Fct39	X1_9 X1_12	3.302	PASS
Fct40	X1_14	0	PASS
Fct41	X1_16	0	PASS
Fct42	X1_18	0	PASS
Fct43	X1_20	0	PASS
Fct44	X2_3 X2_6	3.3	PASS
Fct45	X2_9 X2_12	3.3	PASS
Fct46	X2_14	0	PASS
Fct47	X2_16	0	PASS
Fct48	X2_18	0	PASS
Fct49	X2_20	0	PASS

The Test Sequence

```
import csv
import time
from time import sleep
from datetime import date
from datetime import datetime
import RPi.GPIO as GPIO
import piplates.RELAYplate as RELAY
import piplates.DAQC2plate as DAQC2
import sys
import serial
ser = serial.Serial(
    port='/dev/ttyACM0',  # Replace ttyS0 with ttyAM0 for Pi1,Pi2,Pi0
    baudrate=9600,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=1
)
RELAY.getID(0)
'Pi-Plate RELAY'

# Set the output voltage
ser.write(bytes(b'VSET1:24.00'))
sleep(0.5)
ser.write(bytes(b'ISET1:0.250'))
sleep(0.5)
GPIO.setmode(GPIO.BCM)
GPIO.setup(14, GPIO.OUT)
GPIO.setup(12, GPIO.OUT)

sleep(0.5)

now = datetime.now()
current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)
today = date.today()
print("Today's date:", today)

barcode = input("Scan barcode. If the product doesn't have any, press enter: ")
print("Barcode scanned: " + barcode)
sleep(0.5)

# Reading limit values from a textfile into an array
lim = []
with open('limits.txt', 'r') as fd:
    reader = csv.reader(fd)
    for row in reader:
        lim.append(row)

Report_Name = "Report_"+str(barcode)+".csv"
```

```
RELAY.relayON(0, 7)
RELAY.relayON(0, 1)
sleep(0.5)

# Regulator test
Fct1_Limits = "lo="+lim[0][1]+" hi=" + \
    lim[0][2]  # Show limit values in debugging
Fct1     =     DAQC2.getADC(0,0)      #     Read     X3_5     with     analog     input     0
DAQC2plate
with open(Report_Name, 'w') as f:
    f.write("Test Report "+","+str(today)+","+current_time+"\n")
    f.write("Relay MUX Board "+","+str(barcode)+"\n")
if float(lim[0][1]) < Fct1 < float(lim[0][2]):
    Fct1_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Test ID"+","+"Test point"+","+"Value"+","+"Result"+"\n")
        f.write("Fct1"+","+"X3 5VDC out"+","+str(Fct1)+","+"PASS"+"\n")
else:
    Fct1_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Test ID"+","+"Test point"+","+"Value"+","+"Result"+"\n")
        f.write("Fct1"+","+"X3 5VDC out"+","+str(Fct1)+","+"FAIL"+"\n")

# Off-state tests for relays K1-K8 with active low control:
# Turn off the voltage output
ser.write(bytes(b'OUT0'))
sleep(0.5)
print("ACTIVE LOW TESTS")
GPIO.output(12, 1)
GPIO.output(14, 1)
sleep(0.5)

# FCT2(K1)
Fct2_Limits = "lo="+lim[1][1]+" hi=" + \
    lim[1][2]  # Show limit values in debugging
Fct2 = DAQC2.getADC(0, 1)  # Read X1_3 X1_6 with analog input 1 DAQC2plate

if float(lim[1][1]) < Fct2 < float(lim[1][2]):
    Fct2_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("\nTests for Relays K1-K8 in off-state with active low control\n\n")
        f.write("Test ID"+","+"Test point"+","+"Value"+","+"Result"+"\n")
        f.write("Fct2"+","+"X1_3 X1_6"+","+str(Fct2)+","+"PASS"+"\n")
else:
    Fct2_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("\nTests for Relays K1-K8 in off-state with active low control\n\n")
        f.write("Test ID"+","+"Test point"+","+"Value"+","+"Result"+"\n")
        f.write("Fct2"+","+"X1_3 X1_6"+","+str(Fct2)+","+"FAIL"+"\n")

Fct3_Limits = "lo="+lim[2][1]+" hi=" + \
    lim[2][2]  # Show limit values in debugging
Fct3 = DAQC2.getADC(0, 2)  # Read X1_9 X1_12 with analog input 2 DAQC2plate
if float(lim[2][1]) < Fct3 < float(lim[2][2]):
    Fct3_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct3"+","+"X1_9 X1_12"+","+str(Fct3)+","+"PASS"+"\n")
else:
    Fct3_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct3"+","+"X1_9 X1_12"+","+str(Fct3)+","+"FAIL"
```

```
# FCT4(K3)
Fct4_Limits = "lo="+lim[3][1]+" hi=" + \
    lim[3][2]  # Show limit values in debugging
Fct4 = DAQC2.getDINbit(0, 0)  # Read X1_14 with digital input 0 DAQC2plate
if Fct4 == float(lim[3][2]):
    Fct4_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct4"+","+"X1_14"+","+str(Fct4)+","+"PASS"+"\n")
else:
    Fct4_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct4"+","+"X1_14"+","+str(Fct4)+","+"PASS"+"\n")

# FCT5(K3)
Fct5_Limits = "lo="+lim[4][1]+" hi=" + \
    lim[4][2]  # Show limit values in debugging
Fct5 = DAQC2.getDINbit(0, 1)  # Read X1_16 with digital input 1 DAQC2plate
if Fct5 == float(lim[4][2]):
    Fct5_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct5"+","+"X1_16"+","+str(Fct5)+","+"PASS"+"\n")
else:
    Fct5_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct5"+","+"X1_16"+","+str(Fct5)+","+"PASS"+"\n")

# FCT6(K4)
Fct6_Limits = "lo="+lim[5][1]+" hi=" + \
    lim[5][2]  # Show limit values in debugging
Fct6 = DAQC2.getDINbit(0, 2)  # Read X1_18 with digital input 2 DAQC2plate
if Fct6 == float(lim[5][2]):
    Fct6_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct6"+","+"X1_18"+","+str(Fct6)+","+"PASS"+"\n")
else:
    Fct6_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct6"+","+"X1_18"+","+str(Fct6)+","+"PASS"+"\n")

# FCT7(K4)
Fct7_Limits = "lo="+lim[5][1]+" hi=" + \
    lim[5][2]  # Show limit values in debugging
Fct7 = DAQC2.getDINbit(0, 3)  # Read X1_20 with digital input 3 DAQC2plate
if Fct7 == float(lim[6][2]):
    Fct7_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct7"+","+"X1_20"+","+str(Fct7)+","+"PASS"+"\n")
else:
    Fct7_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct7"+","+"X1_20"+","+str(Fct7)+","+"PASS"+"\n")

# FCT8(K5)
Fct8_Limits = "lo="+lim[7][1]+" hi=" + \
    lim[7][2]  # Show limit values in debugging
Fct8 = DAQC2.getADC(0, 3)  # Read X2_3 X2_6 with analog input 3 DAQC2plate
if float(lim[7][1]) < Fct8 < float(lim[7][2]):
    Fct8_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct8"+","+"X2_3 X2_6"+","+str(Fct8)+","+"PASS"+"\n")
else:
    Fct8_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct8"+","+"X2_3 X2_6"+","+str(Fct8)+","+"PASS"+"\n")

# FCT9(K6)
Fct9_Limits = "lo="+lim[8][1]+" hi=" + \
    lim[8][2]  # Show limit values in debugging
Fct9 = DAQC2.getADC(0, 4)  # Read X2_9 X2_12 with analog input 4 DAQC2plate
if float(lim[8][1]) < Fct9 < float(lim[8][2]):
    Fct9_Result = "PASS"  # Show test result in debugging
```

```
with open(Report_Name, 'a') as f:
        f.write("Fct9"+","+"X2_9 X2_12"+","+str(Fct9)+","+"PASS"+"\n")
else:
    Fct9_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct9"+","+"X2_9 X2_12"+","+str(Fct9)+","+"PASS"+"\n")

# FCT10(K7)
Fct10_Limits = "lo="+lim[9][1]+" hi=" + \
    lim[9][2]  # Show limit values in debugging
Fct10 = DAQC2.getDINbit(0, 4)  # Read X2_14 with digital input 4 DAQC2plate
if Fct10 == float(lim[9][2]):
    Fct10_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct10"+","+"X2_14"+","+str(Fct10)+","+"PASS"+"\n")
else:
    Fct10_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct10"+","+"X2_14"+","+str(Fct10)+","+"PASS"+"\n")

# FCT11(K7)
Fct11_Limits = "lo="+lim[10][1]+" hi=" + \
    lim[10][2]  # Show limit values in debugging
Fct11 = DAQC2.getDINbit(0, 5)  # Read X2_16 with digital input 5 DAQC2plate
if Fct11 == float(lim[10][2]):
    Fct11_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct11"+","+"X2_16"+","+str(Fct11)+","+"PASS"+"\n")
else:
    Fct11_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct11"+","+"X2_16"+","+str(Fct11)+","+"FAIL"+"\n")

# FCT12(K8)
Fct12_Limits = "lo="+lim[11][1]+" hi=" + \
    lim[11][2]  # Show limit values in debugging
Fct12 = DAQC2.getDINbit(0, 6)  # Read X2_18 with digital input 6 DAQC2plate
if Fct12 == float(lim[11][2]):
    Fct12_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct12"+","+"X2_18"+","+str(Fct12)+","+"PASS"+"\n")
else:
    Fct12_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct12"+","+"X2_18"+","+str(Fct12)+","+"FAIL"+"\n")

# FCT13(K8)
Fct13_Limits = "lo="+lim[12][1]+" hi=" + \
    lim[12][2]  # Show limit values in debugging
Fct13 = DAQC2.getDINbit(0, 7)  # Read X2_20 with digital input 7 DAQC2plate
if Fct13 == float(lim[12][2]):
    Fct1_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct13"+","+"X2_20"+","+str(Fct13)+","+"PASS"+"\n")
else:
    Fct13_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct13"+","+"X2_20"+","+str(Fct13)+","+"FAIL"+"\n")

# On state tests for relays K1-K8 with active low control
# Turn on the output voltage
ser.write(bytes(b'OUT1'))
sleep(0.5)
# FCT14(K1)
GPIO.output(14, 0)
sleep(0.5)

Fct14_Limits = "lo="+lim[13][1]+" hi=" + \
    lim[13][2]  # Show limit values in debugging
Fct14 = DAQC2.getADC(0, 1)  # Read X1_3, X1_6 with analog input 1 DAQC2plate
if float(lim[13][1]) < Fct14 < float(lim[13][2]):
    Fct14_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
```

```
        f.write("\nTests for Relays K1-K8 in on-state with active low control\n\n")
        f.write("Test ID"+","+"Test point"+","+"Value"+","+"Result"+"\n")
        f.write("Fct14"+","+"X1_3 X1_6"+","+str(Fct14)+","+"PASS"+"\n")
else:
    Fct14_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("\nTests for Relays K1-K8 in on-state with active low control\n\n")
        f.write("Test ID"+","+"Test point"+","+"Value"+","+"Result"+"\n")
        f.write("Fct14"+","+"X1_3 X1_6"+","+str(Fct14)+","+"FAIL"+"\n")

# FCT15(K2)
Fct15_Limits = "lo="+lim[14][1]+" hi=" + \
    lim[14][2]  # Show limit values in debugging
Fct15 = DAQC2.getADC(0, 2)  # Read X1_9, X1_12 with analog input 2 DAQC2plate
if float(lim[14][1]) < Fct15 < float(lim[14][2]):
    Fct15_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct15"+","+"X1_9 X1_12"+","+str(Fct15)+","+"PASS"+"\n")
else:
    Fct15_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct15"+","+"X1_9 X1_12"+","+str(Fct15)+","+"FAIL"+"\n")

# FCT16(K3)
Fct16_Limits = "lo="+lim[15][1]+" hi=" + \
    lim[15][2]  # Show limit values in debugging
Fct16 = DAQC2.getDINbit(0, 0)  # Read X1_14 with digital input 0 DAQC2plate
if Fct16 == float(lim[15][2]):
    Fct16_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct16"+","+"X1_14"+","+str(Fct16)+","+"PASS"+"\n")
else:
    Fct16_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct16"+","+"X1_14"+","+str(Fct16)+","+"FAIL"+"\n")

# FCT17(K3)
Fct17_Limits = "lo="+lim[16][1]+" hi=" + \
    lim[16][2]  # Show limit values in debugging
Fct17 = DAQC2.getDINbit(0, 1)  # Read X1_16 with digital input 1 DAQC2plate
if Fct17 == float(lim[16][2]):
    Fct17_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct17"+","+"X1_16"+","+str(Fct17)+","+"PASS"+"\n")
else:
    Fct17_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct17"+","+"X1_16"+","+str(Fct17)+","+"FAIL"+"\n")

# FCT18(K4)
Fct18_Limits = "lo="+lim[17][1]+" hi=" + \
    lim[17][2]  # Show limit values in debugging
Fct18 = DAQC2.getDINbit(0, 2)  # Read X1_18 with digital input 2 DAQC2plate
if Fct18 == float(lim[17][2]):
    Fct18_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct18"+","+"X1_18"+","+str(Fct18)+","+"PASS"+"\n")
else:
    Fct18_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct18"+","+"X1_18"+","+str(Fct18)+","+"FAIL"+"\n")


# FCT19(K4)
Fct19_Limits = "lo="+lim[18][1]+" hi=" + \
    lim[18][2]  # Show limit values in debugging
Fct19 = DAQC2.getDINbit(0, 3)  # Read X1_20 with digital input 3 DAQC2plate
if Fct19 == float(lim[18][2]):
    Fct19_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct19"+","+"X1_20"+","+str(Fct19)+","+"PASS"+"\n")
else:
```

```
    Fct19_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct19"+","+"X1_20"+","+str(Fct19)+","+"FAIL"+"\n")

# FCT20(K5)
Fct20_Limits = "lo="+lim[19][1]+" hi=" + \
    lim[19][2]  # Show limit values in debugging
Fct20 = DAQC2.getADC(0, 3)  # Read X2_3, X2_6 with analog input 3 DAQC2plate
if float(lim[19][1]) < Fct20 < float(lim[19][2]):
    Fct20_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct20"+","+"X2_3 X2_6"+","+str(Fct20)+","+"PASS"+"\n")
else:
    Fct20_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct20"+","+"X2_3 X2_6"+","+str(Fct20)+","+"FAIL"+"\n")

# FCT21(K6)
Fct21_Limits = "lo="+lim[20][1]+" hi=" + \
    lim[20][2]  # Show limit values in debugging
Fct21 = DAQC2.getADC(0, 4)  # Read X2_9, X2_12 with analog input 4 DAQC2plate
if float(lim[20][1]) < Fct21 < float(lim[20][2]):
    Fct21_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct21"+","+"X2_9 X2_12_14"+","+str(Fct21)+","+"PASS"+"\n")
else:
    Fct21_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct21"+","+"X2_9 X2_12"+","+str(Fct21)+","+"FAIL"+"\n")

# FCT22(K7)
Fct22_Limits = "lo="+lim[21][1]+" hi=" + \
    lim[21][2]  # Show limit values in debugging
Fct22 = DAQC2.getDINbit(0, 4)  # Read X2_14 with digital input 4 DAQC2plate
if Fct22 == float(lim[21][2]):
    Fct22_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct22"+","+"X2_14"+","+str(Fct22)+","+"PASS"+"\n")
else:
    Fct22_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct22"+","+"X2_14"+","+str(Fct22)+","+"FAIL"+"\n")

# FCT23(K7)
Fct23_Limits = "lo="+lim[22][1]+" hi=" + \
    lim[22][2]  # Show limit values in debugging
Fct23 = DAQC2.getDINbit(0, 5)  # Read X2_16 with digital input 5 DAQC2plate
if Fct23 == float(lim[22][2]):
    Fct23_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct23"+","+"X2_16"+","+str(Fct23)+","+"PASS"+"\n")
else:
    Fct23_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct23"+","+"X2_16"+","+str(Fct23)+","+"FAIL"+"\n")

# FCT24(K8)
Fct24_Limits = "lo="+lim[23][1]+" hi=" + \
    lim[23][2]  # Show limit values in debugging
Fct24 = DAQC2.getDINbit(0, 6)  # Read X2_18 with digital input 6 DAQC2plate
if Fct24 == float(lim[23][2]):
    Fct24_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct24"+","+"X2_18"+","+str(Fct24)+","+"PASS"+"\n")
else:
    Fct24_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct14"+","+"X1_14"+","+str(Fct14)+","+"FAIL"+"\n")

# FCT25(K8)
# Variable for showing limit values in debugging
Fct25_Limits = "lo="+lim[24][1]+" hi="+lim[24][2]
```

```
Fct25 = DAQC2.getDINbit(0, 7)  # Read X2_20 with digital input 7 DAQC2plate
if Fct25 == float(lim[24][2]):
    Fct25_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct25"+","+"X2_20"+","+str(Fct25)+","+"PASS"+"\n")
else:
    Fct25_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct25"+","+"X2_20"+","+str(Fct25)+","+"FAIL"+"\n")

print("round1")

GPIO.output(12, 1)
RELAY.relayOFF(0, 7)
RELAY.relayOFF(0, 3)
sleep(0.5)

# FCT26(K1)
Fct26_Limits = "lo="+lim[25][1]+" hi=" + \
    lim[25][2]  # Show limit values in debugging
Fct26 = DAQC2.getADC(0, 1)  # Read X1_3, X1_6 with analog input 1 DAQC2plate
if float(lim[25][1]) < Fct26 < float(lim[25][2]):
    Fct26_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("\nTests for Relays K1-K8 in off-state with active high control\n\n")
        f.write("Test ID"+","+"Test point"+","+"Value"+","+"Result"+"\n")
        f.write("Fct26"+","+"X1_3 X1_6"+","+str(Fct26)+","+"PASS"+"\n")
else:
    Fct26_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("\nTests for Relays K1-K8 in off-state with active high control\n\n")
        f.write("Test ID"+","+"Test point"+","+"Value"+","+"Result"+"\n")
        f.write("Fct26"+","+"X1_3 X1_6"+","+str(Fct26)+","+"FAIL"+"\n")

# FCT27(K2)
Fct27_Limits = "lo="+lim[26][1]+" hi=" + \
    lim[26][2]  # Show limit values in debugging
Fct27 = DAQC2.getADC(0, 2)  # Read X1_9, X1_12 with analog input 2 DAQC2plate
if float(lim[26][1]) < Fct26 < float(lim[26][2]):
    Fct27_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct27"+","+"X1_9 X1_12"+","+str(Fct27)+","+"PASS"+"\n")
else:
    Fct27_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct27"+","+"X1_9 X1_12"+","+str(Fct27)+","+"FAIL"+"\n")

# FCT28(K3)
# Variable for showing limit values in debugging
Fct28_Limits = "lo="+lim[27][1]+" hi="+lim[27][2]
Fct28 = DAQC2.getDINbit(0, 0)  # Read X1_14 with digital input 0 DAQC2plate
if Fct28 == float(lim[27][2]):
    Fct28_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct28"+","+"X1_14"+","+str(Fct28)+","+"PASS"+"\n")
else:
    Fct28_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct28"+","+"X1_14"+","+str(Fct28)+","+"FAIL"+"\n")

# FCT29(K3)
# Variable for showing limit values in debugging
Fct29_Limits = "lo="+lim[28][1]+" hi="+lim[28][2]
Fct29 = DAQC2.getDINbit(0, 1)  # Read X1_16 with digital input 1 DAQC2plate
if Fct29 == float(lim[28][2]):
    Fct29_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct29"+","+"X1_16"+","+str(Fct29)+","+"PASS"+"\n")
else:
    Fct29_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
```

```
        f.write("Fct29"+","+"X1_16"+","+str(Fct29)+","+"FAIL"+"\n")

# FCT30(K4)
# Variable for showing limit values in debugging
Fct30_Limits = "lo="+lim[29][1]+" hi="+lim[29][2]
Fct30 = DAQC2.getDINbit(0, 2)  # Read X1_18 with digital input 2 DAQC2plate
if Fct30 == float(lim[29][2]):
    Fct30_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct30"+","+"X1_18"+","+str(Fct30)+","+"PASS"+"\n")
else:
    Fct30_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct30"+","+"X1_18"+","+str(Fct30)+","+"FAIL"+"\n")

# FCT31(K4)
Fct31_Limits = "lo="+lim[30][1]+" hi=" + \
    lim[30][2]  # Show limit values in debugging
Fct31 = DAQC2.getDINbit(0, 3)  # Read X1_20 with digital input 3 DAQC2plate
if Fct31 == float(lim[30][2]):
    Fct31_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct31"+","+"X1_20"+","+str(Fct31)+","+"PASS"+"\n")
else:
    Fct31_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct31"+","+"X1_20"+","+str(Fct31)+","+"FAIL"+"\n")

# FCT32(K5)
Fct32_Limits = "lo="+lim[31][1]+" hi=" + \
    lim[31][2]  # Show limit values in debugging
Fct32 = DAQC2.getADC(0, 3)  # Read X2_3, X2_6 with analog input 3 DAQC2plate
if float(lim[31][1]) < Fct32 < float(lim[32][2]):
    Fct32_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct32"+","+"X2_3 X2_6"+","+str(Fct32)+","+"PASS"+"\n")
else:
    Fct32_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct32"+","+"X2_3 X2_6"+","+str(Fct32)+","+"FAIL"+"\n")

# FCT33(K6)
Fct33_Limits = "lo="+lim[32][1]+" hi=" + \
    lim[32][2]  # Show limit values in debugging
Fct33 = DAQC2.getADC(0, 4)  # Read X2_9, X2_12 with analog input 4 DAQC2plate
if float(lim[32][1]) < Fct26 < float(lim[32][2]):
    Fct33_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct33"+","+"X2_9 X2_12"+","+str(Fct33)+","+"PASS"+"\n")
else:
    Fct33_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct33"+","+"X2_9 X2_12"+","+str(Fct33)+","+"FAIL"+"\n")

# FCT34(K7)
Fct34_Limits = "lo="+lim[33][1]+" hi=" + \
    lim[33][2]  # Show limit values in debugging
Fct34 = DAQC2.getDINbit(0, 4)  # Read X2_14 with digital input 4 DAQC2plate
if Fct34 == float(lim[33][2]):
    Fct34_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct34"+","+"X2_14"+","+str(Fct34)+","+"PASS"+"\n")
else:
    Fct34_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct34"+","+"X2_14"+","+str(Fct34)+","+"FAIL"+"\n")

# FCT35(K7)
Fct35_Limits = "lo="+lim[34][1]+" hi=" + \
    lim[34][2]  # Show limit values in debugging
Fct35 = DAQC2.getDINbit(0, 5)  # Read X2_16 with digital input 5 DAQC2plate
if Fct35 == float(lim[34][2]):
```

```
    Fct35_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct35"+","+"X2_16"+","+str(Fct35)+","+"PASS"+"\n")
else:
    Fct35_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct35"+","+"X2_16"+","+str(Fct35)+","+"FAIL"+"\n")

# FCT36(K8)
Fct36_Limits = "lo="+lim[35][1]+" hi=" + \
    lim[35][2]  # Show limit values in debugging
Fct36 = DAQC2.getDINbit(0, 6)  # Read X2_18 with digital input 6 DAQC2plate
if Fct36 == float(lim[35][2]):
    Fct36_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct36"+","+"X2_18"+","+str(Fct36)+","+"PASS"+"\n")
else:
    Fct36_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct36"+","+"X2_18"+","+str(Fct18)+","+"FAIL"+"\n")

# FCT37(K8)
Fct37_Limits = "lo="+lim[36][1]+" hi=" + \
    lim[36][2]  # Show limit values in debugging
Fct37 = DAQC2.getDINbit(0, 7)  # Read X2_20 with analog input 1 DAQC2plate
if Fct37 == float(lim[36][2]):
    Fct37_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct37"+","+"X2_20"+","+str(Fct37)+","+"PASS"+"\n\n")
else:
    Fct37_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct37"+","+"X2_20"+","+str(Fct37)+","+"FAIL"+"\n\n")

RELAY.relayON(0, 3)
sleep(0.5)

# FCT38(K1)
Fct38_Limits = "lo="+lim[37][1]+" hi=" + \
    lim[37][2]  # Show limit values in debugging
Fct38 = DAQC2.getADC(0, 1)  # Read X1_3, X1_6 with analog input 1 DAQC2plate
if float(lim[37][1]) < Fct38 < float(lim[37][2]):
    Fct38_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Tests for Relays K1-K8 in on-state with active high control\n\n")
        f.write("Test ID"+","+"Test point"+","+"Value"+","+"Result"+"\n")
        f.write("Fct38"+","+"X1_3 X1_6"+","+str(Fct38)+","+"PASS"+"\n")
else:
    Fct38_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Tests for Relays K1-K8 in on-state with active high control\n\n")
        f.write("Test ID"+","+"Test point"+","+"Value"+","+"Result"+"\n")
        f.write("Tests for Relays K1-K8 in on-state with active high control\nFct38" +
                ","+"X1_3 X1_6"+","+str(Fct38)+","+"FAIL"+"\n")

# FCT39(K2)
Fct39_Limits = "lo="+lim[38][1]+" hi=" + \
    lim[38][2]  # Show limit values in debugging
Fct39 = DAQC2.getADC(0, 2)  # Read X1_9, X1_12 with analog input 2 DAQC2plate
if float(lim[38][1]) < Fct39 < float(lim[38][2]):
    Fct39_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct39"+","+"X1_9 X1_12"+","+str(Fct39)+","+"PASS"+"\n")
else:
    Fct39_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct39"+","+"X1_9 X1_12"+","+str(Fct39)+","+"FAIL"+"\n")

# FCT40(K3)
Fct40_Limits = "lo="+lim[39][1]+" hi="+lim[39][2]  # Show limit values in debugging
Fct40 = DAQC2.getDINbit(0, 0)                     # Read X1_14 with digital input 0 DAQC2plate
if Fct40 == float(lim[39][2]):
```

```
    Fct40_Result = "PASS"                           # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct40"+","+"X1_14"+","+str(Fct40)+","+"PASS"+"\n")
else:
    Fct40_Result = "FAIL"                           # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct40"+","+"X1_14"+","+str(Fct14)+","+"FAIL"+"\n")

# FCT41(K3)
Fct41_Limits = "lo="+lim[40][1]+" hi=" + \
    lim[40][2]  # Show limit values in debugging
Fct41 = DAQC2.getDINbit(0, 1)  # Read X1_16with digital input 1 DAQC2plate
if Fct41 == float(lim[40][2]):
    Fct41_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct41"+","+"X1_16"+","+str(Fct41)+","+"PASS"+"\n")
else:
    Fct41_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct41"+","+"X1_16"+","+str(Fct41)+","+"FAIL"+"\n")

# FCT42(K4)
Fct42_Limits = "lo="+lim[41][1]+" hi=" + \
    lim[41][2]  # Show limit values in debugging
Fct42 = DAQC2.getDINbit(0, 2)  # Read X1_18 with digital input 2 DAQC2plate
if Fct42 == float(lim[41][2]):
    Fct42_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct42"+","+"X1_18"+","+str(Fct42)+","+"PASS"+"\n")
else:
    Fct42_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct42"+","+"X1_18"+","+str(Fct42)+","+"FAIL"+"\n")

# FCT43(K4)
Fct43_Limits = "lo="+lim[42][1]+" hi=" + \
    lim[42][2]  # Show limit values in debugging
Fct43 = DAQC2.getDINbit(0, 3)  # Read X1_20 with digital input 3 DAQC2plate
if Fct43 == float(lim[42][2]):
    Fct43_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct43"+","+"X1_20"+","+str(Fct43)+","+"PASS"+"\n")
else:
    Fct43_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct43"+","+"X1_20"+","+str(Fct43)+","+"FAIL"+"\n")

# FCT44(K5)
Fct44_Limits = "lo="+lim[43][1]+" hi=" + \
    lim[43][2]  # Show limit values in debugging
Fct44 = DAQC2.getADC(0, 3)  # Read X2_3, X2_6 with analog input 3 DAQC2plate
if float(lim[43][1]) < Fct44 < float(lim[43][2]):
    Fct44_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct44"+","+"X2_3 X2_6"+","+str(Fct44)+","+"PASS"+"\n")
else:
    Fct44_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct44"+","+"X2_3 X2_6"+","+str(Fct44)+","+"FAIL"+"\n")

# FCT45(K6)
Fct45_Limits = "lo="+lim[44][1]+" hi=" + \
    lim[44][2]  # Show limit values in debugging
Fct45 = DAQC2.getADC(0, 4)  # Read X2_9, X2_12 with analog input 4 DAQC2plate
if float(lim[44][1]) < Fct45 < float(lim[44][2]):
    Fct45_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct45"+","+"X2_9 X2_12"+","+str(Fct45)+","+"PASS"+"\n")
else:
    Fct45_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct45"+","+"X2_9 X2_12"+","+str(Fct45)+","+"FAIL"+"\n")
```

```
# FCT46(K7)
Fct46_Limits = "lo="+lim[45][1]+" hi=" + \
    lim[45][2]  # Show limit values in debugging
Fct46 = DAQC2.getDINbit(0, 4)  # Read X2_14 with digital input 4 DAQC2plate
if Fct46 == float(lim[45][2]):
    Fct46_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct46"+","+"X2_14"+","+str(Fct46)+","+"PASS"+"\n")
else:
    Fct46_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct46"+","+"X2_14"+","+str(Fct46)+","+"FAIL"+"\n")

# FCT47(K7)
Fct47_Limits = "lo="+lim[46][1]+" hi=" + \
    lim[46][2]  # Show limit values in debugging
Fct47 = DAQC2.getDINbit(0, 5)  # Read X2_16 with digital input 5 DAQC2plate
if Fct47 == float(lim[46][2]):
    Fct47_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct47"+","+"X2_16"+","+str(Fct47)+","+"PASS"+"\n")
else:
    Fct47_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct47"+","+"X2_16"+","+str(Fct47)+","+"FAIL"+"\n")

# FCT48(K8)
Fct48_Limits = "lo="+lim[47][1]+" hi=" + \
    lim[47][2]  # Show limit values in debugging
Fct48 = DAQC2.getDINbit(0, 6)  # Read X2_18 with analog input 1 DAQC2plate
if Fct48 == float(lim[47][2]):
    Fct48_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct48"+","+"X2_18"+","+str(Fct48)+","+"PASS"+"\n")
else:
    Fct48_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct48"+","+"X2_18"+","+str(Fct48)+","+"FAIL"+"\n")

# FCT49(K8)
Fct49_Limits = "lo="+lim[48][1]+" hi=" + \
    lim[48][2]  # Show limit values in debugging
Fct49 = DAQC2.getDINbit(0, 7)  # Read X2_20 with analog input 1 DAQC2plate
if Fct49 == float(lim[48][2]):
    Fct49_Result = "PASS"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct49"+","+"X2_20"+","+str(Fct49)+","+"PASS"+"\n")
else:
    Fct49_Result = "FAIL"  # Show test result in debugging
    with open(Report_Name, 'a') as f:
        f.write("Fct49"+","+"X2_20"+","+str(Fct49)+","+"FAIL"+"\n\n")

RELAY.relayOFF(0, 1)
RELAY.relayOFF(0, 3)
sleep(0.5)

ser.write(bytes(b'OUT0'))
sleep(0.5)
ser.write(bytes(b'ISET1:0.000'))
sleep(0.5)
ser.write(bytes(b'VSET1:0.00'))
sleep(0.5)
```