Frankfurt University of Applied Sciences

Fachbereich 2 – Studiengang Informatik

Bachelorthesis Thema

**Entwicklung einer prototypischen Software
zur Ansteuerung der GPIO-Pins
eines Raspberry Pi Einplatinencomputers
unter Windows 10**

**Eingereicht zum Erlangen des akademischen Grads
Bachelor of Science**

Autor: Daniel Heintze
Matrikel-Nummer: 1012709
Referent: Prof. Dr. Christian Baun
Korreferent: Prof. Dr. Egbert Falkenberg

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum: _____

Unterschrift: _____

Danksagung

Hiermit möchte ich mich vor allem bei meiner Familie und meinen Freunden bedanken, die mich über die gesamte Studienzeit unterstützt und begleitet haben. Auch möchte ich mich bei meinem Betreuer, Herrn Prof. Dr. Baun für die Betreuung und Bereitstellung des Themas bedanken. Auch bedanke ich mich beim Herrn Prof. Dr. Falkenberg, der sich als Korreferent zur Verfügung gestellt hat. Außerdem bedanke ich mich noch bei der MASS GmbH in Hanau und deren Mitarbeiter, die es mir ermöglicht haben an diesem Thema zu arbeiten.

Zusammenfassung

Diese Bachelorarbeit befasst sich mit dem Design und der Implementierung einer prototypischen Software zur Ansteuerung der GPIO-Pins eines Raspberry Pi 3B unter Windows 10. Darunter fällt auch die Ansteuerung des I²C-Bus, der RS232 Seriellen Schnittstelle und dem SPI-Bus, woran ein CAN-Controller angeschlossen ist. All die genannten Schnittstellen sind in einer Testumgebung verbaut und verbunden, wodurch es möglich ist, die Funktionen sofort zu überprüfen.

Abstract

This bachelor thesis deals with the design and implementation of a prototypical software, to control the GPIO pins of an Raspberry Pi 3B under Windows 10. Also it contains the control for the I²C bus, the RS232 serial interface and the SPI bus, where a CAN controller is connected. All the mentioned interfaces, are built and connected in a test environment, so it is possible to test the functions immediately.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API = Application Programming Interface

ARM = Acorn Risc Machine

CAN = Controller Area Network

CPU = Central Processing Unit

DSI = Display Serial Interface

GB = Gigabyte

GHz = Gigahertz

GPIO = General Purpose Input/Output

HDMI = High Definition Multimedia Interface

HTML = Hypertext Markup Language

I²C = Inter-Integrated Circuit

IoT = Internet of Things (Internet der Dinge)

IP = Internetprotokoll

LAN = Local Area Network

RAM = Random-Access Memory

RPi = Raspberry Pi

RS232 = Recommended Standard 232

SDK = Software Development Kit

SPI = Serial Peripheral Interface

UI = User Interface

USB = Universal Serial Bus

UWP = Universal Windows Platform

WLAN = Wireless Local Area Network

XAML = Extensible Application Markup Language

Abbildungsverzeichnis

1 Einleitung

Heutzutage werden immer mehr Einplatinencomputer wie der Raspberry Pi in der Industrie verwendet und eingesetzt. Vor allem da diese variabel für verschiedene Tätigkeiten und Einsatzfelder einsetzbar sind [1]. Die MASS GmbH spezialisiert sich im Bau von Industriecomputern mit Verwendung des Raspberry Pi. Bisher wurden diese allerdings mit dem Betriebssystem Linux ausgeliefert und programmiert. Da aber nun die Möglichkeit besteht, auch das Betriebssystem Windows 10 IoT Core zu verwenden, ist es vonnöten eine prototypische Softwarelösung für dieses zu erstellen. Dadurch soll es möglich sein alle nötigen Funktionen unter Windows 10 IoT Core effizient zu testen, um festzustellen ob diese funktionsfähig sind und mögliche Fehler sofort auszugeben. Vor allem ist die Notwendigkeit einer prototypischen Software gegeben, da bis heute noch keine allgemeine Softwarelösung existiert die alle erforderlichen Funktionalitäten vereint [2].

1.1 Zielsetzung

Das Ziel dieser Bachelorarbeit besteht darin, eine funktionsfähige und visuell übersichtliche prototypische Software zu designen und anschließend zu implementieren. Darin sollen Funktionen enthalten sein, um alle Komponenten die am Raspberry Pi angeschlossen sind, auf Funktionsfähigkeit zu überprüfen. Darunter fallen die GPIO-Pins des Raspberry Pi, die Ansteuerung des I²C-Bus, das Testen der RS232 Seriellen Schnittstelle und die Überprüfung des CAN-Controllers der am SPI-Bus angeschlossen ist. Mögliche Fehler sollen erkannt und übersichtlich erkennbar am angeschlossenen Bildschirm angezeigt werden, damit sofort erkannt wird welche Komponente defekt oder nicht ordnungsgemäß funktioniert.

Außerdem soll die prototypische Software dazu dienen, auch als Beispiel zu fungieren. Dadurch soll sich die Ansteuerung bestimmter Komponenten im Quellcode ableiten lassen, um diesen dann zu erweitern und für den dauerhaften Einsatz in der Industrie einsetzbar zu machen. Aufgrund von Auflagen bezüglich der Sicherheit in der Industrie, ist diese prototypische Software nicht dafür geeignet, diese ohne weitere Änderungen einzusetzen. Somit ist ein übersichtlicher und leicht verständlicher Quellcode nötig und auch ein Teil der Zielsetzung.

2 Stand der Technik

In diesem Kapitel sind alle verwendeten und notwendigen Technologien aufgelistet und beschrieben.

2.1 Raspberry Pi

Der Raspberry Pi ist ein Einplatinencomputer, entwickelt von der Raspberry Pi Stiftung. Der Hintergrund hierfür ist es kostengünstige und leistungsstarke Einplatinencomputer für alle Menschen, die damit arbeiten und lernen wollen, zur Verfügung zu stellen. Dies hat den Vorteil, dass jeder Zugang dazu haben kann, um den Umgang mit Technologien zu erlernen und diese einsetzen zu können. Auch befinden sich viele Hilfsmittel auf deren Webseite, um einen leichten Einstieg zu gewährleisten [3].

2.1.1 Allgemeines zum Raspberry Pi

Das Standard Betriebssystem für den Raspberry Pi ist das sogenannte Raspbian. Außerdem besteht die Möglichkeit mit dem Installationsprogramm NOOBS, welches man ebenfalls auf ihrer Webseite herunterladen kann, derzeit elf weitere unterschiedliche Betriebssysteme zu installieren. Diese sind folgende: Ubuntu MATE, Ubuntu Core, Ubuntu Server, Windows 10 IoT Core, OSMC, LibreELEC, Mozilla WebThings, PiNet, RISC OS, Weather Station und IchigoJam RPi [4]. Der Raspberry Pi Einplatinencomputer wird ständig weiterentwickelt und verbessert, um neuere Technik zu integrieren und verwendbar zu machen. Das derzeit beste und neueste Modell ist der Raspberry Pi 4 Modell B, welcher seit Juni 2019 für ungefähr 35 € erhältlich ist. Die Hauptneuerungen sind unter anderem schnellere Netzwerkverbindung, verbesserte USB Kapazität und optional mehr RAM in den Varianten: 1GB, 2GB oder 4GB [5]. Für die Inbetriebnahme eines Raspberry Pi ist noch zusätzlich eine MicroSD-Karte für das Betriebssystem und ein Micro-USB Netzteil für die Stromversorgung notwendig. Beide sind nicht im Lieferumfang enthalten und müssen einzeln erworben werden. Ein Monitor wird über den HDMI-Anschluss angeschlossen. Außerdem benötigt man eine Tastatur die per USB angeschlossen wird, um Einstellungen auf dem Raspberry Pi durchführen zu können. Eine USB-Maus kann die Arbeit erleichtern und beschleunigen, ist aber nicht zwingend notwendig, da alles über die Tastatur erreichbar ist [6].

2.1.2 Raspberry Pi 3 Modell B

Das in dieser Bachelorarbeit verwendete Modell ist das Raspberry Pi 3 Modell B. Der Hauptgrund hierfür ist, dass das Betriebssystem Windows 10 IoT Core derzeit nur bis zu dem genannten Modell kompatibel ist. Für neuere Modelle existiert gegenwärtig nur ein sogenanntes „Insider Preview“, eine unfertige Vorschau. Dort fehlen zum Teil wichtige Treiber. Es ist somit abzuraten das „Insider Preview“ zu installieren und zu verwenden. Wann eine fertige Version für neuere Modelle zur Verfügung stehen wird, ist derzeit noch nicht abzusehen [7]. Das Raspberry Pi 3 Modell B ist das erste Modell der dritten Generation, es ersetzte im Februar 2016 das ältere Raspberry Pi 2 Modell B [8]. Die wichtigen Spezifikationen, ebenfalls entnommen aus [8], sind folgende:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 WLAN und Bluetooth Low Energy (BLE) am Bord
- 100 Base Ethernet Anschluss (Maximaler Datendurchsatz von 100 Mbit/s)
- 40-pin erweiterter GPIO
- 4 USB 2.0 Anschlüsse
- HDMI-Anschluss, um ein Monitor anzuschließen
- DSI Display Port, um ein Raspberry Pi Touchscreen Bildschirm anzuschließen
- MicroSD-Anschluss, für das Betriebssystem und Daten
- Aufgerüstete, geschaltete Micro-USB-Stromquelle für bis zu 2,5A (Ampere)

Somit liefert der Raspberry Pi 3 Modell B genügend Leistung und eine vielfältige Funktionsfähigkeit, um die hier verwendete Testumgebung und das Betriebssystem Windows 10 IoT Core problemlos zu betreiben. Die Produktion soll mindestens bis zum Januar 2026 fortgeführt werden, dies garantiert das zukünftig zusammengebaute Systeme mit diesem Modell betrieben werden können [8].

Abbildung 2.1.1: Raspberry Pi 3 Modell B[1] Quelle: Raspberry Pi Foundation

1 Bildquelle: Raspberry Pi Foundation, *„Raspberry Pi 3 Model B“*, URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ (Aufgerufen am 20 .August 2019)

2.2 Windows 10 IoT Core

Das in dieser Bachelorarbeit verwendete Betriebssystem Windows 10 IoT Core, in der Version 1809 (Build 17763), ist eine Variante von Windows 10, die speziell für Einplatinencomputer mit ARM-CPUs entwickelt worden ist. Das Betriebssystem nutzt dabei die von Microsoft entwickelte Universal Windows Platform (UWP) Anwendungsprogrammierschnittstelle (API). Wie die im Namen verwendete Abkürzung IoT (Internet of Things) schon aussagt, wird es hauptsächlich dafür verwendet um verschiedene Alltagsgegenstände oder Komponenten mit dem Internet und untereinander zu vernetzen [9]. Das Betriebssystem soll mit dem von Microsoft bereitgestellten Programm Windows 10 IoT Core Dashboard installiert werden. Damit ist es möglich das Betriebssystem schnell und zuverlässig auf eine MicroSD-Karte zu installieren und mit einem Gerätenamen und Passwort zu versehen. Außerdem ermöglicht das Dashboard, nach dem der Raspberry Pi mit dem Host Computer per LAN verbunden wurde, die nötige IP Adresse auszulesen um sich direkt über einen Webbrowser in das zuvor installierte Betriebssystem einzuloggen. Dort lassen sich dann alle nötigen Einstellungen, verändern und steuern. Die Standardkonfiguration reicht aber normalerweise aus, um den Raspberry Pi mit Windows 10 IoT Core direkt verwenden zu können [10]. Ein Host Computer ist zwingend notwendig um Einstellungen zu verändern oder neue Programme zu installieren. Ein Nachteil ist, dass sich auf dem Raspberry Pi, lediglich nur zuvor installierte Programme starten lassen. Neben Windows 10 IoT Core existiert auch noch die Windows 10 IoT Enterprise Edition. Diese ist im Vergleich zu der Core Variante eine vollständige Windows 10 Version mit speziellen Features. Dafür ist die Enterprise Edition nicht mehr kompatibel zu dem ARM-CPU des Raspberry Pi und somit nicht Teil dieser Bachelorarbeit [11]. Auch hervorzuheben ist, dass es keinen Windows üblichen Explorer gibt, um sich das Dateisystem anzuschauen und installierte Daten zu durchsuchen.

2.3 Visual Studio Community Edition

Mit Visual Studio ist es möglich, die für Windows 10 IoT Core nötige UWP-App mit den Programmiersprachen C#, sowie XAML zu erstellen. Die Community Edition ist eine kostenlose alternative zu der kostenpflichtigen Visual Studio Professional Variante. Lediglich ein Microsoft Benutzerkonto ist erforderlich, um die Entwicklungsumgebung länger als 30 Tage verwenden zu können [12]. Die hier verwendete Microsoft Visual Studio Community 2019 in der Version 16.1.3 ist die aktuellste der Reihe und bietet auch eine integrierte Remote Verbindung zum Raspberry Pi, um das programmierte Programm direkt am Ziel Rechner ausführen zu können. Mitsamt einer integrierten Überwachung (Remote-Debugger), um am Zielsystem die CPU Auslastung, sowie die RAM Auslastung direkt überprüfen zu können. Dies kann nötig sein um eventuelle Schwachstellen im Programmcode ausfindig zu machen und das Programm dahingehend zu verbessern. Schließlich muss die Speicherauslastung gering gehalten werden, ansonsten kann dies zu einem instabilen System führen oder das System verlangsamen [13]. Es ist notwendig, zur Programmierung benötigte Komponenten nachzuinstallieren, da die Grundinstallation diese nicht mit installiert. In dem Fall hier, wird die „Entwicklung für die universelle Windows-Plattform“ Komponente, sowie das Windows 10 SDK für UWP benötigt.

2.3.1 Programmiersprache C#

C# auch „C Sharp“ ausgesprochen, ist eine objektorientierte Sprache wie C++ oder Java. Außerdem ist C# auch eine typsichere Programmiersprache. Wie es der Name schon vermuten lässt, entstand C# aus der gleichnamigen C-Sprachenfamilie. Der Hauptvorteil dieser Programmiersprache ist, dass sie modern und einfach zu verstehen ist. Dadurch ist diese im Vergleich zu anderen Programmiersprachen übersichtlich gehalten und lässt sich fließend lesen. Sie umfasst auch die Unterstützung für eine komponentenorientierte Programmierung. Damit ist gemeint, dass sogenannte Funktionspakete bereit stehen, die bereits wiederverwendbare Komponenten bereit hält. Diese bestehen aus Eigenschaften, Methoden, sowie Ereignissen. Außerdem enthalten diese spezifische Attribute, die eine eigene Dokumentation enthalten. Dadurch soll es möglich sein, weniger Code selbständig neu programmieren und erstellen zu müssen. Somit wird C# zu einer sehr natürlichen Sprache, die nicht wie andere kryptisch daherkommen. C# enthält auch mehrere Funktionen, die bei der Entwickelung der Anwendung behilflich sein können, um zum Beispiel die Stabilität zu erhöhen. Unter anderem ist eine sogenannte „Garbage Collection“ integriert, die automatisch Arbeitsspeicher von nicht verwendeten Objekten freigibt. Außerdem bietet die Ausnahmebehandlung eine strukturierte Fehlererkennung und damit einen erweiterten Ansatz zur Wiederherstellung. Ebenfalls verfügt die Programmiersprache über ein

einheitliches Typsystem. Dies bedeutet, dass alle C#-Typen von einem einzelnen „Objekt-Stammtyp“ erben, darunter auch primitive Typen wie „int“. Dadurch teilen sich alle Typen gemeinsame Operationen und Werte jeglicher Art können gespeichert, transportiert und konsequent bearbeitet werden. Außerdem wird eine dynamische Zuordnung von Objekten, durch benutzerdefinierte Verweistypen und Wertetypen ermöglicht. Ebenfalls hervorzuheben ist die Versionsverwaltung von C#, auf die viel Wert gelegt wird. Dadurch wird sichergestellt, dass alle C#-Programme, sowie Bibliotheken kompatibel bleiben [14].

2.3.2 Beschreibungssprache XAML

XAML oder auch „Extensible Application Markup Language“, ist eine Markupsprache, beziehungsweise auch Beschreibungssprache. Somit ist diese ähnlich von der Struktur her aufgebaut wie HTML, welches ebenso eine Beschreibungssprache ist. XAML vereinfacht die Entwicklung einer UI (User Interface) für eine „.NET Framework“ Anwendung. In dieser Bachelorarbeit wird XAML dazu verwendet, die gesamte Benutzeroberfläche zu designen und zu implementieren. Dies funktioniert zusammen mit C#, welches problemlos auf die Elemente die in XAML definiert werden, zugreifen kann. Gemeinsam mit C# lassen sich somit komplizierte visuelle Anwendungen programmieren. Dadurch unterscheidet sich XAML zu anderen Beschreibungssprachen, die nicht direkt mit einer Programmiersprache verknüpft sind. Außerdem lässt XAML zu, dass mehrere Personen an der gleichen UI und Logik gemeinsam arbeiten und das sogar mit unterschiedlichen Werkzeugen. Durch die Übersichtlichkeit der Sprache und der zuvor genannten Verknüpfung mit C#, eignet sie sich besonders für eine prototypische Softwarelösung [15].

3 Aufbau der Testumgebung

Im folgenden Kapitel wird die gesamte, von der MASS GmbH zur Verfügung gestellte, Testumgebung beschrieben und gezeigt. Die Testumgebung wurde speziell dafür hergestellt, alle benötigten Funktionen direkt testen zu können. Durch die modulare Bauweise ist es möglich jederzeit die Testumgebung, um weitere Funktionen zu erweitern oder eingebaute Komponenten auszutauschen.

3.1 Allgemeines über die Testumgebung

Hauptsächlicher Bestandteil der Testumgebung ist eine Platte aus Metall, an der alle Anschlüsse eingebaut sind, sowie all die zum Test notwendigen Leuchtdioden und Kippschalter. Zur besseren Standfestigkeit wurde ein Metallblock als Bodenplatte verwendet, dadurch ist genügend Gewicht vorhanden, welches die Testumgebung an Ort und Stelle festhält. An der Vorderseite zu erwähnen sind noch die beiden Spannungs-Kontrollleuchten, eine in der Farbe Grün für 3,3 Volt und eine Rote für 5 Volt. Diese zeigen an, ob jene Spannung auch wirklich anliegt. Falls diese nicht aufleuchten, würde die Spannungsversorgung nicht ordnungsgemäß funktionieren. Außerdem ist vorne ein Schalter für die Stromversorgung eingebaut, um die gesamte Testumgebung ein-, sowie auszuschalten. Siehe Abbildung 3.1.1 für die Vorderansicht.

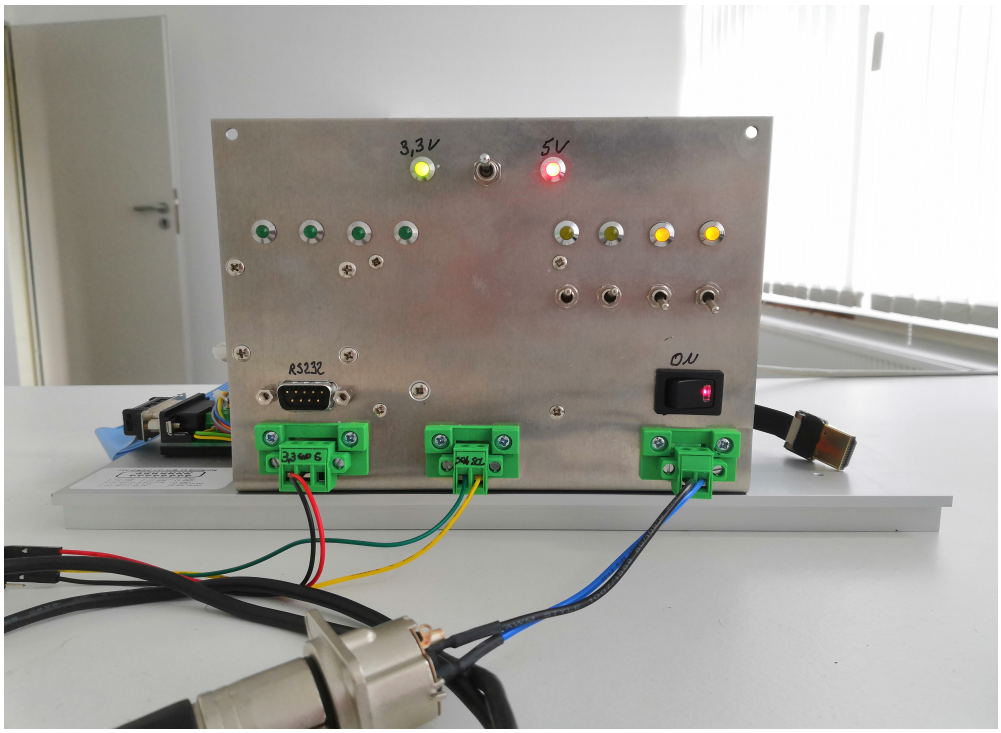

Abbildung 3.1.1: Vorderansicht der Testumgebung, Quelle: eigenes Bild

Direkt hinter der Platte ist ein DC-DC-Wandler angebracht, auch Gleichspannungswandler genannt. Dieser wandelt einen Spannungseingang von 18 bis maximal 36 Volt Gleichstrom, in eine Ausgangsspannung von 5 Volt und einer Stromstärke von 3 Ampere um. Die gesamte Testumgebung ist darauf angewiesen, dass die Spannung von 24 Volt auf 5 Volt runter transferiert wird, um die notwendige Betriebsspannung für den Raspberry Pi zu gewährleisten. Direkt auf der Rückseite der Platte ist der Raspberry Pi 3 Modell B montiert. Auf den 40 GPIO-Pins des Raspberry Pi ist eine weitere Platine aufgesteckt. Welche dann alle Pin Belegungen, auf alle Inputs und Outputs, sowie Busse verteilt. In der Draufsicht, siehe Abbildung 3.1.2, sind all die zuvor genannten Komponenten sichtbar.

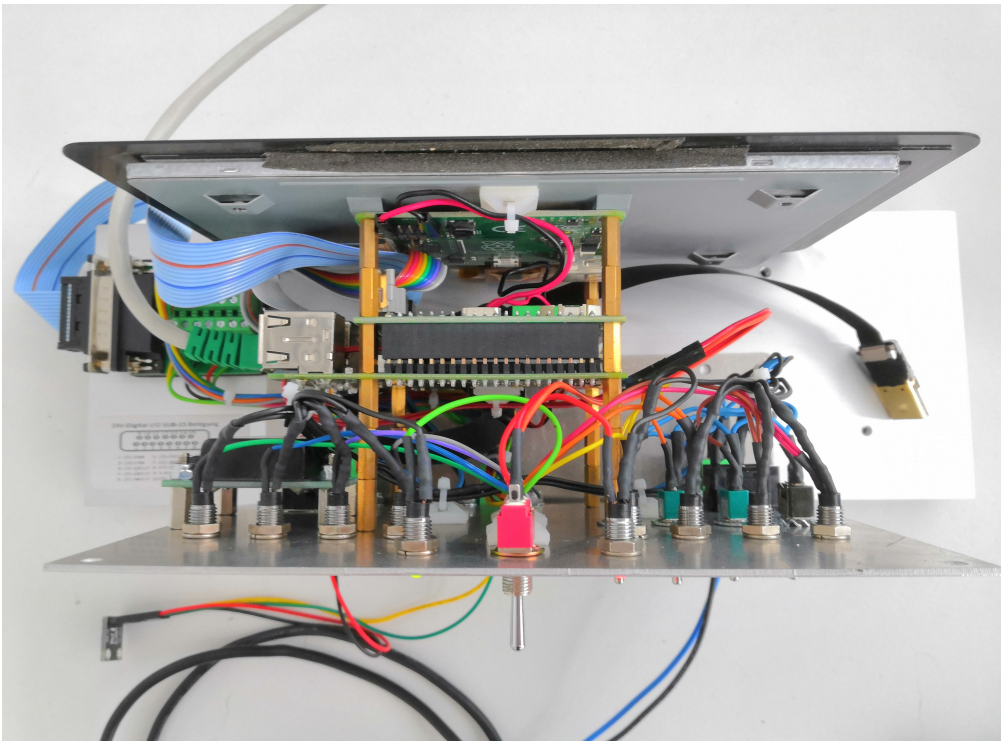

Abbildung 3.1.2: Draufsicht der Testumgebung, Quelle: eigenes Bild

Außerdem ist hinten an der Testumgebung ein „Raspberry Pi 7 Zoll Touchscreen Display“ montiert. Dieser ist über eine Erweiterungsplatine direkt mit der Netzteilplatine, sowie dem Raspberry Pi verbunden. Das Bildsignal wird über eine zusätzliche Adapterplatte konvertiert. Die maximale Auflösung des Bildschirms beträgt 800 x 480 Pixel [16]. Über den Touchscreen ist es möglich den Raspberry Pi zu bedienen, ohne dass eine Tastatur oder eine Maus angeschlossen ist. Außerdem simuliert es zeitgleich den Einsatz in der Industrie, da fertige Industriecomputer in der Regel mit einem Touchscreen Display ausgeliefert werden. Es ist aber auch ein HDMI-Anschluss vorhanden, falls man an der Testumgebung einen Monitor anschließen möchte. Dazu müsste man aber die Verbindung zum Touchscreen Display unterbrechen. In der Abbildung 3.1.3 ist zu sehen wie das Touchscreen Display im eingeschalteten Zustand aussieht.

Abbildung 3.1.3: Rückansicht der Testumgebung, Quelle: eigenes Bild

3.2 GPIO

GPIO oder auch General Purpose Input Output genannt, sind programmierbare Ein- und Ausgänge die für verschiedene Funktionen verwendet werden können. Im Falle des Raspberry Pi ist die GPIO in Form einer Stiftleiste herausgeführt. Diese dienen dann als Schnittstelle für verschiedene Einsatzfelder. Somit kann man mit dem Raspberry Pi, digitale Signale von außen erkennen und verarbeiten, diese werden auch Input genannt. Auch eigens erstellte Signale können nach außen senden, welche dann als Output bekannt sind [17]. Die GPIO-Pins werden von der GPIO-Platine an die Frontplatte der Testumgebung ausgeführt. An der befinden sich vier Grüne Leuchtdioden für die Outputs und insgesamt vier weitere Gelbe Leuchtdioden für die Inputs, wo insgesamt vier Kippschalter vorgeschaltet sind, um mit denen die Inputs zu steuern. Auf der GPIO-Platine befinden sich ebenfalls vier Optokoppler mit einer maximalen Stromstärke von 50 Milliampere pro Port. Diese wandeln hauptsächlich Signale von 24 Volt auf 3,3 Volt runter. Darüber hinaus sind noch zwei Photomos mit jeweils zwei Kanälen auf der GPIO-Platine angebracht. Diese vertragen jeweils eine maximale Stromstärke von bis zu 500 Milliampere und wandeln Signale von 3,3 Volt auf 24 Volt um.

3.3 I²C-Bus

I²C oder auch „I Quadrat C“ ausgesprochen, ist ein serieller Zweidraht-Bus der synchron läuft. Dieser kann jeweils Daten und den Takt in beide Richtungen auf einer Leitung versenden. Der Bus ist außerdem nur für die Kommunikation über kleinere Distanzen geeignet. Aus lizenzrechtlichen Gründen wird der Bus auch TWI (two wire interface) genannt, da der I²C-Bus von dem Hersteller Philips in den 80er Jahren entwickelt worden ist [18]. Der Bus enthält maximal 128 Adressen, wovon mindestens einer der Adressen der sogenannte Master ist und die anderen 127 Adressen die sogenannten Slaves sind. Es ist aber auch möglich den Bus als sogenannten „Multi-Master-Bus“ zu betreiben, wodurch mehrere Master ermöglicht werden. Jeder Slave enthält eine eigene, einzigartige codierte Adresse. Dadurch kann jeder Slave individuell von einem Master angesprochen werden. Außerdem existiert noch ein sogenannter Broadcast-Kanal mit dem es möglich ist, alle angeschlossenen Slaves gleichzeitig anzusprechen. Der eingeschränkte Adressbereich kann ein Problem darstellen, da dieser sich auf einem bestimmten definierten Bereich befindet, dadurch können Slaves indirekt andere Adressen ausschließen und blockieren [18]. In der Testumgebung wird der I²C-Bus auf eine 3-polige Phoenix-Buchse an die Front raus geführt, um die Möglichkeit zu besitzen, noch zusätzliche Slave-Geräte anzuschließen und mehrere Adressen im Bus finden zu können. Auf der Adresse „0x68“, ist eine sogenannte „Real Time Clock“ (Echtzeituhr) auch RTC abgekürzt angebracht. Die genaue Typenbezeichnung des Moduls ist DS3231 und dieser ist direkt auf der GPIO-Platine angebracht. Vorne an der Phoenix Buchse ist zusätzlich ein Drucksensor zum Zweck der Findung mehrerer Adressen auf dem I²C-Bus angeschlossen. Diese Adresse ist Unbekannt und soll nachher durch den Programmcode gefunden und angezeigt werden.

3.4 RS232 Serielle Schnittstelle

Die Serielle Schnittstelle namens RS232, dient dazu Daten zwischen dem Raspberry Pi und einem an der Schnittstelle angeschlossenen Peripheriegerät zu übertragen. Die Datenübertragung erfolgt seriell, das heißt die Daten werden nacheinander und nicht zur selben Zeit übertragen. Die Serielle Schnittstelle wurde mittlerweile durch den USB (Universal Serial Bus) ersetzt, findet aber immer noch durch seine Einfachheit und leichten Implementierbarkeit Verwendung, vor allem in der Industrie zur Maschinensteuerung [19]. Der UART (Universal Asynchronous Receiver Transmitter) RS232-Wandler wird in der Testumgebung von der GPIO-Platine zu einer DB9 9-poligen Buchse, die an der Front Platte angebracht ist, raus geführt. An dieser ist es möglich mit einem Dongle (Computerperipheriegerät), in dem eine Schleife eingebaut ist welches hauptsächlich aus einem Draht besteht, den sogenannten Schleifentest durchzuführen. Mehr dazu im Kapitel 4.2.4.

3.5 CAN-Controller

Der CAN-Controller MCP2515 ist für den CAN-Bus (Controller Area Network) zuständig und ist mit dem Raspberry Pi über den SPI-Bus (Serial Peripherial Interface) verbunden. Der SPI-Bus besteht aus drei Leitungen und überträgt die Daten seriell synchron. Die drei Leitungen lassen sich in folgende Aufgabenbereiche unterteilen:

- Die Übertragung vom Master zum Slave, auch MOSI (Master Out → Slave In) abgekürzt.
- Die Übertragung vom Slave zum Master, auch MISO (Master In ← Slave Out) abgekürzt.
- Sowie der SCK (Serial Clock) für den Takt der Übertragung.

Außerdem wird zusätzlich noch für jeden Slave eine weitere Leitung benötigt, die man Slave Select (SS) oder auch Chip Select (CS) nennt. Dadurch kann der Master den richtigen Slave zur Kommunikation selektieren, dafür muss der Master lediglich die zugehörige Leitung von High auf Low stellen. Zusätzlich bekommt der Slave vom Master mitgeteilt, dass eine Nachricht beginnt und das darauffolgende Byte als Kommando verarbeiten soll [20].

Der CAN-Bus verbindet mehrere Komponenten die meistens Knoten genannt werden, über eine 2-Draht Leitung miteinander. Das CAN-Protokoll wurde von dem Unternehmen Bosch im Jahre 1983 für den Einsatz in Kraftfahrzeugen entwickelt. Durch die hohe Störsicherheit und der Echtzeitfähigkeit ist der CAN-Bus auch in der Automatisierungstechnik von Interesse. Außerdem wird das CAN-Protokoll weiterhin von der Organisation „Can in Automation“ (CiA) weiterentwickelt [21].

In der Testumgebung ist wie zuvor erwähnt, der CAN-Controller MCP2515 verbaut und direkt mit dem Raspberry Pi über den SPI-Bus verbunden. Zu Testzwecken wurde die Stromversorgung vom Controller, durch löten und einer neuen Leitung zu einem Kippschalter umgeleitet, der an der Frontplatte der Testumgebung angebracht ist. Damit ist es möglich die Spannungsversorgung für den MCP2515 zu simulieren, ob er bestückt oder nicht bestückt ist und das ohne ihn jedes Mal zu entfernen oder zu installieren.

4 Design der Lösung

In diesem Kapitel ist die Vorgehensweise für das Konzept mit den möglichen Lösungsansätzen beschrieben. Hier sind die benötigten Komponenten erwähnt, die in dieser Bachelorarbeit verwendet wurden.

4.1 Einrichtung aller benötigten Komponenten

Zuallererst ist es notwendig das Betriebssystem Windows 10 IoT Core auf eine MicroSD-Karte zu installieren. Dazu muss das Windows 10 IoT Core Dashboard heruntergeladen und auf dem Host Computer installiert werden. Benötigt wird eine MicroSD-Karte mit mindestens 16GB Speicherplatz. Es sollte darauf geachtet werden, dass die MicroSD-Karte nicht einer alten Generation entspricht, da dies Auswirkungen auf die Leistung und Qualität des Betriebssystems haben kann. Empfohlene Speichermedien lassen sich aus einer offiziellen Liste entnehmen, siehe dazu Quelle [22]. Der Host Computer benötigt zu dem ein Kartenlesegerät für MicroSD-Karten, worin die zu installierende MicroSD-Karte eingesteckt ist. Nachdem alles vorbereitet ist wird per Dashboard das Betriebssystem mit folgenden Einstellungen installiert:

- Gerätetyp: Raspberry Pi 2 & 3
- Betriebssystem: Windows 10 IoT Core (17763)
- Laufwerk: Der Laufwerksbuchstabe der MicroSD-Karte
- Gerätename: minwinpc (Hier ist jeder beliebige Name zulässig)
- Passwort: p@ssw0rd (Beliebiges Passwort ist möglich)
- WLAN ist deaktiviert, kann falls nötig aktiviert werden

Anschließend wenn die Installation des Betriebssystems vollständig abgeschlossen ist, ist es notwendig die MicroSD-Karte ordnungsgemäß vom Host Computer zu trennen. Daraufhin wird die MicroSD-Karte in den Raspberry Pi eingesetzt. Nun muss die Testumgebung in Betrieb genommen werden, in dem sie zuallererst per LAN-Leitung mit dem Host Computer verbunden wird. Die Stromversorgung der Testumgebung wird mit einer Leitung an der Buchse befestigt, sowie mit einer Steckdose verbunden. Wenn alles ordnungsgemäß verbunden ist kann mit dem Ein- und Ausschalter, die Testumgebung eingeschaltet werden. Nach einer gewissen Zeit erscheint nun der Raspberry Pi im Dashboard, mitsamt seiner IP-Adresse. Damit ist es nun möglich, mit einem Internetbrowser auf diesen zuzugreifen und sich mit den zuvor angegebenen Passwort einzuloggen. Der Standardbenutzername der vergeben wird, ist der „Administrator“. Es ist nicht notwendig Einstellungen im Raspberry Pi vorzunehmen, da bereits alles fertig eingestellt ist. Die Remoteverbindung, die funktionieren sollte, ist

nicht in der vorliegenden Version funktionsfähig. Somit ist es nicht möglich, von außen heraus, den Bildschirm vom Raspberry Pi anzuzeigen. Ob dies in der Zukunft möglich sein wird ist ungewiss. Damit ist der Raspberry Pi und die Testumgebung fertig eingerichtet.

Am Host Computer selbst ist Visual Studio Community 2019 essentiell, um die Programmierung durchzuführen. Darüber hinaus ist das Windows 10 SDK für UWP zu installieren. Nachdem beides installiert ist, ist noch ein Benutzerkonto bei Microsoft erforderlich, um Visual Studio länger als 30 Tage benutzen zu können. Damit wäre alles vorbereitet und startbereit.

4.2 Aufbau eines Konzepts

In diesem Unterkapitel wird näher auf das Konzept und der Wahl der Komponenten eingegangen.

4.2.1 Auswahl des Raspberry Pi

Wie schon zuvor erwähnt, ist es wegen Windows 10 IoT Core nicht möglich eine andere Version des Raspberry Pi zu verwenden. Da die derzeitige Version des Betriebssystems nur den Raspberry Pi 3 Modell B unterstützt und die Insider Version für das nächst bessere Modell des Raspberry Pi, derzeit unvollständig ist und nicht alle Treiber dafür vorhanden sind. Somit ist die Entscheidung fest getroffen und es würde keinen Sinn ergeben die Insider Version zu verwenden. Falls zukünftig eine neue Version erscheinen sollte, sollte es eine Möglichkeit geben denn Programmcode auch dafür zu compilieren.

4.2.2 Konzeption der GPIO-Schnittstelle

Die Zielsetzung in diesem Teil der Arbeit, besteht darin, die Outputs und die Inputs programmtechnisch ansteuern zu können. Das heißt, auf der Benutzeroberfläche soll es möglich sein, die Outputs mit sogenannten Buttons schalten zu können. Während die Inputs mit physischen Kippschaltern an der Frontplatte der Testumgebung betätigt werden und auf der Benutzeroberfläche sollen diese Zustände abgelesen und angezeigt sein. Zusätzlich soll es möglich sein, die Inputs mit den Outputs zu verknüpfen, damit die Änderungen an den Zuständen der Inputs, ebenso die Outputs steuern. Dies dient hauptsächlich Testzwecken, um zu sehen ob die Zustände problemlos weitergegeben werden können. Außerdem ist es nötig eine Invertierung der Inputs hinzuzufügen. Wenn die Invertierung eingeschaltet ist, sollen die Anzeigen für die Inputs auf der Benutzeroberfläche, das Gegenteil ausgeben. Ein Kippschalter im eingeschalteten Zustand soll auf der Benutzeroberfläche den Zustand ausgeschaltet haben. Während ein Kippschalter der ausgeschaltet wäre, in der Benutzeroberfläche wiederum eingeschaltet angezeigt wäre. Dies hat den Hintergrund, dass es Maschinen in der Industrie gibt die

als Input Signale verarbeiten. Diese Signale sagen aus, dass die Maschine funktionsfähig ist und wenn der Input auf den Zustand ausgeschaltet fällt, ein Alarm ausgelöst werden soll. Somit sind die Inputs in dem Fall immer eingeschaltet und sobald das Signal nicht mehr anliegt, soll dies sofort sichtbar sein. Auch ist es notwendig anzuzeigen, das die GPIO Schnittstelle erfolgreich initialisiert ist, beziehungsweise bei einem Fehler, eine entsprechende Fehlermeldung ausgegeben wird. Darüber hinaus soll der Quellcode übersichtlich aufgebaut sein, so dass es in der Zukunft möglich ist diesen zu verstehen und gegebenenfalls mit weiteren Funktionen zu erweitern. Es ist ebenfalls notwendig, am Ende des Tests alle Inputs und Outputs auf ihren Nominal Zustand zurückzusetzen.

4.2.3 Konzeption der Findung von I²C-Adressen

Die Zielstellung hier, besteht hauptsächlich auf der Findung von allen Adressen auf dem sogenannten I²C-Bus. Der Programmcode soll in diesem Fall, den Adressbereich von „0x03“ bis „0x77“ durchsuchen und jede gefundene Adresse in einer gesamt Liste ausgeben. Adressen woran kein I²C Gerät angeschlossen ist, sollen nicht ausgegeben werden. Dies ist notwendig, da man nicht immer weiß, welche Adresse ein angeschlossenes Gerät hat. Es ist aber nicht möglich die genaue Bezeichnung des Geräts auszulesen, da diese nicht darauf gespeichert wird. Außerdem soll die Echtzeituhr, nach dem alle Geräteadressen gefunden sind, ausgelesen und synchronisierbar sein. Das heißt mit einem Button soll es ermöglicht werden, die Zeit mit der Systemzeit des Raspberry Pi abzugleichen. In dem Fall hier, ist kein auffangen von Fehlern notwendig. Da es nur nötig ist, die Adressen zu finden und der I²C-Bus normalerweise immer vorhanden ist.

4.2.4 Konzeption des Schleifentests für den RS232

Das Ziel hierbei ist es, zuallererst eine Verbindung mit der Seriellen Schnittstelle herzustellen und diese zu initialisieren. Nach dem diese initialisiert ist, soll für mehrere Zyklen eine Nachricht versendet werden, diese wird wiederum zurück empfangen und auf der Benutzeroberfläche ausgegeben. Hierfür wird ein sogenannter Dongle an der RS232 Schnittstelle angeschlossen, um zu testen ob die Serielle Schnittstelle die Nachrichten versenden kann. In diesem Dongle ist lediglich eine Schleife eingebaut und hat somit keinerlei Funktion, außer die Serielle Schnittstelle zu testen. Falls ein Fehler auftritt, soll diese wiederum auf der Benutzeroberfläche ausgegeben werden. Außerdem wenn der Dongle nicht angeschlossen ist, soll dies erkannt werden und ebenfalls eine Fehlermeldung erscheinen. Ansonsten könnte dies problematisch sein, wenn keine Bedingung vorhanden ist, den Test abzuschließen falls der Dongle nicht angeschlossen ist. Dazu muss ein Timer (Zeitmesser) im Programmcode eingebaut sein, der nach einer bestimmten Zeit den Test automatisch abbricht und somit anzeigt das der Dongle nicht gefunden wurde.

4.2.5 Konzeption zur Initialisierung des CAN-Controllers

Um den CAN-Controller ansteuern und initialisieren zu können ist es erforderlich den SPI-Bus zu initialisieren. Das Ziel hierbei ist, ein Test der den Controller mit den richtigen Parametern initialisiert und überprüft dass dieser existiert. Falls der Controller erfolgreich gefunden und initialisiert worden ist, soll dies als Meldung auf der Benutzeroberfläche angezeigt werden. Außerdem ist es nötig, falls der CAN-Controller nicht existiert oder initialisiert werden kann, dies ebenfalls als Fehlermeldung auszugeben. Der CAN-Controller kann über einen Kippschalter Ein- und Ausgeschaltet werden, somit ist ein Test dieser beiden Zustände möglich. Der Test soll wiederholbar aufgebaut sein, damit man nicht die Anwendung neu starten muss.

4.3 Mögliche Lösungen der Benutzeroberfläche

Das Hauptziel der Benutzeroberfläche ist diese so übersichtlich wie nur möglich zu halten und dass die Anwendung nicht zu überladen wirkt. Es sollen alle Tests enthalten und direkt auswählbar sein. Es ist außerdem wichtig, dass die Anwendung geschlossen werden kann, aber es ist nicht notwendig Daten zu speichern. Dadurch wird die Anwendung bei jedem neu Start, auf den Anfangszustand zurückgesetzt. Ansonsten hat die Zielsetzung für die Benutzeroberfläche keine weiteren Bedingungen.

Die erste Überlegung ist es, die gesamte Anwendung in einzelne Abschnitte zu unterteilen. Das heißt, dass jeder Test als solches, Eigenständig agiert. Dadurch lässt sich sicherstellen, dass die einzelnen Abschnitte sich nicht untereinander stören. Also müssen beim wechseln der Tests, alle Variablen zurückgesetzt sein und vom Arbeitsspeicher freigegeben werden. Dies ist vor allem deswegen wichtig, da nicht freigegebener Arbeitsspeicher sich ansammeln und dadurch die Anwendung beeinträchtigen könnte. Im schlimmsten Fall könnte die Anwendung dadurch sogar abstürzen. Wenn die Anwendung gestartet wird, soll erst einmal ein Startbildschirm angezeigt werden, von dort aus sollen alle Tests erreichbar sein. Es gibt mehrere Möglichkeiten ein Hauptmenü zu designen. Die erste ist Möglichkeit ist es, am Anfang ein Menü anzuzeigen welches in der Mitte alle möglichen Tests anzeigt. Wenn man ein Test auswählt, soll das Menü verlassen werden und die Seite des gewählten Tests stattdessen geladen und angezeigt werden. Dort ist es dann möglich mit einem Button wieder zurück zum Hauptmenü zu kommen. Dies hat den Nachtteil, dass man jedes Mal wieder zurück zum Hauptmenü muss, um einen anderen Test starten zu können. Dafür ist der Vorteil, dass alle Tests ordnungsgemäß gestartet und beendet werden können. In dem Fall ist dies nicht die optimale Lösung, da diese nicht Benutzerfreundlich ist und unnötig viele Betätigungen vom Benutzer erfordert.

Deswegen ist es vorteilhafter wenn es ein Hauptfenster gibt, welcher alle Buttons für jeden einzelnen Test enthält. Das bedeutet, dass es ein kleineres Fenster gibt, welches dann den angeforderten Test ausgibt. Dies hat den Vorteil, dass egal in welchem Test man sich befindet, jederzeit den nächsten Test aufrufen kann, ohne wiederum ins Hauptmenü zurück gehen zu müssen. Der Nachteil hierbei ist den Programmcode so abzusichern, dass es nicht zu unvorhergesehenen Fehlern kommt. Dazu muss bei einem Wechsel jegliche zuvor benutzte Variable und Verbindung geschlossen, sowie vom Arbeitsspeicher freigegeben werden. Dies erfordert einen erhöhten Aufwand in der Programmierung, ist aber dennoch die bessere Lösung, da der Benutzer somit alles auf einem Blick hat und effizient alle Tests nacheinander durchführen kann.

Ebenso wird Wert darauf gelegt die Benutzeroberfläche mit Grafiken zu versehen. Dazu wurden verschiedene Grafiken von der MASS GmbH erstellt und zur Verfügung gestellt. Mit diesen Grafiken soll die Anwendung auch visuell ansprechend gestaltet sein. Dadurch erhöht sich die Übersichtlichkeit der Anwendung, wenn die Grafiken sich voneinander sowohl farblich, als auch geometrisch unterscheiden. Der Unterschied zwischen einem Input und Output soll damit klarer erkennbar sein.

5 Implementierung

In diesem Kapitel wird der Hauptteil der Bachelorarbeit behandelt. Hier wird beschrieben wie alles implementiert wurde und alle endgültigen Entscheidungen aufgezeigt. Außerdem sind hier ebenso Bildschirmaufnahmen der fertigen Anwendung zu sehen.

5.1 Erstellung der Anwendung in Visual Studio

Zuallererst muss die Anwendung erstellt werden. Dazu muss im geöffneten Visual Studio ein neues Projekt erstellt werden. Dort gibt es vielerlei Möglichkeiten und verschiedene Programmiersprachen zur Auswahl. Hier wird aber die „Leere App UWP“ (Universelle Windows-App) ausgewählt, die als Programmiersprache C#, sowie als Beschreibungssprache XAML verwendet. Nach der Auswahl wird ein Name für das Projekt angegeben. In diesem Fall wird das Projekt „MASS_Platinen_Test“ genannt. Daraufhin ist die Erstellung des Projekts abgeschlossen. Rechts in der Menüführung des Projekts öffnet man nun die „Package.appxmanifest“ Datei und verändert am Ende den Code zu:

```
<Capabilities>
  <DeviceCapability Name="serialcommunication">
    <Device Id="any">
      <Function Type="name:serialPort" />
    </Device>
  </DeviceCapability>
</Capabilities>
```

Dies ist notwendig damit die Serielle Schnittstelle später angesprochen werden kann. Ansonsten führt es zu Fehlern und es ist nicht möglich die Serielle Schnittstelle zu verwenden [23].

Außerdem wird beim Debugger auf die ARM-Plattform umgestellt, da hier nur für jene Plattform programmiert wird. Zusätzlich wird der Zielcomputer zum testen des Codes auf den Remotecomputer umgestellt. Dazu muss die IP-Adresse des Raspberry Pi anhand des Dashboards ermittelt und dort eingefügt werden. Somit sind die Vorbereitungen abgeschlossen und es kann nun am Quellcode gearbeitet werden. Diese Vorgänge sind nur notwendig wenn eine neue Anwendung erstellt wird, falls der fertige Programmcode geladen wird, ist dieser Schritt nicht notwendig. Die Umstellung des Zielcomputers beim Debugger ist aber notwendig, damit diese Anwendung auf der richtigen Plattform getestet wird und nicht lokal im Host Computer.

5.2 Ansteuerung der GPIO

Der wichtigste Teil der Implementierung ist das ansteuern der GPIO. Die Hauptquellen [24] und [25] wurden hierfür als Vorlage, sowie zur Inspiration verwendet. Da der Umfang des Programmcodes groß ausfällt, werden hier nur wichtige Teile beschrieben und kleinere Ausschnitte gezeigt. Für den vollständigen Programmcode, siehe Anhang ab Seite 39 bis Seite 54 für den C#-Code und ab Seite 55 bis Seite 62 für den XAML-Code. Zuallererst ist es notwendig nötige Klassenbibliotheken zu inkludieren,d a ansonsten nicht alle Funktionen verwendet werden können. Diese sind wie folgt:

```
using Windows.Devices.Gpio;
using Windows.UI.Core;
using Windows.UI.Xaml.Media.Imaging;
```

Auch wichtig zu notieren ist es, dass die Inputs die Nummerierungen: 16, 17, 26 und 27 haben, sowie die Outputs die Nummerierungen: 22, 23, 24 und 25 besitzen. Anhand der Nummern ist es möglich diese direkt anzusprechen. Als erstes deklariert man die nötigen Variablen, als Beispiel ein Ausschnitt:

```
private GpioPin OutPin22;
private const int LED_22 = 22;
```

Durch die erste Zeile wird eine Variable erstellt, die nachher die Daten des geöffneten Pins enthält. Während die zweite Zeile lediglich dazu dient, die Nummer des Pins zu speichern. Diese zwei Zeilen werden für alle weiteren Outputs, sowie Inputs wiederholt und an deren Nummerierungen angepasst.

5.2.1 Methode InitGPIO()

Diese Methode ist dafür zuständig alles nötige zu initialisieren und auf einen Anfangszustand zu setzen. Mit folgender Zeile „var gpio = GpioController.GetDefault();“, wird erst einmal der zuständige GPIO-Controller geladen. Anschließend kann man direkt mit „if (gpio == null)“ prüfen, ob der Controller gefunden ist. Denn falls der Wert „null“ ergibt, existiert kein Controller womit die GPIO angesteuert werden kann. Jetzt können alle Variablen, wie folgt initialisiert werden:

```
OutPin22 = gpio.OpenPin(LED_22);
OutPin22.Write(GpioPinValue.Low);
OutPin22.SetDriveMode(GpioPinDriveMode.Output);
```

Wie zuvor erwähnt bekommt die Variable „OutPin22“, nun mit der Anweisung alle nötigen Daten geliefert und speichert diese. Somit ist der Pin 22 geöffnet und in der zugehörigen Variable gespeichert. Dadurch ist es möglich in der zweiten Zeile dem Pin 22 den Zustand „low“ zu übergeben, damit dieser auf den ausgeschalteten Zustand gebracht wird. Anschließend setzen wir den Modus des Pins auf Output, da dieser Pin

Zustände ausgeben soll. Diese drei Codezeilen werden für alle drei weiteren Outputs wiederholt. Bei den Inputs wird lediglich nur die erste Zeile vom vorherigen Ausschnitt benutzt. Denn für diese ist es notwendig einen anderen Modus anhand dieser Zeilen einzustellen:

```
if (InPin16.IsDriveModeSupported(GpioPinDriveMode.InputPullDown))
        InPin16.SetDriveMode(GpioPinDriveMode.InputPullDown);
else    InPin16.SetDriveMode(GpioPinDriveMode.Input);
```

Hier wird erst abgefragt ob der zugehörige Pin den speziellen Modus „InputPullDown“ unterstützt. Wenn dies der Fall ist wird der Modus auf „InputPullDown“ gestellt, ansonsten wird dieser nur auf den Standard „Input“ gesetzt. Dieser spezielle Modus ist notwendig, da es ansonsten eine Verzögerung gibt, wodurch bei Betätigung eines Kippschalters die Anwendung eine Veränderung des Zustands nicht direkt registriert. Bei den Inputs ist es nicht möglich den Zustand selbst zu setzen, da hierfür die Zustände ausgelesen werden müssen. Dafür sind folgende Zeilen notwendig:

```
if (InPin16.Read() == GpioPinValue.High) {
        In16Image.Source = ChangeImageSource(diodeOn);
        In16Status.Text = "Input 16 ON";  }
else {
        In16Image.Source = ChangeImageSource(diodeOff);
        In16Status.Text = "Input 16 OFF"; }
```

In den Zeilen wird der derzeitige Zustand des Inputs ausgelesen. Danach wird verglichen ob der ausgelesene Wert auf „high“ steht, also eingeschaltet ist. Falls dies der Fall ist, soll der zugehörige Text und das Bild auf „eingeschaltet“ geändert werden. Ansonsten sollen der Zustand „ausgeschaltet“ gesetzt sein. Da diese Beispiele nur für den Pin 16 gelten, müssen für die anderen Pins abgewandelte Zeilen implementiert werden. Dazwischen sind einzelne Zeilen eingefügt, die dazu da sind, gefundene Fehler abzufangen und diese dann als Text auszugeben. Falls keine Fehler erkannt, wird ein Text ausgegeben, dass alles erfolgreich initialisiert worden ist. Zuletzt wird als Beispiel, mit folgender Zeile: „InPin16.ValueChanged += InPin16_ValueChanged;“ für jeweils jeden einzelnen Input, der sogenannte Ereignis-Listener gestartet. Dieser erkennt Veränderungen der Inputs und startet zugehörigen Code, welcher im nächsten Kapitel beschrieben ist.

5.2.2 Ereignis-Listener für die Inputs

Es gibt insgesamt vier Ereignis-Listener, die aktiviert werden, sobald sich der zugehörige Zustand eines Inputs verändert. Es ist dabei unerheblich ob der Zustand auf „low“ oder „high“ fällt, in beiden Fällen wird die jeweilige Methode aufgerufen. Darin wird zuerst mit „var task = Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>“, eine Aufgabe gestartet die unabhängig vom Hauptcode verarbeitet wird. Dadurch bleibt die Anwendung reagierend und hängt sich nicht für eine kurze Zeit auf, wenn diese Methode aufgerufen wird. Daraufhin wird geprüft ob die Variablen für die

Invertierung und Umschaltung, entweder auf wahr oder falsch gesetzt sind. Erst dann wird geprüft ob die Spannung an den Inputs fällt oder steigt. Daraufhin soll je nach dem was passiert, die Anwendung richtig aktualisiert werden. Das heißt der Text und das Bild des jeweiligen Inputs sollten aktualisiert sein. Da dieser Programmcode umfangreich ausfällt, siehe Seite 41 bis 47 im Anhang.

5.2.3 Umschaltung der Inputs auf die Outputs

Für das Kontrollkästchen der Umschaltung von den Inputs auf die Outputs, wurde folgende Quelle verwendet [26]. Wenn das Kontrollkästchen aktiviert wird, sollen zuallererst die Schaltflächen für die Steuerung der Outputs deaktiviert sein, damit der Benutzer diese nicht gleichzeitig steuern kann. Da die Outputs mit der Schaltung der Inputs gesteuert werden sollen. Anschließend tauscht folgende Zeile „output22toggle.Content = ChangeImage(diodeOff);“ die Bilder der Schaltflächen mit Bildern von Dioden um. Dies wird wiederum drei weitere Male für die anderen Outputs wiederholt. Außerdem werden noch die derzeitigen Zustände der Inputs ausgelesen und die Outputs automatisch daran angepasst, das heißt wenn ein Input eingeschaltet ist, wird automatisch der zugehörige Output ebenfalls eingeschaltet. Natürlich wird zuvor auch noch geprüft ob die Invertierung eingeschaltet ist und dementsprechend werden die richtigen Schaltungen durchgeführt. Falls das Kontrollkästchen erneut betätigt und somit ausgeschaltet ist, müssen alle Outputs wieder ausgeschaltet und die Bilder von Dioden auf Schaltflächen umgetauscht werden. Für den gesamten Programmcode, siehe Seite 47 bis 48 im Anhang.

5.2.4 Invertierung der Inputs

Für die Invertierung wurde ebenfalls ein Kontrollkästchen eingefügt, welches ebenfalls dieselbe Quelle [26] verwendet. Um es kurz zu fassen, da der Programmcode hierbei ziemlich groß ist, wird hier bei der Invertierung geprüft, welche Zustände gerade anliegen und diese einfach umgekehrt. Dies geschieht sowohl beim einschalten, als auch dem ausschalten der Invertierung. Es wird eine Reihe von Bedingungen abgearbeitet und das entsprechende richtige ausgewählt, so dass das zugehörige Bild und der Text abgeändert werden kann. Siehe dazu Seite 48 bis 51 im Anhang.

5.2.5 Schaltflächen für die Schaltung der Outputs

Hierfür wurden sogenannte „Toggle Buttons“ verwendet, oder auch Schaltflächen genannt. Die Quelle hierfür ist unter [27] zu sehen. Jene Schaltflächen haben zwei verschiedene Zustände, wenn sie „checked“ oder „unchecked“ sind, das heißt sie sind entweder aktiviert oder deaktiviert. Jede Schaltfläche hat zwei zugehörige Events, eines welches die Diode einschaltet und das zugehörige Bild verändert, sowie eines welches die Diode ausschaltet und das zugehörige Bild ebenso verändert. Eine Besonderheit hierbei ist, dass die Vorlage zu den Schaltflächen verändert wurde, um einen Effekt zu entfernen. Denn wenn die Schaltflächen eingeschaltet waren, dann wurde ein graues Viereck um die Schaltfläche gerendert, dies sah unvorteilhaft aus und musste geändert werden. Da es so vorgesehen ist, dass die Schaltflächen bei Betätigung das Bild verändert und somit direkt sichtbar ist, dass diese nun eingeschaltet ist. In Visual Studio lässt sich die Vorlage kopieren, so dass diese komplett in den Code eingefügt wird. In dem vorliegenden Code wird nach den Schlüsselwörtern „Disabled“ und „Checked“ gesucht, hierbei werden jeweils bei beiden drei Werte auf „Transparent“ geändert. Dadurch erlangt man die nötige Transparenz der Schaltflächen wenn sie Deaktiviert, sowie betätigt sind. Alle anderen Werte werden auf den Standard belassen, damit diese wie vorgesehen funktionieren. Leider ist es notwendig den kompletten Code so zu belassen, da ansonsten gewisse Effekte nicht mehr angezeigt würden. Für den vollständigen Programmcode sind die Seiten von 51 bis 52 im Anhang zu beachten.

5.2.6 Methode zum Wechsel der Bilder

In dieser Anwendung existieren einige Bilder, die je nach Zustand gewechselt werden müssen. Dazu ist eine Methode notwendig, um den Zielpfad des neuen Bildes übergeben zu können. Die Quelle [28] fand hier ihren Einsatz. Der Code wurde speziell in eine Methode verlagert, die allgemein gehalten ist. Folgender Code enthält dafür alles nötige:

```
private Image ChangeImage(String destination) {
        Image myImage = new Image();
        BitmapImage biImage = new BitmapImage();
        biImage.UriSource = new Uri(destination, UriKind.Absolute);
        myImage.Stretch = Stretch.Uniform;
        myImage.Margin = new Thickness(-25, -25, -25, -25);
        myImage.Source = biImage;
        return myImage; }
```

Damit die Methode ihre Funktion erfüllen kann, muss dafür die Adresse beim aufrufen der Funktion mit angegeben sein. Die Pfade sind am Ende des Programmcodes deklariert und in feste Variablen gespeichert, zum Beispiel: „private const string diodeOn = "ms-appx:///GpioAssets/diodeOn.png";“. Nach dem der Pfad übergeben ist, müssen vorerst die notwendigen Variablen initialisiert werden. Anschließend wird mit

der Variable „destination“, der Pfad als Quelle an die zuvor initialisierte Variable „biImage“ übergeben. Der Typ dieses Pfades ist absolut, andere Typen sind möglich, aber hier nicht anwendbar. Anschließend stellt man noch verschiedene Einstellungen fest, dass das Bild automatisch skaliert und vergrößert wird. Zum Schluss fügt man der Variable „myImage“ die Quelle ein und gibt die vollständige Variable mit allen nötigen Werten zurück. Damit erhält man das neue Bild und kann dieses der alten Variable zuordnen, die zuvor diese Methode aufgerufen hat. Als Beispiel für das aufrufen: „In16Image.Source = ChangeImageSource(diodeOn);“. Zusätzlich existiert noch eine weitere ähnliche Methode, die dafür zuständig ist, speziell den XAML Image Typ zu unterstützen. Siehe dazu Seite 52 bis 53 im Anhang für den zuständigen Code.

5.2.7 Methode zum schließen der GPIO-Pins

Die geöffneten GPIO-Pins müssen, zum Beispiel beim Verlassen des Programms geschlossen und vom Arbeitsspeicher freigegeben werden. Dazu wurde eine spezielle Methode implementiert die jederzeit aufrufbar ist. Diese schaut bei jedem Pin nach, ob diese nicht den Wert „null“ enthalten. Denn falls sie diesen Wert besitzen, ist es nicht notwendig diese freizugeben. Ansonsten werden sie zum Beispiel mit „OutPin22.Dispose();“ geschlossen, sowie vom Arbeitsspeicher freigegeben. Dies ist für jeden weiteren Pin erforderlich, siehe dazu Seite 53 im Anhang.

5.2.8 Ergebnis der Seite zur Ansteuerung der GPIO

Somit sind alle essentiellen Funktionen programmiert, die zur Ansteuerung der GPIO und zur Funktion der Benutzeroberfläche benötigt werden. Hier folgen Bildschirmaufnahmen der fertigen Seite, wie sie in der Finalen Anwendung zu sehen ist.

Abbildung 5.2.1: GPIO Seite - Standard, Quelle: Eigenes Werk

In der Abbildung 5.2.1 ist zu sehen wie beide Kontrollkästchen ausgeschaltet sind und somit alles auf Standard gesetzt ist. Als Beispiel wurden hier Output 22 und 23 eingeschaltet um den Unterschied zwischen eingeschalteten und ausgeschalteten Zustand aufzuzeigen. Außerdem ist noch zu sehen, wie die beiden Kippschalter für Input 26 und 27 betätigt sind, so dass die Dioden im Programm eingeschaltet zu sehen sind.

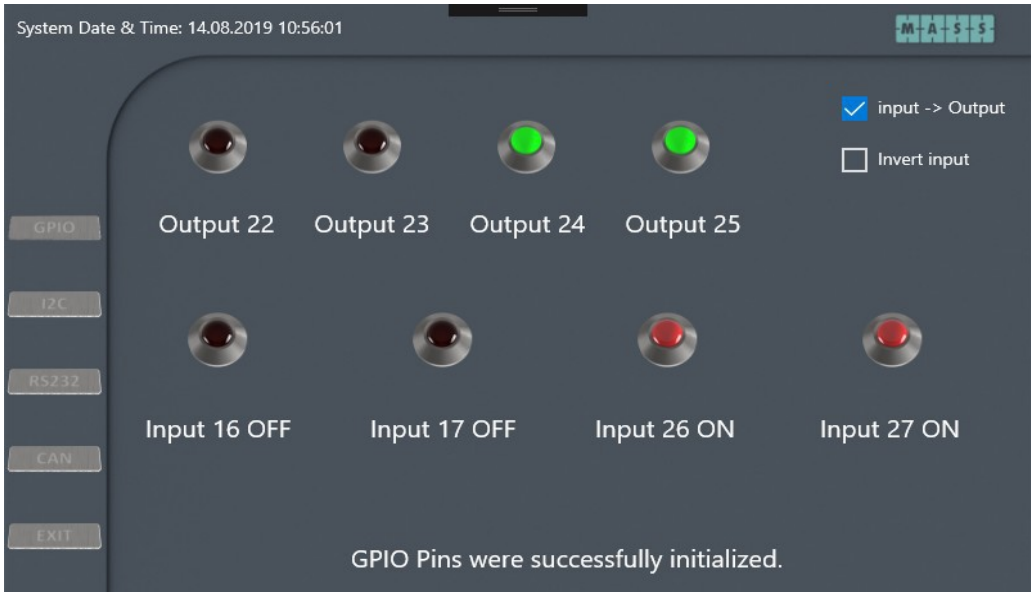

Abbildung 5.2.2: GPIO Seite – Umschaltung, Quelle: Eigenes Werk

Im Vergleich zur Abbildung 5.2.1, sieht man in der Abbildung 5.2.2 den Unterschied, wenn die Umschaltung betätigt wurde und nun die Outputs mit den Inputs gesteuert werden. Sofort sind alle Bilder der Schaltflächen mit den Bildern von Dioden ausgetauscht, die in diesem Fall Grün leuchten. Außerdem erkannte das Programm, dass bereits Input 26 und 27 ein eingeschaltetes Signal liefern. Dadurch sind Output 24 und 25 direkt im eingeschalteten Zustand zu sehen. Was nicht zu sehen ist, ist dass gleichzeitig auch die physischen Output Leuchten 24 und 25 anfangen zu leuchten.

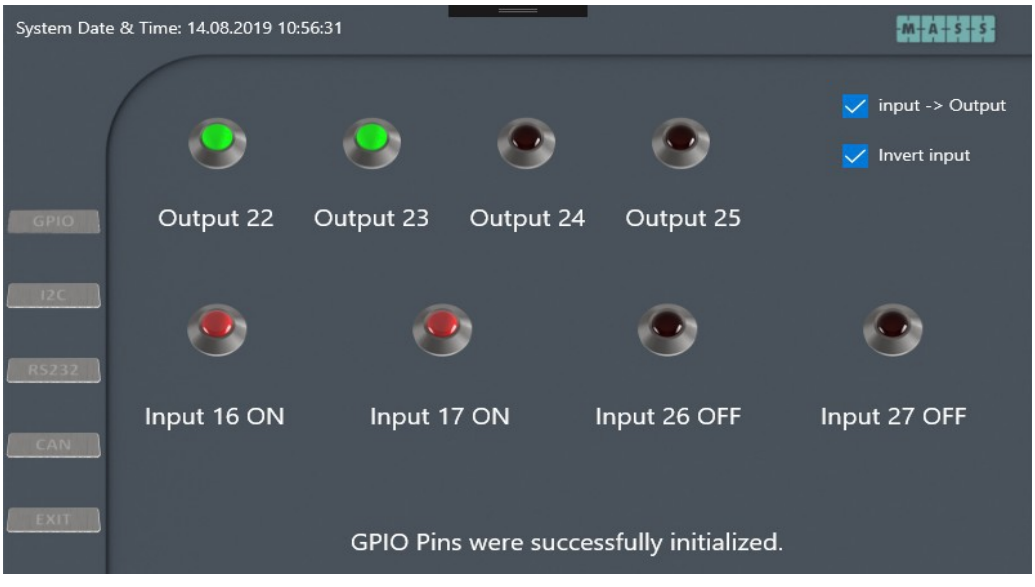

Abbildung 5.2.3: GPIO Seite – Umschaltung und Invertierung, Quelle: Eigenes Werk

Wie in der Abbildung 5.2.3 zu sehen ist, wurde noch zusätzlich die Invertierung geschaltet. Dadurch werden die Inputs umgekehrt verarbeitet und diese, die nun das Signal eingeschaltet senden, sind nun die Dioden auf der Benutzeroberfläche im ausgeschaltetem Zustand. Während die anderen die zuvor ausgeschaltet angezeigt waren, jetzt eingeschaltet sind. Die Outputs haben sich dementsprechend ebenfalls angepasst. Damit ist diese Seite fertiggestellt und alle nötigen Funktionen wurden implementiert. Außerdem ist alles so übersichtlich wie möglich gehalten, wie es die Zielsetzung vorgibt.

5.3 Auslesen der Adressen auf dem I²C-Bus

Hier wird die Implementierung, der Suche nach den Adressen auf dem I²C-Bus beschrieben und erklärt. Die Hauptquelle [29] wird hierbei hauptsächlich verwendet und ein Teil des Programmcodes wurde davon entnommen. Weitere Quellenangaben werden in den nachfolgenden Unterkapiteln erscheinen. Die nötigen Klassenbibliotheken die inkludiert wurden sind folgende:

```
using Windows.UI.Core;
using System.Threading.Tasks;
using Windows.Devices.Enumeration;
using Windows.Devices.I2c;
using System.Timers;
```

Dadurch können alle nötigen Funktionen und Methoden implementiert werden.

5.3.1 Methode FindDevicesAsync()

Diese Methode wurde wie zuvor schon erwähnt, komplett aus der Quelle [29] entnommen und geringfügig angepasst. Lediglich der Adressbereich der durchsucht werden soll, wurde angepasst. Der Adressbereich der nötig ist beginnt bei der Adresse „0x03“ und endet bei „0x77“. Das Einzige welches zusätzlich eingefügt werden musste ist eine Variable die für die sogenannte „Progress Bar“ hoch zählt und diese den Wert an jene übergibt. Dadurch ist es möglich den Fortschritt live mitzuverfolgen, so dass man nicht den Eindruck hat, dass nichts passieren würde. Schließlich benötigt der Benutzer einen Fortschrittsbalken damit dieser weiß, wann der Test fertig ist. Für die „Progress Bar“ ist die Quelle [30] zu beachten, da diese als Inspiration benutzt wurde. Der benutzte Programmcode initialisiert zuallererst den I²C-Controller und fordert alle notwendigen Parameter an. Anschließend folgt eine Schleife, die erst Mal alle nötigen Einstellungen für den Controller einstellt, unter anderem den Übertragungsmodus und die Übertragungsgeschwindigkeit. Danach werden alle Adressen durchlaufen die im Adressbereich definiert sind und er versucht auf jeder Adresse den Wert „0“ zu schreiben. Falls der Wert „0“ nicht auf der Adresse geschrieben werden kann, wird ein Fehler ausgegeben. Dieser wird direkt aufgefangen, so dass das Programm nicht

abstürzt. Ansonsten wenn der Wert erfolgreich geschrieben wurde, wird diese Adresse in einer Liste gespeichert. Sobald alle Adressen durchlaufen sind, wird am Ende die komplette Liste zurückgegeben. Da der Code umfangreich ist, ist dieser im Anhang auf Seite 63 bis 64 zu sehen.

5.3.2 Aufruf der Methode und Ausgabe der Liste

Die zuvor implementierte Methode soll sofort beim betreten der Seite ausgeführt werden. Es soll keinen Button für den Benutzer geben, um diesen Test manuell auszuführen. Deswegen wird die Methode direkt beim Wechsel zu der Seite gestartet. Folgender Code ist dazu notwendig:

```
var task = Dispatcher.RunAsync(CoreDispatcherPriority.Normal, async () => {
        IEnumerable<byte> address = await FindDevicesAsync();
        if (address.Count() > 0) {
          Status.Text = string.Format("{0} device(s) found.", address.Count());
          for (int i = 0; i < address.Count(); {
                listView1.Items.Add(string.Format("Device {0}: Adress:
        0x{1:X}", i + 1, address.ElementAt(i))); } }
        SetTimer();
        setRtcTime.Visibility = Visibility.Visible; });
```

Um diesen gesamten Aufruf ist noch ein „try“ Block, der Ausnahmen auffängt, so dass das Programm nicht abstürzt. Siehe dazu Seite 65 im Anhang, für den vollständigen Code. Zuallererst wird eine Aufgabe erstellt die asynchron zum Hauptcode läuft. Dadurch bleibt die Anwendung reagierend. In der ersten Zeile des Aufrufs wird die Methode aus dem Kapitel 5.3.1 aufgerufen und auf diese gewartet. Erst nach dem das Ergebnis als Liste zurückgegeben worden ist, wird ausgegeben wie viele Geräte von der Methode gefunden wurden. Anschließend wird eine Schleife durchlaufen bis alle Geräte die in der Liste eingetragen sind, ausgegeben werden. Als Ausgabe wurde eine sogenannte „List View“ verwendet, diese zeigt anschließend eine Liste auf der Benutzeroberfläche an. Die Inspiration hierfür stammt aus der Quelle [31]. Die Liste besitzt außerdem eine „Scrollbar“ an der Seite, so dass wenn jene Liste zu groß für den Bildschirm ist, es möglich ist nach unten zu scrollen. Am Schluss wird noch ein Timer gestartet und die Echtzeituhr angezeigt. Dazu mehr im folgenden Kapitel.

5.3.3 Echtzeituhr und Timer

Zuallererst ist ein Timer nötig, welcher aus Quelle [32] entnommen ist. Der Timer wird wie im vorherigen Kapitel zu sehen, im Aufruf am Ende gestartet und zählt in einem Intervall von einer Sekunde hoch. Das heißt nach jeder Sekunde die vergangen ist, wird eine Event Methode aufgerufen und ausgeführt. In dieser Event Methode, wird erst der I²C-Controller wiederum initialisiert und alle nötigen Einstellungen für die Kommunikation mit der Echtzeituhr vorbereitet. Um die Echtzeituhr auszulesen, wurde hauptsächlich die Quelle [33] verwendet. Außerdem stammen aus Quelle [33], vier spezielle Methoden, die die Konvertierung der Werte übernehmen, so dass am Ende das Datum und die Uhrzeit ausgegeben werden kann. An den übernommenen Methoden wurde nichts verändert. Anschließend wird so jede Sekunde, die Zeit von der Echtzeituhr aktualisiert und angezeigt. Darüber hinaus existiert noch ein Button der dafür da ist, die Echtzeituhr mit der Systemuhr des Raspberry Pi zu synchronisieren. Dafür ist folgende Quelle [34] notwendig, um die derzeitige Systemzeit auszulesen. Somit wenn der Button ausgelöst wird, startet die Methode die Initialisierung des I²C-Bus um eine Verbindung mit der Echtzeituhr aufzubauen. Anschließend kann mit folgender Zeile: „device.Write(ConvertTimeToByteArray(DateTime.Now));“, die Zeit der Echtzeituhr synchronisiert werden. Hierbei muss wiederum eine Konvertierung stattfinden, mit den vier Methoden die aus Quelle [33] stammen. Für den gesamten Programmcode, siehe Seite 65 bis 67 im Anhang.

5.3.4 Ergebnis der Benutzeroberfläche vom I²C-Bus

Da nun alles nötige implementiert wurde, sind hier nun die fertigen Bildschirmaufnahmen zu sehen, die in der Finalen Anwendung vorkommen.

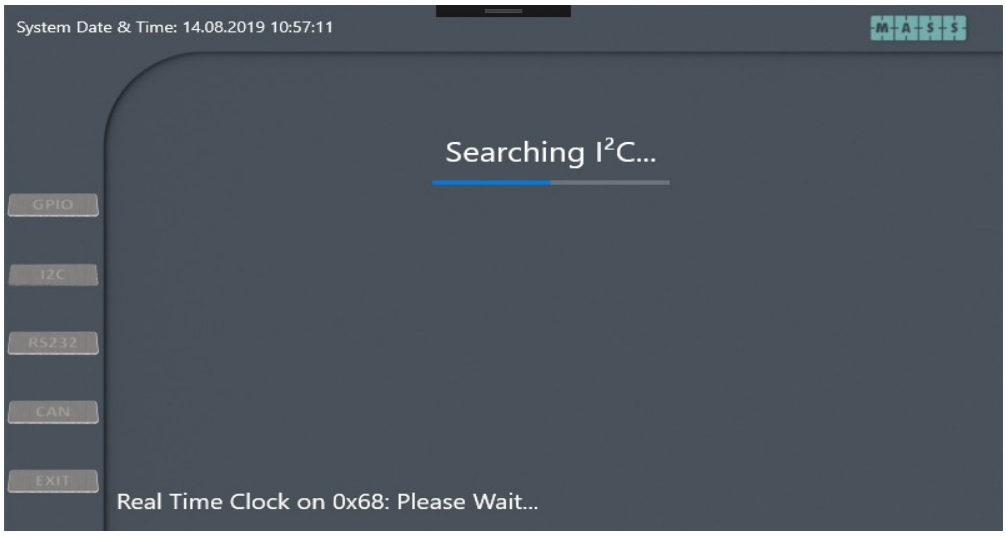

Abbildung 5.3.1: I²C Seite – Während der Suche, Quelle: Eigenes Werk

In der Abbildung 5.3.1 ist zu sehen, wie es während der Suche nach den I²C-Adressen aussieht. Hier ist die Suche zu circa 50% abgeschlossen, wie man es an der blauen „Progress Bar“ erkennen kann. Die Echtzeituhr ist bis jetzt noch nicht initialisiert, solange die Suche noch läuft.

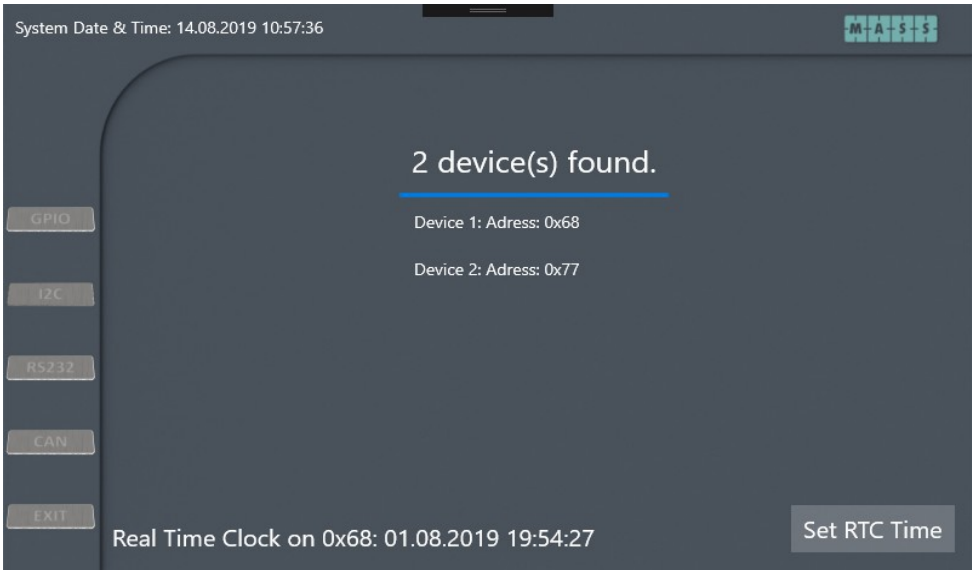

Abbildung 5.3.2: I²C Seite – Abgeschlossene Suche, Quelle: Eigenes Werk

Wie in der Abbildung 5.3.2 zu sehen ist, ist die Suche nun abgeschlossen und es wurden zwei Geräte gefunden. Die Liste wurde nun mit den Adressen der beiden Geräte aufgefüllt, so dass diese zu sehen sind. Daraufhin ist auch die Echtzeituhr initialisiert und wird seit dem jede Sekunde erneut aktualisiert. Ein Button um die Echtzeituhr mit der Systemzeit zu synchronisieren ist ebenfalls erschienen.

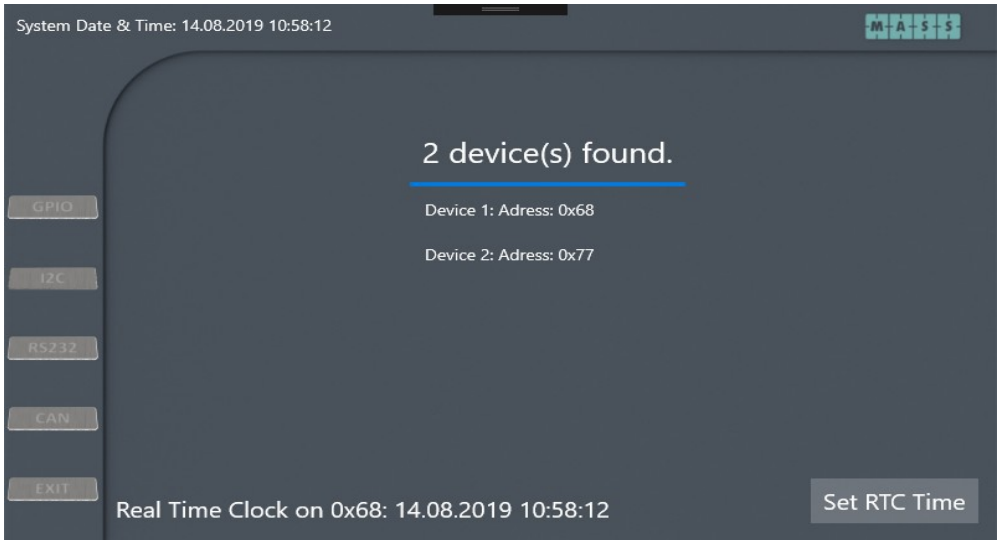

Abbildung 5.3.3: I²C Seite – Synchronisation der Echtzeituhr, Quelle: Eigenes Werk

In der letzten Abbildung 5.3.3 zum I²C-Bus ist zu sehen, dass nach dem der Button zur Synchronisation betätigt wurde, die Uhrzeit dieselbe ist, wie die der Systemzeit oben links in der Ecke. Außerdem lässt es sich mit der Abbildung 5.3.2 vergleichen, dass die Zeit eindeutig angepasst worden ist. Auch nach Neustart der Testumgebung behält die Echtzeituhr ihre Daten, sofern eine Batterie für diese integriert ist.

5.4 Serielle Schnittstelle RS232 Schleifentest

In diesem Kapitel wird auf den Schleifentest für die Serielle Schnittstelle RS232 eingegangen. Hierfür wurde wiederum die Quelle [23] verwendet, da diese alles nötige für die Ansteuerung der Seriellen Schnittstelle enthielt. Als erstes müssen folgende Klassenbibliotheken inkludiert werden:

```
using Windows.Storage.Streams;
using Windows.Devices.Enumeration;
using Windows.Devices.SerialCommunication;
using Windows.UI.Core;
using System.Threading.Tasks;
using System.Timers;
```

Hiermit lassen sich alle nötigen Funktionen verwenden, was essentiell ist. Der Test lässt sich mit einem Button starten. Sobald der Test läuft, wird der Button mit einem anderen Button ersetzt, mit dem man den Test wiederum abbrechen kann.

5.4.1 Timer und die Abbruch-Funktion

Zuallererst ist es notwendig einen Timer einzubauen, damit es möglich ist den Test zu unterbrechen, oder dass dieser automatisch nach einer vorgegebenen Zeit abbricht. Dazu wird wieder die Quelle [32] benötigt, wie im vorherigen Kapitel beim I²C-Bus. Der Unterschied hierbei ist, dass der Timer nur einen Intervall von sechs Sekunden besitzt und dieser nicht wiederholt wird. Das heißt nach sechs Sekunden wird das zugehörige Event ausgelöst und der dortige Programmcode ausgeführt. Die „Timer Event“ Methode, enthält fünf sogenannte „Dispose“ Funktionen. Es wird versucht somit sowohl die Datenübertragung, als auch die Verbindung der Seriellen Schnittstelle zu schließen und aus dem Arbeitsspeicher freizugeben. Außerdem wird ebenfalls der Timer gestoppt und aus dem Arbeitsspeicher gelöscht. Für den Programmcode, siehe Seite 71 bis 72 im Anhang. Außerdem gibt es zusätzlich einen Button, um den Test vorzeitig abzubrechen. Die Funktionen sind identisch zur „Timer Event“ Methode.

5.4.2 Serial() Methode

Dies ist die Hauptmethode dieser Seite. Wie zuvor erwähnt, wurde der gesamte Programmcode aus der Quelle [23] entnommen. Zusätzlich wurde der Code für den Wechsel der beiden Buttons erweitert. Auch musste eine Schleife hinzugefügt werden, damit zu sehen ist, dass mehrere Daten durch die Schleife des „Dongle“ übertragen wurden. Gleichzeitig wird bei jedem Durchlauf ein neuer Text ausgegeben. Nach zehn Durchläufen ist die Schleife beendet und der Timer wird ordnungsgemäß aus dem Arbeitsspeicher entfernt, bevor dieser sein Intervall beendet. Für den gesamten Quellcode, ist Seite 69 bis 71 im Anhang zu beachten.

5.4.3 Gesamtergebnis der RS232 Benutzeroberfläche

Hier wird nun die fertig programmierte Seite anhand von Bildschirmaufnahmen gezeigt. Zu sehen ist hier die Finale Version der gesamten Anwendung.

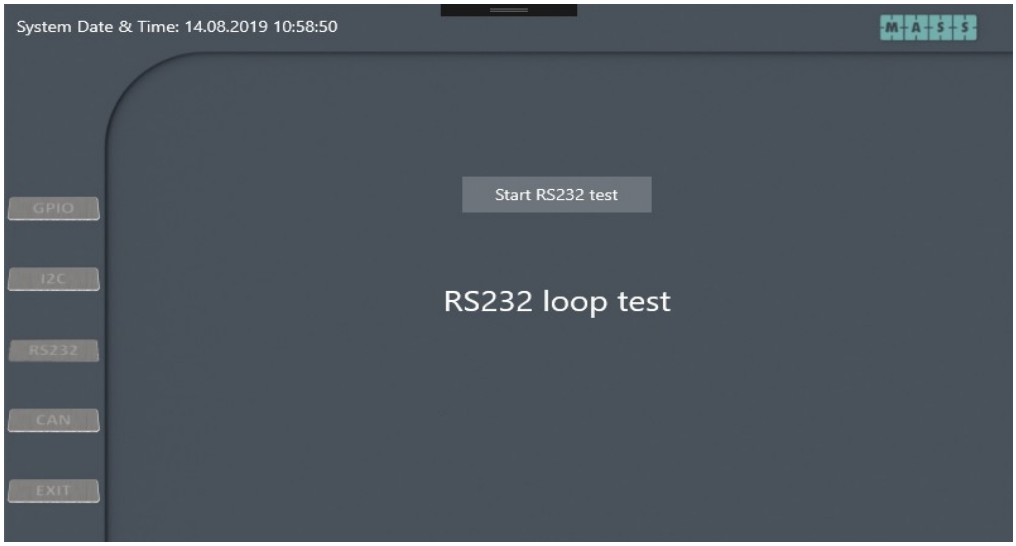

Abbildung 5.4.1: RS232 Seite – Startseite, Quelle: Eigenes Werk

Die Abbildung 5.4.1 zeigt, wie der RS232 Schleifentest aussieht bevor man diesen startet. Es ist lediglich nur der Button zum starten des Programms zu sehen und ein Text der anzeigt, dass man sich beim Schleifentest befindet.

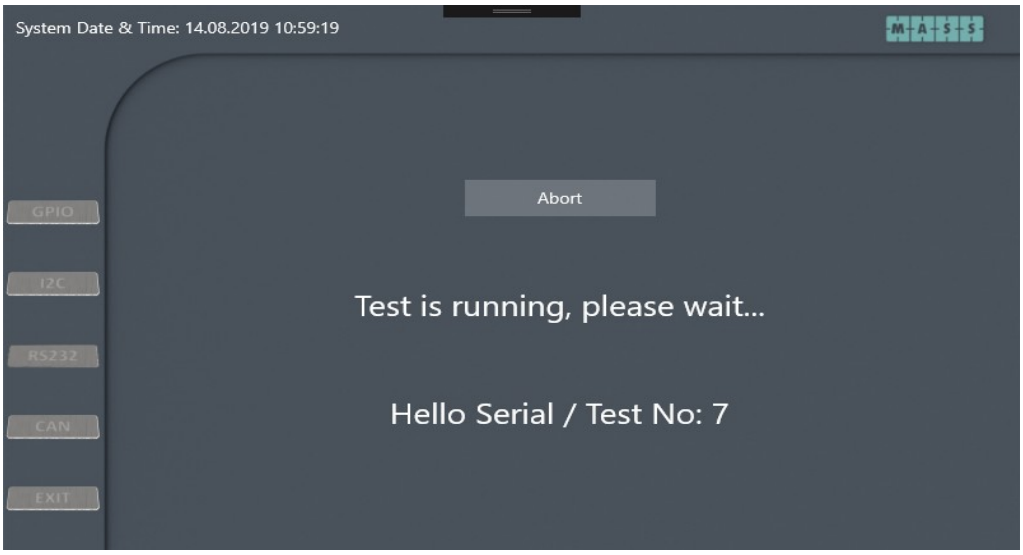

Abbildung 5.4.2: RS232 Seite – Laufender Schleifentest, Quelle: Eigenes Werk

Wie in der Abbildung 5.4.2 nun zu sehen ist, sieht so ein laufender Schleifentest aus. Hier ist mittlerweile erfolgreich die siebte Schleife durchlaufen. Wie zu erkennen ist, wurde der Start Button durch den Abbruch-Button ersetzt. Außerdem wird noch gezeigt, dass der Test am laufen ist und man auf das Ende warten sollte.

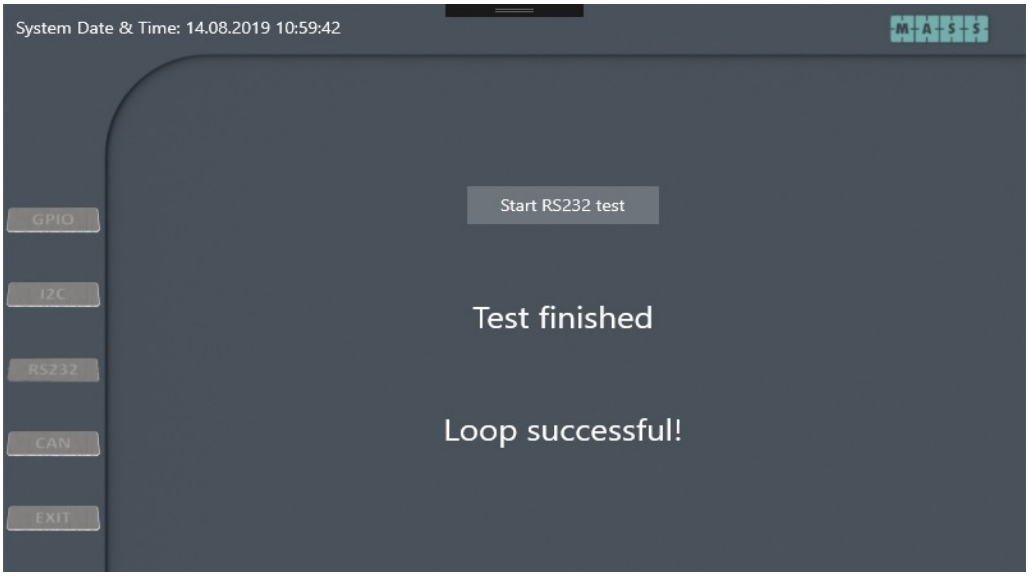

Abbildung 5.4.3: RS232 Seite – Erfolgreicher Schleifentest, Quelle: Eigenes Werk

In dieser Abbildung 5.4.3, sieht man den erfolgreich beendeten Schleifentest der RS232 Seriellen Schnittstelle. Es ist nun möglich den Schleifentest mit Betätigung des Start Buttons zu wiederholen. Da bei Beendigung des Schleifentests alle benutzten Variablen aus dem Arbeitsspeicher freigegeben worden sind.

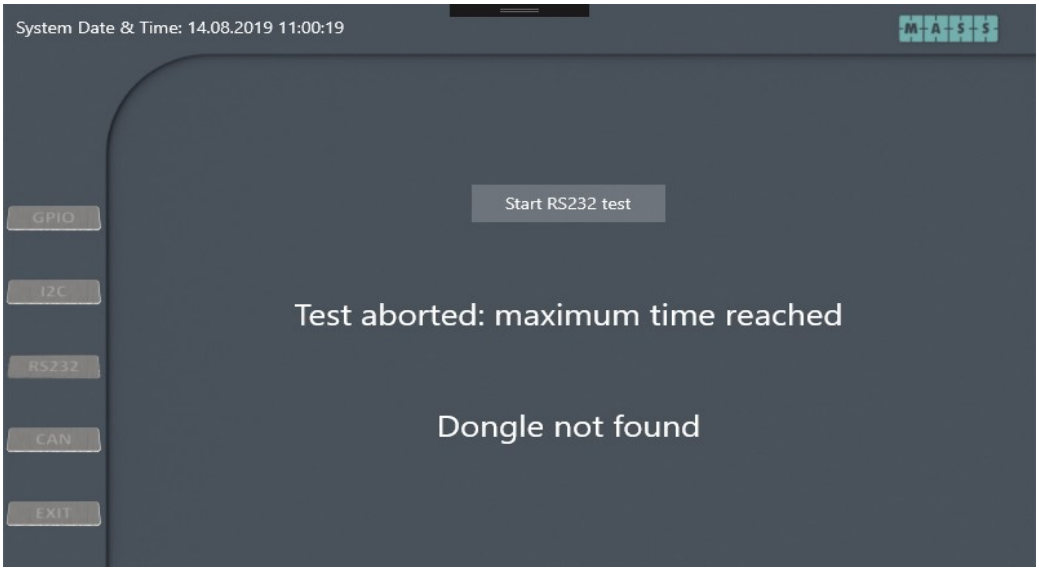

Abbildung 5.4.4: RS232 Seite – Test gescheitert, Quelle: Eigenes Werk

In der letzten Abbildung 5.4.4 ist zu sehen, dass der Schleifentest gescheitert ist. Da die maximale Zeit entweder überschritten ist, oder der „Dongle“ nicht gefunden wurde, falls dieser nicht angeschlossen ist. Es ist nun wieder möglich den Schleifentest sofort zu wiederholen. Falls der Test vom Benutzer selbst abgebrochen wird, erscheint lediglich eine Meldung das der Test erfolgreich abgebrochen ist. Dafür wurde keine zusätzliche Abbildung aufgenommen.

5.5 Initialisierung des CAN-Controllers

Als letzter Test soll der CAN-Controller MCP2515 initialisiert und getestet werden, ob dieser existiert. Dazu muss erst eine Verbindung mit dem SPI-Bus hergestellt sein. Da die Implementierung davon schwer ausfiel und auch nach mehreren Versuchen gescheitert ist, wurde eine externe Treiber Klassenbibliothek dafür verwendet. Der gesamte Programmcode wurde aus der Quelle [35] entnommen und als Klassenbibliothek inkludiert. Das heruntergeladene Projekt musste als vorhandenes Projekt in Visual Studio hinzugefügt werden. Anschließend ist ein Verweis dazu hinterlegt worden. Dadurch ist nur eine Klassenbibliothek mit folgender Zeile zu inkludieren: „using Mcp2515Can;“. Das Einzige was in dem verwendeten Projekt nicht vorhanden war, ist eine „Dispose“ Funktion, um nach dem Test die nicht mehr benötigten Verbindungen aus dem Arbeitsspeicher freizugeben. Dies wurde hinzugefügt, so dass es möglich ist, den Test mehrmals durchzuführen ohne dass ein Neustart notwendig ist. Folgende Zeilen sind die hinzugefügten:

```
public void Dispose() {
        mDev.Dispose();
        mInterrupt.Dispose(); }
```

Die neue Methode wird in Zeile 110 mit dem Aufruf „Dispose();“ aufgerufen, nach dem versucht wurde den Controller zu initialisieren. Der gesamte Quellcode der Datei „Mcp2515CanController.cs“ ist im Anhang auf Seite 75 bis 77 zu sehen. Da leider keine Anleitung, oder ein Textdokument vom Autor beilag, ist es nicht möglich genau auszusagen was der Programmcode durchführt. Aber es wird damit erfolgreich der CAN-Controller initialisiert und anhand dessen lässt sich feststellen, dass dieser Funktionsfähig ist. Falls nämlich keine Initialisierung möglich ist, wirft der Programmcode eine Ausnahme aus, die aufgefangen wird. Somit ist es möglich eine gezielte Fehlermeldung anzuzeigen, dass der besagte Controller nicht gefunden wurde, oder nicht Funktionsfähig zu sein scheint. Die Eigenleistung hierbei ist, auf Seite 78 bis 79 im Anhang ersichtlich. Denn es musste wie schon bei den anderen Seiten, ein Start-Button und ein Abbruch-Button implementiert werden. Außerdem musste die Ausnahmebehandlung implementiert sein, damit die Fehlermeldungen aufgefangen werden und nicht die gesamte Anwendung zum Absturz bringen.

5.5.1 Benutzeroberfläche des CAN-Controller Tests

In diesem Kapitel sind Bildschirmaufnahmen des CAN-Controller Tests zu sehen. Diese wurden in der Finalen Anwendung aufgenommen.

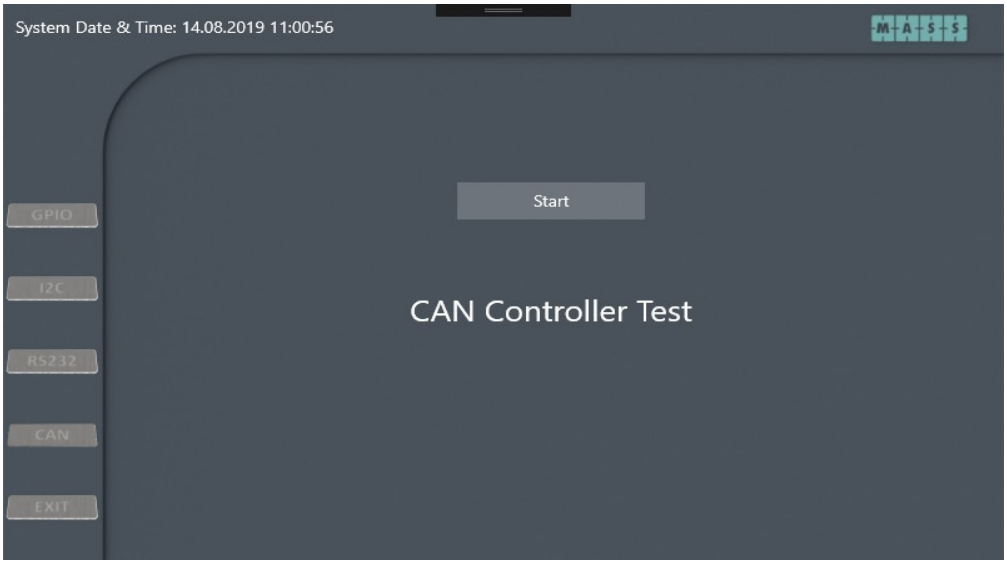

Abbildung 5.5.1: CAN Seite – Startseite, Quelle: Eigenes Werk

Wie in der Abbildung 5.5.1 zu sehen ist, ähnelt der Aufbau dem des RS232 Schleifentests mit der Abbildung 5.4.1. Es ist wieder ein Start Button vorhanden, mit dem der Test gestartet werden kann. Außerdem ist mit einem Textfeld angezeigt in welchem Test man sich befindet.

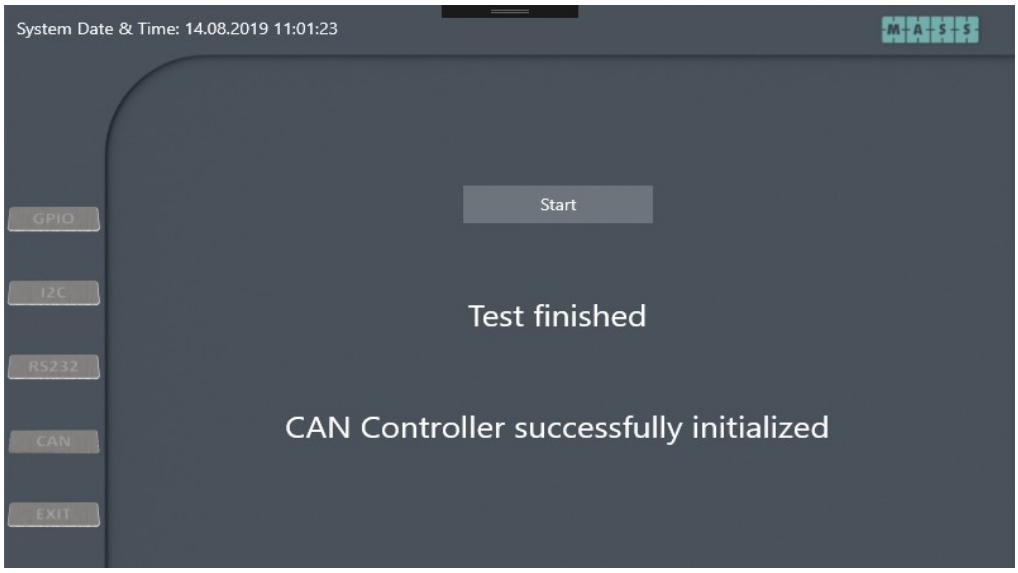

Abbildung 5.5.2: CAN Seite – Test erfolgreich, Quelle: Eigenes Werk

In der Abbildung 5.5.2 ist zu sehen, wie der erfolgreiche Test aussieht. Es wird angezeigt, dass der CAN-Controller erfolgreich initialisiert werden konnte und dass der Test abgeschlossen ist.

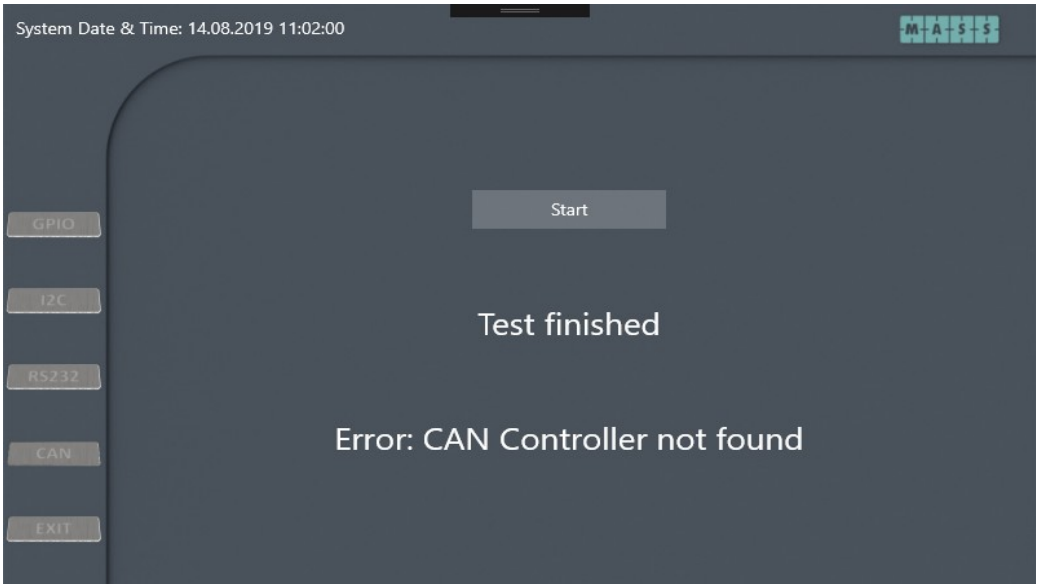

Abbildung 5.5.3: CAN Seite – Test gescheitert, Quelle: Eigenes Werk

In der letzten Abbildung 5.5.3 ist zu sehen, wie die Fehlermeldung aussieht wenn der CAN-Controller nicht gefunden wurde. Beziehungsweise wenn dieser nicht initialisiert werden konnte, da er zum Beispiel falsch eingebaut wurde. Damit ist auch der CAN-Controller Test erfolgreich implementiert und vollständig funktionsfähig. Das Design ist simpel genug und nicht überladen, damit der Test effizient und schnell durchgeführt werden kann.

5.6 Zusammenführung aller Komponenten

In diesem Kapitel wird darauf eingegangen wie alle einzelnen Komponenten in eine gesamte Anwendung zusammengeführt worden sind. Da alle Tests in einer Anwendung sein sollen, ist es notwendig diese zu verbinden. Dazu ist jeder einzelne Programmcode auf je einer Seite vorhanden. Insgesamt gibt es somit vier Test-Seiten und eine Hauptmenü-Seite. Zusätzlich existiert noch eine „Exit“ Seite, die nur dazu da ist, um am Ende bevor die Anwendung geschlossen wird, alle verwendeten Verbindungen und Variablen sicher aus dem Arbeitsspeicher zu entfernen. Deswegen ist es auch essentiell bei jedem Verlassen einer Seite, alle nicht mehr notwendigen Variablen und Verbindungen mit „GC.Collect();“ aus dem Arbeitsspeicher freizugeben. Dafür gibt es auf jeder einzelnen Seite jeweils zwei Event Methoden, eine die ausgeführt wird wenn die Seite geladen wird „OnNavigatedTo“ und eine wenn diese Verlassen wurde „OnNavigatedFrom“. Als Inspiration diente die Quelle [36], wo die Möglichkeit gefunden wurde, eine Hauptseite mit einer Subseite zu versehen. Diese Subseite lädt dann alle Tests ein, während die Hauptseite unberührt bleibt. Dadurch ist es möglich von jedem Test aus, den nächsten zu starten oder die Anwendung zu beenden.

5.6.1 Das Hauptmenü

Hier wird das Hauptmenü vorgestellt dessen Aufgabe es ist, alle anderen Seiten zusammenzuhalten und auch als Ankerpunkt zu dienen. Das Hauptmenü ist immer sichtbar und ist das Erste welches beim starten der Anwendung angezeigt wird. Wichtig hervorzuheben ist folgender Code in der XAML-Datei des Hauptmenüs:

```
<Grid.Background>
        <ImageBrush ImageSource="backPlate.png" Stretch="Fill"/>
</Grid.Background>
```

Hier wird als Hintergrundbild eine vorgefertigte Vorlage geladen, die von der MASS GmbH bereitgestellt worden ist. Dadurch sieht der Hintergrund der Anwendung strukturiert aus. Darauf wurden am linken Rand fünf Buttons hinzugefügt, vier für alle Tests und ein Button um die Anwendung zu beenden. Das Design dieser fünf Buttons wurde ebenfalls von der MASS GmbH dafür erstellt. Wie zuvor erwähnt, stammt aus Quelle [36] die Idee mit folgendem Code ein Subfenster zu erstellen:

```
<Frame x:Name="subFrame" Margin="75,50,0,0" Width="715" Height="420">
        <Image x:Name="Platine" Source="gpio-platine.png"/>
</Frame>
```

Wie im Code zu sehen ist, wurde das neue Fenster „subFrame“ genannt. Die Werte die bei „Margin“ übergeben werden, dienen zur Positionierung des Fensters. Die Werte „Width“ und „Height“ setzen die Breite des Subfensters auf 715 Pixel, sowie die Höhe auf 420 Pixel. Dadurch ist das Subfenster kleiner als das Hauptfenster des Hauptmenüs. Zusätzlich ist als Verschönerung ein Bild eingefügt, damit das Hauptmenü beim starten nicht so leer wirkt.

In der C#-Datei des Hauptmenüs befindet sich der ganze Code für die Umschaltung der Seiten, wenn die Buttons betätigt werden. Aber zuallererst wird ein Text in der linken oberen Ecke erstellt und ein Timer gestartet. Dies ist notwendig, da dort die Systemzeit sowie das Datum angezeigt wird und im Intervall von einer Sekunde aktualisiert wird. Hierzu sind die Quellen [32] und [34] wieder zu beachten. Auch wurde die Methode, wie sie in Kapitel 5.2.6 zu sehen war, wieder verwendet. Diese ist zum Wechsel der Bilder nötig, um die Buttons im gedrückten, sowie nicht gedrückten Zustand zu zeigen. Damit wird gleichzeitig auch aufgezeigt in welchem Test man sich befindet. Zum Wechsel der Subfenster Seiten ist als Beispiel folgende Zeile notwendig:

```
subFrame.Navigate(typeof(GpioPage), null);
```

In dem Fall hier, wird beim Aufruf dieser Codezeile der Inhalt des Subfensters mit der GPIO-Seite ausgetauscht. Wenn der „Exit“ Button betätigt wird, dann öffnet sich eine Leere Seite, dadurch werden alle Variablen und Verbindungen ordnungsgemäß aus dem Arbeitsspeicher entfernt. Zudem wird auch noch der Timer für die Uhrzeit gestoppt.

Nachdem alles ordnungsgemäß beendet wurde, wird mit folgender Zeile die Anwendung komplett beendet:

```
MASS_Platinen_Test.App.Current.Exit();
```

Es gibt auch andere Möglichkeiten die Anwendung zu beenden, aber nur mit dieser ist es garantiert, dass die Anwendung ordnungsgemäß beendet und nichts zurückgelassen wird.

5.6.2 Ergebnis des Hauptmenüs

Damit ist nun alles implementiert und die Anwendung vollständig. In der letzten Abbildung 5.6.1 ist die fertige Anwendung zu sehen, nach dem diese gestartet wurde.

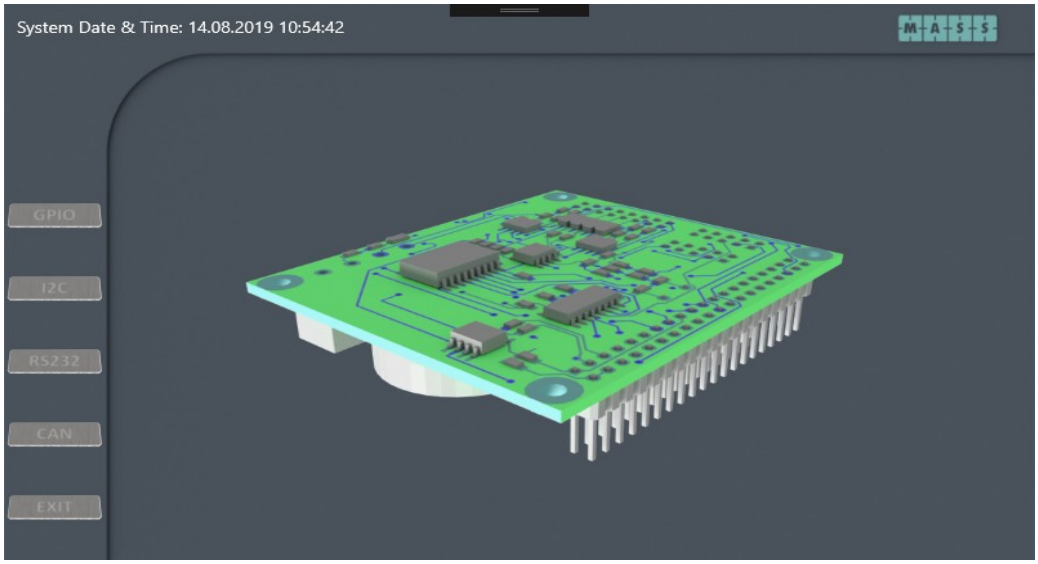

Abbildung 5.6.1: Hauptmenü – Erste Seite der Anwendung, Quelle: Eigenes Werk

6 Fazit

Die Entwicklung einer prototypischen Software zur Ansteuerung der GPIO-Pins unter Windows 10 IoT für den Raspberry Pi 3 Modell B konnte somit erfolgreich abgeschlossen werden. Alle nötigen Zielsetzungen wurden erfüllt und die Anwendung ist lauffähig. Es wurden mehrere Tests durchgeführt und keine schwerwiegenden Fehler gefunden. Die Performance der Anwendung ist zufriedenstellend, da die CPU des Raspberry Pi nicht belastet wird und es existieren keine Speicherlecks, dadurch bleibt die Anwendung stets stabil. Alle Tests lassen sich effizient durchführen und die Übersichtlichkeit ist immer gegeben. Außerdem ist der Quellcode übersichtlich geschrieben und mit nötigen Kommentaren gefüllt. Somit ist eine zukünftige Erweiterung des Programmcodes gegeben und kann jederzeit durchgeführt werden.

Lediglich die Programmierung unter C# und XAML war anfangs schwierig. Aber nach einer kurzen Einarbeitungszeit ließ sich dies problemlos bewältigen. Auch ein Problem war die Ansteuerung des SPI-Bus, welches sich nicht direkt lösen ließ. Durch das auffinden der Treiber Klassenbibliothek, konnte aber die Verbindung zum CAN-Controller aufgebaut werden und auch dieses Ziel wurde somit erfolgreich erreicht. Durch die Bereitstellung von Grafiken, durch die MASS GmbH, ließ sich auch grafisch die Anwendung aufwerten. Die Übersichtlichkeit blieb dadurch erhalten und die Anwendung blieb weiterhin performant. Noch zu erwähnen ist, ist dass die vorgefertigte Testumgebung bei der Umsetzung sehr geholfen hat. Denn durch die übersichtliche Anordnung aller Leuchtdioden und Kippschalter, ließen sich alle programmierten Funktionen direkt testen, was der Entwicklung des Quellcodes gut tat.

6.1 Ausblick

Da die vorgegebene Zeit nicht vollkommen ausgereicht hatte, um alles zu perfektionieren, gibt es noch einige Verbesserungen die in der Zukunft durchgeführt werden können. Der Programmcode ist übersichtlich gehalten, aber noch nicht vollständig vereinfacht. Vor allem im C#-Code der GPIO-Seite lassen sich noch einige Verbesserungen durchführen. Da es dort viele wiederholende Elemente gibt kann man diese kürzen und vereinfachen. Man kann Methoden dafür entwickeln wo man dann nur nötige Variablen übergibt, die die zugehörigen Inputs und Outputs identifizieren kann, aber alles nötige enthalten, um die sich wiederholenden Codesegmente zu vereinen. Einige Ideen dahingehend waren schon vorhanden, aber durch Zeitmangel nicht mehr durchführbar.

Ebenso ist es noch notwendig den Programmcode dynamisch an verschiedene Auflösungen des Bildschirms anzupassen. Da der vorliegende Code nur für die Auflösung von 800 x 480 Pixel vorgesehen ist. Falls also ein Bildschirm mit einer größeren Auflösung angeschlossen wird, sind zum Beispiel die Buttons des Hauptmenüs immer noch klein und auch das Subfenster behält die feste Größe die vorgegeben wurde. Teile des Codes skalieren bereits einige Elemente, aber dies ist nicht für alle der Fall. Ein möglicher Lösungsansatz wäre, dass beim starten der Anwendung die jetzige Bildschirmauflösung abgerufen wird. Anschließend lässt sich durch die ermittelte Auflösung, die nötigen Einstellungen dynamisch ändern. Natürlich müsste man für verschiedene Auflösungen, auch verschiedene Einstellungssätze speichern.

Auch müsste der Programmcode für verschiedene Versionen von Windows 10 IoT Core angepasst werden. Da zum Beispiel ältere Versionen des Betriebssystems nicht kompatibel mit dem jetzigen Programmcode sind. Dahingehend müsste der Code angepasst werden, dafür ist aber eine aufwendige Suche nötig. Für zukünftige Versionen des Betriebssystems kann es ebenfalls nötig sein, den Code anzupassen. Notwendig ist auf jeden Fall die neu Kompilierung des Quellcodes, für zukünftige Versionen von Windows 10 IoT Core.

7 Anhang

Der Anhang enthält den gesamten geschriebenen und benutzten Quellcode.

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 1

```
1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4  using System.Linq;
5  using System.Runtime.InteropServices.WindowsRuntime;
6  using Windows.Devices.Gpio;
7  using Windows.Foundation;
8  using Windows.Foundation.Collections;
9  using Windows.UI.Core;
10 using Windows.UI.Xaml;
11 using Windows.UI.Xaml.Controls;
12 using Windows.UI.Xaml.Controls.Primitives;
13 using Windows.UI.Xaml.Data;
14 using Windows.UI.Xaml.Input;
15 using Windows.UI.Xaml.Media;
16 using Windows.UI.Xaml.Media.Imaging;
17 using Windows.UI.Xaml.Navigation;
18
19 namespace MASS_Platinen_Test
20 {
21     /// <summary>
22     /// GpioPage: This page contains all needed elements to control the inputs
       and outputs of the Raspberry Pi3.
23     /// Also you can set the flag to control the outputs with the inputs.
24     /// Or you can set a flag to invert the inputs, for special cases.
25     /// Author: Daniel Heintze
26     /// </summary>
27     public sealed partial class GpioPage : Page
28     {
29         public GpioPage()
30         {
31             this.InitializeComponent();
32         }
33
34         //Initializes all pins and reads the inputs to set the graphics right,
           also activates the on change events.
35         private void InitGPIO()
36         {
37             var gpio = GpioController.GetDefault();
38
39             //If no gpio were found, show a error message.
40             if (gpio == null)
41             {
42                 GpioStatus.Text = "GPIO Controller not found.";
43             }
44
45             try
46             {
47                 //Opens the four output pins.
48                 OutPin22 = gpio.OpenPin(LED_22);
49                 OutPin23 = gpio.OpenPin(LED_23);
50                 OutPin24 = gpio.OpenPin(LED_24);
51                 OutPin25 = gpio.OpenPin(LED_25);
52
53                 //Opens the four input pins.
54                 InPin16 = gpio.OpenPin(SWITCH_16);
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 2

```
55                  InPin17 = gpio.OpenPin(SWITCH_17);
56                  InPin26 = gpio.OpenPin(SWITCH_26);
57                  InPin27 = gpio.OpenPin(SWITCH_27);
58
59                  //Write the value "low" on all outputs.
60                  OutPin22.Write(GpioPinValue.Low);
61                  OutPin23.Write(GpioPinValue.Low);
62                  OutPin24.Write(GpioPinValue.Low);
63                  OutPin25.Write(GpioPinValue.Low);
64
65                  //Set the drive mode to "output" for all outputs.
66                  OutPin22.SetDriveMode(GpioPinDriveMode.Output);
67                  OutPin23.SetDriveMode(GpioPinDriveMode.Output);
68                  OutPin24.SetDriveMode(GpioPinDriveMode.Output);
69                  OutPin25.SetDriveMode(GpioPinDriveMode.Output);
70
71                  //Try to set the drive mode to "Input Pull Down" for all      ↵
                      inputs, else set them to standard input.
72                  if (InPin16.IsDriveModeSupported                            ↵
                      (GpioPinDriveMode.InputPullDown))
73                      InPin16.SetDriveMode(GpioPinDriveMode.InputPullDown);
74                  else
75                      InPin16.SetDriveMode(GpioPinDriveMode.Input);
76
77                  if (InPin17.IsDriveModeSupported                            ↵
                      (GpioPinDriveMode.InputPullDown))
78                      InPin17.SetDriveMode(GpioPinDriveMode.InputPullDown);
79                  else
80                      InPin17.SetDriveMode(GpioPinDriveMode.Input);
81
82                  if (InPin26.IsDriveModeSupported                            ↵
                      (GpioPinDriveMode.InputPullDown))
83                      InPin26.SetDriveMode(GpioPinDriveMode.InputPullDown);
84                  else
85                      InPin26.SetDriveMode(GpioPinDriveMode.Input);
86
87                  if (InPin27.IsDriveModeSupported                            ↵
                      (GpioPinDriveMode.InputPullDown))
88                      InPin27.SetDriveMode(GpioPinDriveMode.InputPullDown);
89                  else
90                      InPin27.SetDriveMode(GpioPinDriveMode.Input);
91              }
92              catch (Exception ex)
93              {
94                  GpioStatus.Text = "GPIO failed to initialize.";
95                  ErrorHandling.Text = string.Format("{0}", ex);
96                  error = true;
97              }
98
99              //Checks if a input is on high and updates the graphics          ↵
                  accordingly.
100             if (InPin16.Read() == GpioPinValue.High)
101             {
102                 In16Image.Source = ChangeImageSource(diodeOn);
103                 In16Status.Text = "Input 16 ON";
104             }
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 3

```
105             else
106             {
107                 In16Image.Source = ChangeImageSource(diodeOff);
108                 In16Status.Text = "Input 16 OFF";
109             }
110             if (InPin17.Read() == GpioPinValue.High)
111             {
112                 In17Image.Source = ChangeImageSource(diodeOn);
113                 In17Status.Text = "Input 17 ON";
114             }
115             else
116             {
117                 In17Image.Source = ChangeImageSource(diodeOff);
118                 In17Status.Text = "Input 17 OFF";
119             }
120             if (InPin26.Read() == GpioPinValue.High)
121             {
122                 In26Image.Source = ChangeImageSource(diodeOn);
123                 In26Status.Text = "Input 26 ON";
124             }
125             else
126             {
127                 In26Image.Source = ChangeImageSource(diodeOff);
128                 In26Status.Text = "Input 26 OFF";
129             }
130             if (InPin27.Read() == GpioPinValue.High)
131             {
132                 In27Image.Source = ChangeImageSource(diodeOn);
133                 In27Status.Text = "Input 27 ON";
134             }
135             else
136             {
137                 In27Image.Source = ChangeImageSource(diodeOff);
138                 In27Status.Text = "Input 27 OFF";
139             }
140
141             //Starts the event checks, if the value of a input is changed.
142             InPin16.ValueChanged += InPin16_ValueChanged;
143             InPin17.ValueChanged += InPin17_ValueChanged;
144             InPin26.ValueChanged += InPin26_ValueChanged;
145             InPin27.ValueChanged += InPin27_ValueChanged;
146
147             if (error != true)
148             {
149                 GpioStatus.Text = "GPIO Pins were successfully initialized.";
150             }
151         }
152
153         /* All four checks, if the value of the input was changed, for every
              input.
154          * Also checks if the inverted or "Input to Output" flags are set and
              handles the update of the graphics accordingly.
155          */
156         private void InPin16_ValueChanged(GpioPin sender,
              GpioPinValueChangedEventArgs e)
157         {
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 4

```
158             var task = Dispatcher.RunAsync(CoreDispatcherPriority.Normal, ()
                  =>
159             {
160                 if (invert == false)
161                 {
162                     if (umschalt == false)
163                     {
164                         if (e.Edge == GpioPinEdge.RisingEdge)
165                         {
166                             In16Image.Source = ChangeImageSource(diodeOn);
167                             In16Status.Text = "Input 16 ON";
168                         }
169                         else
170                         {
171                             In16Image.Source = ChangeImageSource(diodeOff);
172                             In16Status.Text = "Input 16 OFF";
173                         }
174                     }
175                     else if (umschalt == true)
176                     {
177                         if (e.Edge == GpioPinEdge.RisingEdge)
178                         {
179                             In16Image.Source = ChangeImageSource(diodeOn);
180                             In16Status.Text = "Input 16 ON";
181                             OutPin22.Write(GpioPinValue.High);
182                             output22toggle.Content = ChangeImage(diodeOn2);
183                         }
184                         else
185                         {
186                             In16Image.Source = ChangeImageSource(diodeOff);
187                             In16Status.Text = "Input 16 OFF";
188                             OutPin22.Write(GpioPinValue.Low);
189                             output22toggle.Content = ChangeImage(diodeOff);
190                         }
191                     }
192                 }
193                 else if (invert == true)
194                 {
195                     if (umschalt == false)
196                     {
197                         if (e.Edge == GpioPinEdge.FallingEdge)
198                         {
199                             In16Image.Source = ChangeImageSource(diodeOn);
200                             In16Status.Text = "Input 16 ON";
201                         }
202                         else
203                         {
204                             In16Image.Source = ChangeImageSource(diodeOff);
205                             In16Status.Text = "Input 16 OFF";
206                         }
207                     }
208                     else if (umschalt == true)
209                     {
210                         if (e.Edge == GpioPinEdge.FallingEdge)
211                         {
212                             In16Image.Source = ChangeImageSource(diodeOn);
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 5

```
213                             In16Status.Text = "Input 16 ON";
214                             OutPin22.Write(GpioPinValue.High);
215                             output22toggle.Content = ChangeImage(diodeOn2);
216                         }
217                         else
218                         {
219                             In16Image.Source = ChangeImageSource(diodeOff);
220                             In16Status.Text = "Input 16 OFF";
221                             OutPin22.Write(GpioPinValue.Low);
222                             output22toggle.Content = ChangeImage(diodeOff);
223                         }
224                     }
225                 }
226             });
227         }
228         private void InPin17_ValueChanged(GpioPin sender,
              GpioPinValueChangedEventArgs e)
229         {
230             var task = Dispatcher.RunAsync(CoreDispatcherPriority.Normal, ()
                => {
231                 if (invert == false)
232                 {
233                     if (umschalt == false)
234                     {
235                         if (e.Edge == GpioPinEdge.RisingEdge)
236                         {
237                             In17Image.Source = ChangeImageSource(diodeOn);
238                             In17Status.Text = "Input 17 ON";
239                         }
240                         else
241                         {
242                             In17Image.Source = ChangeImageSource(diodeOff);
243                             In17Status.Text = "Input 17 OFF";
244                         }
245                     }
246                     else if (umschalt == true)
247                     {
248                         if (e.Edge == GpioPinEdge.RisingEdge)
249                         {
250                             In17Image.Source = ChangeImageSource(diodeOn);
251                             In17Status.Text = "Input 17 ON";
252                             OutPin23.Write(GpioPinValue.High);
253                             output23toggle.Content = ChangeImage(diodeOn2);
254                         }
255                         else
256                         {
257                             In17Image.Source = ChangeImageSource(diodeOff);
258                             In17Status.Text = "Input 17 OFF";
259                             OutPin23.Write(GpioPinValue.Low);
260                             output23toggle.Content = ChangeImage(diodeOff);
261                         }
262                     }
263                 }
264                 else if (invert == true)
265                 {
266                     if (umschalt == false)
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 6

```
267                     {
268                         if (e.Edge == GpioPinEdge.FallingEdge)
269                         {
270                             In17Image.Source = ChangeImageSource(diodeOn);
271                             In17Status.Text = "Input 17 ON";
272                         }
273                         else
274                         {
275                             In17Image.Source = ChangeImageSource(diodeOff);
276                             In17Status.Text = "Input 17 OFF";
277                         }
278                     }
279                     else if (umschalt == true)
280                     {
281                         if (e.Edge == GpioPinEdge.FallingEdge)
282                         {
283                             In17Image.Source = ChangeImageSource(diodeOn);
284                             In17Status.Text = "Input 17 ON";
285                             OutPin23.Write(GpioPinValue.High);
286                             output23toggle.Content = ChangeImage(diodeOn2);
287                         }
288                         else
289                         {
290                             In17Image.Source = ChangeImageSource(diodeOff);
291                             In17Status.Text = "Input 17 OFF";
292                             OutPin23.Write(GpioPinValue.Low);
293                             output23toggle.Content = ChangeImage(diodeOff);
294                         }
295                     }
296                 }
297             });
298         }
299         private void InPin26_ValueChanged(GpioPin sender,
              GpioPinValueChangedEventArgs e)
300         {
301             var task = Dispatcher.RunAsync(CoreDispatcherPriority.Normal, ()
                  => {
302                 if (invert == false)
303                 {
304                     if (umschalt == false)
305                     {
306                         if (e.Edge == GpioPinEdge.RisingEdge)
307                         {
308                             In26Image.Source = ChangeImageSource(diodeOn);
309                             In26Status.Text = "Input 26 ON";
310                         }
311                         else
312                         {
313                             In26Image.Source = ChangeImageSource(diodeOff);
314                             In26Status.Text = "Input 26 OFF";
315                         }
316                     }
317                     else if (umschalt == true)
318                     {
319                         if (e.Edge == GpioPinEdge.RisingEdge)
320                         {
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 7

```
321                             In26Image.Source = ChangeImageSource(diodeOn);
322                             In26Status.Text = "Input 26 ON";
323                             OutPin24.Write(GpioPinValue.High);
324                             output24toggle.Content = ChangeImage(diodeOn2);
325                         }
326                         else
327                         {
328                             In26Image.Source = ChangeImageSource(diodeOff);
329                             In26Status.Text = "Input 26 OFF";
330                             OutPin24.Write(GpioPinValue.Low);
331                             output24toggle.Content = ChangeImage(diodeOff);
332                         }
333                     }
334                 }
335                 else if (invert == true)
336                 {
337                     if (umschalt == false)
338                     {
339                         if (e.Edge == GpioPinEdge.FallingEdge)
340                         {
341                             In26Image.Source = ChangeImageSource(diodeOn);
342                             In26Status.Text = "Input 26 ON";
343                         }
344                         else
345                         {
346                             In26Image.Source = ChangeImageSource(diodeOff);
347                             In26Status.Text = "Input 26 OFF";
348                         }
349                     }
350                     else if (umschalt == true)
351                     {
352                         if (e.Edge == GpioPinEdge.FallingEdge)
353                         {
354                             In26Image.Source = ChangeImageSource(diodeOn);
355                             In26Status.Text = "Input 26 ON";
356                             OutPin24.Write(GpioPinValue.High);
357                             output24toggle.Content = ChangeImage(diodeOn2);
358                         }
359                         else
360                         {
361                             In26Image.Source = ChangeImageSource(diodeOff);
362                             In26Status.Text = "Input 26 OFF";
363                             OutPin24.Write(GpioPinValue.Low);
364                             output24toggle.Content = ChangeImage(diodeOff);
365                         }
366                     }
367                 }
368             });
369         }
370         private void InPin27_ValueChanged(GpioPin sender,                    ↵
              GpioPinValueChangedEventArgs e)
371         {
372             var task = Dispatcher.RunAsync(CoreDispatcherPriority.Normal, ()   ↵
                => {
373                 if (invert == false)
374                 {
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 8

```
375                     if (umschalt == false)
376                     {
377                         if (e.Edge == GpioPinEdge.RisingEdge)
378                         {
379                             In27Image.Source = ChangeImageSource(diodeOn);
380                             In27Status.Text = "Input 27 ON";
381                         }
382                         else
383                         {
384                             In27Image.Source = ChangeImageSource(diodeOff);
385                             In27Status.Text = "Input 27 OFF";
386                         }
387                     }
388                     else if (umschalt == true)
389                     {
390                         if (e.Edge == GpioPinEdge.RisingEdge)
391                         {
392                             In27Image.Source = ChangeImageSource(diodeOn);
393                             In27Status.Text = "Input 27 ON";
394                             OutPin25.Write(GpioPinValue.High);
395                             output25toggle.Content = ChangeImage(diodeOn2);
396                         }
397                         else
398                         {
399                             In27Image.Source = ChangeImageSource(diodeOff);
400                             In27Status.Text = "Input 27 OFF";
401                             OutPin25.Write(GpioPinValue.Low);
402                             output25toggle.Content = ChangeImage(diodeOff);
403                         }
404                     }
405                 }
406                 else if (invert == true)
407                 {
408                     if (umschalt == false)
409                     {
410                         if (e.Edge == GpioPinEdge.FallingEdge)
411                         {
412                             In27Image.Source = ChangeImageSource(diodeOn);
413                             In27Status.Text = "Input 27 ON";
414                         }
415                         else
416                         {
417                             In27Image.Source = ChangeImageSource(diodeOff);
418                             In27Status.Text = "Input 27 OFF";
419                         }
420                     }
421                     else if (umschalt == true)
422                     {
423                         if (e.Edge == GpioPinEdge.FallingEdge)
424                         {
425                             In27Image.Source = ChangeImageSource(diodeOn);
426                             In27Status.Text = "Input 27 ON";
427                             OutPin25.Write(GpioPinValue.High);
428                             output25toggle.Content = ChangeImage(diodeOn2);
429                         }
430                         else
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 9

```
431                         {
432                             In27Image.Source = ChangeImageSource(diodeOff);
433                             In27Status.Text = "Input 27 OFF";
434                             OutPin25.Write(GpioPinValue.Low);
435                             output25toggle.Content = ChangeImage(diodeOff);
436                         }
437                     }
438                 }
439             });
440         }
441 
442         //If the "Input to Output" box ist checked, set all outputs
              accordingly to the inputs, also checks if the inputs were inverted.
443         private void LedTestCheck(object sender, RoutedEventArgs e)
444         {
445             umschalt = true;
446 
447             //Deactivates all output toggle buttons.
448             output22toggle.IsChecked = false;
449             output23toggle.IsChecked = false;
450             output24toggle.IsChecked = false;
451             output25toggle.IsChecked = false;
452             output22toggle.IsEnabled = false;
453             output23toggle.IsEnabled = false;
454             output24toggle.IsEnabled = false;
455             output25toggle.IsEnabled = false;
456 
457             output22toggle.Content = ChangeImage(diodeOff);
458             output23toggle.Content = ChangeImage(diodeOff);
459             output24toggle.Content = ChangeImage(diodeOff);
460             output25toggle.Content = ChangeImage(diodeOff);
461 
462             if (invert == false)
463             {
464                 if (InPin16.Read() == GpioPinValue.High)
465                 {
466                     OutPin22.Write(GpioPinValue.High);
467                     output22toggle.Content = ChangeImage(diodeOn2);
468                 }
469                 if (InPin17.Read() == GpioPinValue.High)
470                 {
471                     OutPin23.Write(GpioPinValue.High);
472                     output23toggle.Content = ChangeImage(diodeOn2);
473                 }
474                 if (InPin26.Read() == GpioPinValue.High)
475                 {
476                     OutPin24.Write(GpioPinValue.High);
477                     output24toggle.Content = ChangeImage(diodeOn2);
478                 }
479                 if (InPin27.Read() == GpioPinValue.High)
480                 {
481                     OutPin25.Write(GpioPinValue.High);
482                     output25toggle.Content = ChangeImage(diodeOn2);
483                 }
484             }
485             else if (invert == true)
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 10

```
486             {
487                 if (InPin16.Read() == GpioPinValue.Low)
488                 {
489                     OutPin22.Write(GpioPinValue.High);
490                     output22toggle.Content = ChangeImage(diodeOn2);
491                 }
492                 if (InPin17.Read() == GpioPinValue.Low)
493                 {
494                     OutPin23.Write(GpioPinValue.High);
495                     output23toggle.Content = ChangeImage(diodeOn2);
496                 }
497                 if (InPin26.Read() == GpioPinValue.Low)
498                 {
499                     OutPin24.Write(GpioPinValue.High);
500                     output24toggle.Content = ChangeImage(diodeOn2);
501                 }
502                 if (InPin27.Read() == GpioPinValue.Low)
503                 {
504                     OutPin25.Write(GpioPinValue.High);
505                     output25toggle.Content = ChangeImage(diodeOn2);
506                 }
507             }
508         }
509
510         //If the "Input to Output" box ist unchecked, write the value low to
              all outputs and enable the corresponding toggle buttons.
511         private void LedTestUnchecked(object sender, RoutedEventArgs e)
512         {
513             umschalt = false;
514
515             OutPin22.Write(GpioPinValue.Low);
516             OutPin23.Write(GpioPinValue.Low);
517             OutPin24.Write(GpioPinValue.Low);
518             OutPin25.Write(GpioPinValue.Low);
519
520             output22toggle.Content = ChangeImage(switchOff);
521             output23toggle.Content = ChangeImage(switchOff);
522             output24toggle.Content = ChangeImage(switchOff);
523             output25toggle.Content = ChangeImage(switchOff);
524
525             output22toggle.IsEnabled = true;
526             output23toggle.IsEnabled = true;
527             output24toggle.IsEnabled = true;
528             output25toggle.IsEnabled = true;
529         }
530
531         //If the invert box is checked, invert all values and update the
              graphics.
532         private void InvertCheck(object sender, RoutedEventArgs e)
533         {
534             invert = true;
535
536             if (InPin16.Read() == GpioPinValue.High)
537             {
538                 if (umschalt == true)
539                 {
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 11

```
540                     OutPin22.Write(GpioPinValue.Low);
541                     output22toggle.Content = ChangeImage(diodeOff);
542                 }
543                 In16Image.Source = ChangeImageSource(diodeOff);
544                 In16Status.Text = "Input 16 OFF";
545             }
546             else if (InPin16.Read() == GpioPinValue.Low)
547             {
548                 if (umschalt == true)
549                 {
550                     OutPin22.Write(GpioPinValue.High);
551                     output22toggle.Content = ChangeImage(diodeOn2);
552                 }
553                 In16Image.Source = ChangeImageSource(diodeOn);
554                 In16Status.Text = "Input 16 ON";
555             }
556             if (InPin17.Read() == GpioPinValue.High)
557             {
558                 if (umschalt == true)
559                 {
560                     OutPin23.Write(GpioPinValue.Low);
561                     output23toggle.Content = ChangeImage(diodeOff);
562                 }
563                 In17Image.Source = ChangeImageSource(diodeOff);
564                 In17Status.Text = "Input 17 OFF";
565             }
566             else if (InPin17.Read() == GpioPinValue.Low)
567             {
568                 if (umschalt == true)
569                 {
570                     OutPin23.Write(GpioPinValue.High);
571                     output23toggle.Content = ChangeImage(diodeOn2);
572                 }
573                 In17Image.Source = ChangeImageSource(diodeOn);
574                 In17Status.Text = "Input 17 ON";
575             }
576             if (InPin26.Read() == GpioPinValue.High)
577             {
578                 if (umschalt == true)
579                 {
580                     OutPin24.Write(GpioPinValue.Low);
581                     output24toggle.Content = ChangeImage(diodeOff);
582                 }
583                 In26Image.Source = ChangeImageSource(diodeOff);
584                 In26Status.Text = "Input 26 OFF";
585             }
586             else if (InPin26.Read() == GpioPinValue.Low)
587             {
588                 if (umschalt == true)
589                 {
590                     OutPin24.Write(GpioPinValue.High);
591                     output24toggle.Content = ChangeImage(diodeOn2);
592                 }
593                 In26Image.Source = ChangeImageSource(diodeOn);
594                 In26Status.Text = "Input 26 ON";
595             }
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 12

```
596             if (InPin27.Read() == GpioPinValue.High)
597             {
598                 if (umschalt == true)
599                 {
600                     OutPin25.Write(GpioPinValue.Low);
601                     output25toggle.Content = ChangeImage(diodeOff);
602                 }
603                 In27Image.Source = ChangeImageSource(diodeOff);
604                 In27Status.Text = "Input 27 OFF";
605             }
606             else if (InPin27.Read() == GpioPinValue.Low)
607             {
608                 if (umschalt == true)
609                 {
610                     OutPin25.Write(GpioPinValue.High);
611                     output25toggle.Content = ChangeImage(diodeOn2);
612                 }
613                 In27Image.Source = ChangeImageSource(diodeOn);
614                 In27Status.Text = "Input 27 ON";
615             }
616         }
617
618         //If the invert box is unchecked, revert all values and graphics
              accordingly.
619         private void InvertUnchecked(object sender, RoutedEventArgs e)
620         {
621             invert = false;
622             if (InPin16.Read() == GpioPinValue.High)
623             {
624                 if (umschalt == true)
625                 {
626                     OutPin22.Write(GpioPinValue.High);
627                     output22toggle.Content = ChangeImage(diodeOn2);
628                 }
629                 In16Image.Source = ChangeImageSource(diodeOn);
630                 In16Status.Text = "Input 16 ON";
631             }
632             else if (InPin16.Read() == GpioPinValue.Low)
633             {
634                 if (umschalt == true)
635                 {
636                     OutPin22.Write(GpioPinValue.Low);
637                     output22toggle.Content = ChangeImage(diodeOff);
638                 }
639                 In16Image.Source = ChangeImageSource(diodeOff);
640                 In16Status.Text = "Input 16 OFF";
641             }
642             if (InPin17.Read() == GpioPinValue.High)
643             {
644                 if (umschalt == true)
645                 {
646                     OutPin23.Write(GpioPinValue.High);
647                     output23toggle.Content = ChangeImage(diodeOn2);
648                 }
649                 In17Image.Source = ChangeImageSource(diodeOn);
650                 In17Status.Text = "Input 17 ON";
```

```
651             }
652             else if (InPin17.Read() == GpioPinValue.Low)
653             {
654                 if (umschalt == true)
655                 {
656                     OutPin23.Write(GpioPinValue.Low);
657                     output23toggle.Content = ChangeImage(diodeOff);
658                 }
659                 In17Image.Source = ChangeImageSource(diodeOff);
660                 In17Status.Text = "Input 17 OFF";
661             }
662             if (InPin26.Read() == GpioPinValue.High)
663             {
664                 if (umschalt == true)
665                 {
666                     OutPin24.Write(GpioPinValue.High);
667                     output24toggle.Content = ChangeImage(diodeOn2);
668                 }
669                 In26Image.Source = ChangeImageSource(diodeOn);
670                 In26Status.Text = "Input 26 ON";
671             }
672             else if (InPin26.Read() == GpioPinValue.Low)
673             {
674                 if (umschalt == true)
675                 {
676                     OutPin24.Write(GpioPinValue.Low);
677                     output24toggle.Content = ChangeImage(diodeOff);
678                 }
679                 In26Image.Source = ChangeImageSource(diodeOff);
680                 In26Status.Text = "Input 26 OFF";
681             }
682             if (InPin27.Read() == GpioPinValue.High)
683             {
684                 if (umschalt == true)
685                 {
686                     OutPin25.Write(GpioPinValue.High);
687                     output25toggle.Content = ChangeImage(diodeOn2);
688                 }
689                 In27Image.Source = ChangeImageSource(diodeOn);
690                 In27Status.Text = "Input 27 ON";
691             }
692             else if (InPin27.Read() == GpioPinValue.Low)
693             {
694                 if (umschalt == true)
695                 {
696                     OutPin25.Write(GpioPinValue.Low);
697                     output25toggle.Content = ChangeImage(diodeOff);
698                 }
699                 In27Image.Source = ChangeImageSource(diodeOff);
700                 In27Status.Text = "Input 27 OFF";
701             }
702         }
703 
704         //All toggle button checks for every output.
705         private void OUTPUT22Check(object sender, RoutedEventArgs e)
706         {
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 14

```
707             OutPin22.Write(GpioPinValue.High);
708             output22toggle.Content = ChangeImage(switchOn);
709         }
710         private void OUTPUT22Unchecked(object sender, RoutedEventArgs e)
711         {
712             OutPin22.Write(GpioPinValue.Low);
713             output22toggle.Content = ChangeImage(switchOff);
714         }
715         private void OUTPUT23Check(object sender, RoutedEventArgs e)
716         {
717             OutPin23.Write(GpioPinValue.High);
718             output23toggle.Content = ChangeImage(switchOn);
719         }
720         private void OUTPUT23Unchecked(object sender, RoutedEventArgs e)
721         {
722             OutPin23.Write(GpioPinValue.Low);
723             output23toggle.Content = ChangeImage(switchOff);
724         }
725         private void OUTPUT24Check(object sender, RoutedEventArgs e)
726         {
727             OutPin24.Write(GpioPinValue.High);
728             output24toggle.Content = ChangeImage(switchOn);
729         }
730         private void OUTPUT24Unchecked(object sender, RoutedEventArgs e)
731         {
732             OutPin24.Write(GpioPinValue.Low);
733             output24toggle.Content = ChangeImage(switchOff);
734         }
735         private void OUTPUT25Check(object sender, RoutedEventArgs e)
736         {
737             OutPin25.Write(GpioPinValue.High);
738             output25toggle.Content = ChangeImage(switchOn);
739         }
740         private void OUTPUT25Unchecked(object sender, RoutedEventArgs e)
741         {
742             OutPin25.Write(GpioPinValue.Low);
743             output25toggle.Content = ChangeImage(switchOff);
744         }
745
746         //Loads a new image and updates the existing one.
747         private Image ChangeImage(String destination)
748         {
749             Image myImage = new Image();
750             BitmapImage biImage = new BitmapImage();
751             biImage.UriSource = new Uri(destination, UriKind.Absolute);
752             myImage.Stretch = Stretch.Uniform;
753             myImage.Margin = new Thickness(-25, -25, -25, -25);
754             myImage.Source = biImage;
755             return myImage;
756         }
757
758         //Loads a new image and updates the existing one, specifically for the
               image type in XAML.
759         private ImageSource ChangeImageSource(String destination)
760         {
761             BitmapImage biImage = new BitmapImage();
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 15

```
762             biImage.UriSource = new Uri(destination, UriKind.Absolute);
763             return biImage;
764         }
765
766         //Closes all pins and set them free.
767         public void Release()
768         {
769             if (OutPin22 != null)
770             {
771                 OutPin22.Dispose();
772             }
773             if (OutPin23 != null)
774             {
775                 OutPin23.Dispose();
776             }
777             if (OutPin24 != null)
778             {
779                 OutPin24.Dispose();
780             }
781             if (OutPin25 != null)
782             {
783                 OutPin25.Dispose();
784             }
785             if (InPin16 != null)
786             {
787                 InPin16.Dispose();
788             }
789             if (InPin17 != null)
790             {
791                 InPin17.Dispose();
792             }
793             if (InPin26 != null)
794             {
795                 InPin26.Dispose();
796             }
797             if (InPin27 != null)
798             {
799                 InPin27.Dispose();
800             }
801         }
802
803         //Activates the release of the pins and activates the garbage
              collection, when the page is left.
804         protected override void OnNavigatedFrom(NavigationEventArgs e)
805         {
806             base.OnNavigatedFrom(e);
807             Release();
808             GC.Collect();
809         }
810
811         //Starts the initialization, when the page is opened.
812         protected override void OnNavigatedTo(NavigationEventArgs e)
813         {
814             base.OnNavigatedTo(e);
815             InitGPIO();
816         }
```

...\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml.cs 16

```
817
818         //All needed variables are found here, you can change the values if
              you need other pins.
819         private const int LED_22 = 22;
820         private const int LED_23 = 23;
821         private const int LED_24 = 24;
822         private const int LED_25 = 25;
823         private const int SWITCH_16 = 16;
824         private const int SWITCH_17 = 17;
825         private const int SWITCH_26 = 26;
826         private const int SWITCH_27 = 27;
827         private GpioPin OutPin22;
828         private GpioPin OutPin23;
829         private GpioPin OutPin24;
830         private GpioPin OutPin25;
831         private GpioPin InPin16;
832         private GpioPin InPin17;
833         private GpioPin InPin26;
834         private GpioPin InPin27;
835         private bool umschalt = false;
836         private bool invert = false;
837         private bool error = false;
838
839         //File paths of the images are stored here.
840         private const string diodeOn = "ms-appx:///GpioAssets/diodeOn.png";
841         private const string diodeOff = "ms-appx:///GpioAssets/diodeOff.png";
842         private const string diodeOn2 = "ms-appx:///GpioAssets/diodeOn2.png";
843         private const string switchOn = "ms-appx:///GpioAssets/switchOn.png";
844         private const string switchOff = "ms-appx:///GpioAssets/
              switchOff.png";
845     }
846 }
847
```

...pos\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml 1

```
<Page
    x:Class="MASS_Platinen_Test.GpioPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MASS_Platinen_Test"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="Transparent">

    <!-- Author: Daniel Heintze -->

    <!-- Changes the style of the toggle buttons, to not show a shadow if pressed.
      -->
    <Page.Resources>
        <Style x:Key="ToggleButtonStyle1" TargetType="ToggleButton">
            <Setter Property="Background" Value="{ThemeResource
              ToggleButtonBackground}"/>
            <Setter Property="Foreground" Value="{ThemeResource
              ToggleButtonForeground}"/>
            <Setter Property="BorderBrush" Value="{ThemeResource
              ToggleButtonBorderBrush}"/>
            <Setter Property="BorderThickness" Value="{ThemeResource
              ToggleButtonBorderThemeThickness}"/>
            <Setter Property="Padding" Value="{StaticResource ButtonPadding}"/>
            <Setter Property="HorizontalAlignment" Value="Left"/>
            <Setter Property="VerticalAlignment" Value="Center"/>
            <Setter Property="FontFamily" Value="{ThemeResource
              ContentControlThemeFontFamily}"/>
            <Setter Property="FontWeight" Value="Normal"/>
            <Setter Property="FontSize" Value="{ThemeResource
              ControlContentThemeFontSize}"/>
            <Setter Property="UseSystemFocusVisuals" Value="{StaticResource
              UseSystemFocusVisuals}"/>
            <Setter Property="FocusVisualMargin" Value="-3"/>
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="ToggleButton">
                        <ContentPresenter x:Name="ContentPresenter"
                     AutomationProperties.AccessibilityView="Raw"
                     Background="{TemplateBinding Background}"
                     BorderThickness="{TemplateBinding BorderThickness}"
                     BorderBrush="{TemplateBinding BorderBrush}"
                     ContentTemplate="{TemplateBinding ContentTemplate}"
                     Content="{TemplateBinding Content}"
                     CornerRadius="{TemplateBinding CornerRadius}"
                     ContentTransitions="{TemplateBinding ContentTransitions}"
                     HorizontalContentAlignment="{TemplateBinding
                     HorizontalContentAlignment}" Padding="{TemplateBinding
                     Padding}" VerticalContentAlignment="{TemplateBinding
                     VerticalContentAlignment}">
                            <VisualStateManager.VisualStateGroups>
                                <VisualStateGroup x:Name="CommonStates">
                                    <VisualState x:Name="Normal">
                                        <Storyboard>
                                            <PointerUpThemeAnimation
```

```
                    Storyboard.TargetName="ContentPresenter"/>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="PointerOver">
                                <Storyboard>
                                    <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Background">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource ToggleButtonBackgroundPointerOver}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="BorderBrush">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource ToggleButtonBorderBrushPointerOver}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Foreground">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource ToggleButtonForegroundPointerOver}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <PointerUpThemeAnimation
                    Storyboard.TargetName="ContentPresenter"/>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="Pressed">
                                <Storyboard>
                                    <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Background">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource ToggleButtonBackgroundPressed}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="BorderBrush">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource ToggleButtonBorderBrushPressed}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Foreground">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource ToggleButtonForegroundPressed}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <PointerDownThemeAnimation
                    Storyboard.TargetName="ContentPresenter"/>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="Disabled">
                                <Storyboard>
                                    <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Background">
```

...pos\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml 3

```
                                        <DiscreteObjectKeyFrame KeyTime="0"
 Value="Transparent"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <ObjectAnimationUsingKeyFrames
Storyboard.TargetName="ContentPresenter"
Storyboard.TargetProperty="Foreground">
                                        <DiscreteObjectKeyFrame KeyTime="0"
 Value="Transparent"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <ObjectAnimationUsingKeyFrames
Storyboard.TargetName="ContentPresenter"
Storyboard.TargetProperty="BorderBrush">
                                        <DiscreteObjectKeyFrame KeyTime="0"
 Value="Transparent"/>
                                    </ObjectAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="Checked">
                                <Storyboard>
                                    <ObjectAnimationUsingKeyFrames
Storyboard.TargetName="ContentPresenter"
Storyboard.TargetProperty="Background">
                                        <DiscreteObjectKeyFrame KeyTime="0"
 Value="Transparent"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <ObjectAnimationUsingKeyFrames
Storyboard.TargetName="ContentPresenter"
Storyboard.TargetProperty="Foreground">
                                        <DiscreteObjectKeyFrame KeyTime="0"
 Value="Transparent"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <ObjectAnimationUsingKeyFrames
Storyboard.TargetName="ContentPresenter"
Storyboard.TargetProperty="BorderBrush">
                                        <DiscreteObjectKeyFrame KeyTime="0"
 Value="Transparent"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <PointerUpThemeAnimation
Storyboard.TargetName="ContentPresenter"/>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="CheckedPointerOver">
                                <Storyboard>
                                    <ObjectAnimationUsingKeyFrames
Storyboard.TargetName="ContentPresenter"
Storyboard.TargetProperty="Background">
                                        <DiscreteObjectKeyFrame KeyTime="0"
 Value="{ThemeResource
ToggleButtonBackgroundCheckedPointerOver}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <ObjectAnimationUsingKeyFrames
Storyboard.TargetName="ContentPresenter"
Storyboard.TargetProperty="BorderBrush">
                                        <DiscreteObjectKeyFrame KeyTime="0"
 Value="{ThemeResource
ToggleButtonBorderBrushCheckedPointerOver}"/>
```

```
                                </ObjectAnimationUsingKeyFrames>
                                <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Foreground">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource
                    ToggleButtonForegroundCheckedPointerOver}"/>
                                </ObjectAnimationUsingKeyFrames>
                                <PointerUpThemeAnimation
                    Storyboard.TargetName="ContentPresenter"/>
                            </Storyboard>
                        </VisualState>
                        <VisualState x:Name="CheckedPressed">
                            <Storyboard>
                                <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Background">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource ToggleButtonBackgroundCheckedPressed}"/
                    >
                                </ObjectAnimationUsingKeyFrames>
                                <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Foreground">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource ToggleButtonForegroundCheckedPressed}"/
                    >
                                </ObjectAnimationUsingKeyFrames>
                                <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="BorderBrush">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource
                    ToggleButtonBorderBrushCheckedPressed}"/>
                                </ObjectAnimationUsingKeyFrames>
                                <PointerDownThemeAnimation
                    Storyboard.TargetName="ContentPresenter"/>
                            </Storyboard>
                        </VisualState>
                        <VisualState x:Name="CheckedDisabled">
                            <Storyboard>
                                <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Background">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource
                    ToggleButtonBackgroundCheckedDisabled}"/>
                                </ObjectAnimationUsingKeyFrames>
                                <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Foreground">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource
                    ToggleButtonForegroundCheckedDisabled}"/>
                                </ObjectAnimationUsingKeyFrames>
                                <ObjectAnimationUsingKeyFrames
```

```
...pos\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml                    5
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="BorderBrush">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource
                    ToggleButtonBorderBrushCheckedDisabled}"/>
                                </ObjectAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                        <VisualState x:Name="Indeterminate">
                            <Storyboard>
                                <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Background">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource ToggleButtonBackgroundIndeterminate}"/>
                                </ObjectAnimationUsingKeyFrames>
                                <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Foreground">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource ToggleButtonForegroundIndeterminate}"/>
                                </ObjectAnimationUsingKeyFrames>
                                <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="BorderBrush">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource ToggleButtonBorderBrushIndeterminate}"/
                    >
                                </ObjectAnimationUsingKeyFrames>
                                <PointerUpThemeAnimation
                    Storyboard.TargetName="ContentPresenter"/>
                            </Storyboard>
                        </VisualState>
                        <VisualState x:Name="IndeterminatePointerOver">
                            <Storyboard>
                                <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Background">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource
                    ToggleButtonBackgroundIndeterminatePointerOver}"/>
                                </ObjectAnimationUsingKeyFrames>
                                <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="BorderBrush">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource
                    ToggleButtonBorderBrushIndeterminatePointerOver}"/>
                                </ObjectAnimationUsingKeyFrames>
                                <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetName="ContentPresenter"
                    Storyboard.TargetProperty="Foreground">
                                    <DiscreteObjectKeyFrame KeyTime="0"
                     Value="{ThemeResource
                    ToggleButtonForegroundIndeterminatePointerOver}"/>
                                </ObjectAnimationUsingKeyFrames>
```

...pos\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml 6

```
                                    <PointerUpThemeAnimation
                Storyboard.TargetName="ContentPresenter"/>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="IndeterminatePressed">
                                <Storyboard>
                                    <ObjectAnimationUsingKeyFrames
                Storyboard.TargetName="ContentPresenter"
                Storyboard.TargetProperty="Background">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                 Value="{ThemeResource
                ToggleButtonBackgroundIndeterminatePressed}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <ObjectAnimationUsingKeyFrames
                Storyboard.TargetName="ContentPresenter"
                Storyboard.TargetProperty="BorderBrush">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                 Value="{ThemeResource
                ToggleButtonBorderBrushIndeterminatePressed}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <ObjectAnimationUsingKeyFrames
                Storyboard.TargetName="ContentPresenter"
                Storyboard.TargetProperty="Foreground">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                 Value="{ThemeResource
                ToggleButtonForegroundIndeterminatePressed}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <PointerDownThemeAnimation
                Storyboard.TargetName="ContentPresenter"/>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="IndeterminateDisabled">
                                <Storyboard>
                                    <ObjectAnimationUsingKeyFrames
                Storyboard.TargetName="ContentPresenter"
                Storyboard.TargetProperty="Background">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                 Value="{ThemeResource
                ToggleButtonBackgroundIndeterminateDisabled}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <ObjectAnimationUsingKeyFrames
                Storyboard.TargetName="ContentPresenter"
                Storyboard.TargetProperty="Foreground">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                 Value="{ThemeResource
                ToggleButtonForegroundIndeterminateDisabled}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                    <ObjectAnimationUsingKeyFrames
                Storyboard.TargetName="ContentPresenter"
                Storyboard.TargetProperty="BorderBrush">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                 Value="{ThemeResource
                ToggleButtonBorderBrushIndeterminateDisabled}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
```

...pos\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml 7

```
                                </VisualStateGroup>
                            </VisualStateManager.VisualStateGroups>
                        </ContentPresenter>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
    </Page.Resources>

    <Grid>
        <StackPanel HorizontalAlignment="Left" VerticalAlignment="Center"
          Margin="10,60,0,0">
            <Image x:Name="In16Image" Source="GpioAssets/diodeOff.png" Width="150"
              Height="100" Stretch="Uniform"></Image>
            <TextBlock x:Name="In16Status" Text="Input 16 Init"
              Margin="20,10,10,10" TextAlignment="Left" FontSize="20" />
        </StackPanel>
        <StackPanel HorizontalAlignment="Left" VerticalAlignment="Center"
          Margin="185,60,0,0">
            <Image x:Name="In17Image" Source="GpioAssets/diodeOff.png" Width="150"
              Height="100" Stretch="Uniform"></Image>
            <TextBlock x:Name="In17Status" Text="Input 17 Init"
              Margin="20,10,10,10" TextAlignment="Left" FontSize="20" />
        </StackPanel>
        <StackPanel HorizontalAlignment="Left" VerticalAlignment="Center"
          Margin="360,60,0,0">
            <Image x:Name="In26Image" Source="GpioAssets/diodeOff.png" Width="150"
              Height="100" Stretch="Uniform"></Image>
            <TextBlock x:Name="In26Status" Text="Input 26 Init"
              Margin="20,10,10,10" TextAlignment="Left" FontSize="20" />
        </StackPanel>
        <StackPanel HorizontalAlignment="Left" VerticalAlignment="Center"
          Margin="535,60,0,0">
            <Image x:Name="In27Image" Source="GpioAssets/diodeOff.png" Width="150"
              Height="100" Stretch="Uniform"></Image>
            <TextBlock x:Name="In27Status" Text="Input 27 Init"
              Margin="20,10,10,10" TextAlignment="Left" FontSize="20" />
        </StackPanel>

        <StackPanel HorizontalAlignment="Right" VerticalAlignment="Top"
          Margin="0,10,10,0">
            <CheckBox x:Name = "cb1"
            Content = "input -> Output"
            Checked = "LedTestCheck"
            Unchecked = "LedTestUnchecked"
            Margin = "5"
            FontSize="14"/>

            <CheckBox x:Name = "cb2"
            Content = "Invert input"
            Checked = "InvertCheck"
            Unchecked = "InvertUnchecked"
            Margin = "5"
            FontSize="14"/>
        </StackPanel>
```

...pos\MASS_Platinen_Test\MASS_Platinen_Test\GpioPage.xaml 8

```
        <StackPanel HorizontalAlignment="Center" VerticalAlignment="Bottom">
            <TextBlock x:Name="GpioStatus" Text="Initializing GPIO..."
              Margin="10,50,10,10" TextAlignment="Center" FontSize="20"
              Width="500" />
            <TextBlock x:Name="ErrorHandling" Text="" Margin="10,50,10,10"
              TextAlignment="Center" FontSize="20" Width="500"
              Visibility="Collapsed"/>
        </StackPanel>

        <StackPanel HorizontalAlignment="Left" VerticalAlignment="Top"
          Orientation="Horizontal"  Margin="25,10,0,0" Height="100">
            <ToggleButton x:Name = "output22toggle"
            Checked = "OUTPUT22Check" Unchecked = "OUTPUT22Unchecked"
            Margin = "10" Width="100" Height="75" Background="Transparent"
              Style="{StaticResource ToggleButtonStyle1}">
                <Image Source="GpioAssets/switchOff.png" Stretch="Uniform"
                  Margin="-25,-25,-25,-25"/>
            </ToggleButton>
            <ToggleButton x:Name = "output23toggle"
            Checked = "OUTPUT23Check" Unchecked = "OUTPUT23Unchecked"
            Margin = "10" Width="100" Height="75" Background="Transparent"
              Style="{StaticResource ToggleButtonStyle1}">
                <Image Source="GpioAssets/switchOff.png" Stretch="Uniform"
                  Margin="-25,-25,-25,-25"/>
            </ToggleButton>
            <ToggleButton x:Name = "output24toggle"
            Checked = "OUTPUT24Check" Unchecked = "OUTPUT24Unchecked"
            Margin = "10" Width = "100" Height="75" Background="Transparent"
              Style="{StaticResource ToggleButtonStyle1}">
                <Image Source="GpioAssets/switchOff.png" Stretch="Uniform"
                  Margin="-25,-25,-25,-25"/>
            </ToggleButton>
            <ToggleButton x:Name = "output25toggle"
            Checked = "OUTPUT25Check" Unchecked = "OUTPUT25Unchecked"
            Margin = "10" Width = "100" Height="75" Background="Transparent"
              Style="{StaticResource ToggleButtonStyle1}">
                <Image Source="GpioAssets/switchOff.png" Stretch="Uniform"
                  Margin="-25,-25,-25,-25"/>
            </ToggleButton>
        </StackPanel>

        <StackPanel HorizontalAlignment="Left" VerticalAlignment="Top"
          Orientation="Horizontal"  Margin="25,100,0,0" Height="50">
            <TextBlock Text="Output 22" Margin="15,10,10,10" TextAlignment="Center"
               FontSize="20"/>
            <TextBlock Text="Output 23" Margin="20,10,10,10" TextAlignment="Center"
               FontSize="20"/>
            <TextBlock Text="Output 24" Margin="20,10,10,10" TextAlignment="Center"
               FontSize="20"/>
            <TextBlock Text="Output 25" Margin="20,10,10,10" TextAlignment="Center"
               FontSize="20"/>
        </StackPanel>
    </Grid>
</Page>
```

...s\MASS_Platinen_Test\MASS_Platinen_Test\I2cPage.xaml.cs 1

```
1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4  using System.Linq;
5  using System.Runtime.InteropServices.WindowsRuntime;
6  using System.Threading.Tasks;
7  using System.Timers;
8  using Windows.Devices.Enumeration;
9  using Windows.Devices.I2c;
10 using Windows.Foundation;
11 using Windows.Foundation.Collections;
12 using Windows.UI.Core;
13 using Windows.UI.Xaml;
14 using Windows.UI.Xaml.Controls;
15 using Windows.UI.Xaml.Controls.Primitives;
16 using Windows.UI.Xaml.Data;
17 using Windows.UI.Xaml.Input;
18 using Windows.UI.Xaml.Media;
19 using Windows.UI.Xaml.Navigation;
20
21 namespace MASS_Platinen_Test
22 {
23     /// <summary>
24     /// I2cPage: This page contains the test for the i2c bus.
25     /// It scans the i2c bus from address 0x03 to 0x77 to find all devices
       which are connected to the bus.
26     /// The found i2c devices are then displayed in a list, with their found
       address.
27     /// Also you can set the date and time for the connected real time clock
       on address 0x68.
28     /// Author: Daniel Heintze
29     /// </summary>
30     public sealed partial class I2cPage : Page
31     {
32         public I2cPage()
33         {
34             this.InitializeComponent();
35         }
36
37         //Starts an asynchronous task when called, this task scans an adress
           range and returns the found adresses.
38         public async Task<IEnumerable<byte>> FindDevicesAsync()
39         {
40             IList<byte> returnValue = new List<byte>();
41             // ***
42             // *** Get a selector string that will return all I2C controllers
                 on the system
43             // ***
44             string aqs = I2cDevice.GetDeviceSelector();
45             // ***
46             // *** Find the I2C bus controller device with our selector string
47             // ***
48             var dis = await DeviceInformation.FindAllAsync(aqs).AsTask();
49             if (dis.Count > 0)
50             {
```

...s\MASS_Platinen_Test\MASS_Platinen_Test\I2cPage.xaml.cs 2

```
51                  const int minimumAddress = 0x03;
52                  const int maximumAddress = 0x77;
53
54                  int i = 0;
55
56                  for (byte address = minimumAddress; address <= maximumAddress;
                        address++)
57                  {
58                      var settings = new I2cConnectionSettings(address);
59                      settings.BusSpeed = I2cBusSpeed.FastMode;
60                      settings.SharingMode = I2cSharingMode.Shared;
61                      // ***
62                      // *** Create an I2cDevice with our selected bus
                        controller and I2C settings
63                      // ***
64                      using (I2cDevice device = await I2cDevice.FromIdAsync(dis
                        [0].Id, settings))
65                      {
66                          if (device != null)
67                          {
68                              try
69                              {
70                                  byte[] writeBuffer = new byte[1] { 0 };
71                                  device.Write(writeBuffer);
72                                  // ***
73                                  // *** If no exception is thrown, there is
74                                  // *** a device at this address.
75                                  // ***
76                                  returnValue.Add(address);
77                              }
78                              catch
79                              {
80                                  // ***
81                                  // *** If the address is invalid, an exception
                         will be thrown.
82                                  // ***
83                              }
84                          }
85                      }
86                      //this was inserted by me, to update a progress bar with
                        the variable "i", which is incremented by 1 for every
                        adress that was searched.
87                      ProgressValue.Value = i;
88                      i++;
89                  }
90              }
91              return returnValue;
92          }
93
94          //Activates the garbage collection, when the page is left.
95          protected override void OnNavigatedFrom(NavigationEventArgs e)
96          {
97              base.OnNavigatedFrom(e);
98              if (aTimer != null)
99              {
100                 aTimer.Stop();
```

...s\MASS_Platinen_Test\MASS_Platinen_Test\I2cPage.xaml.cs 3

```
101                 aTimer.Dispose();
102             }
103             GC.Collect();
104         }
105
106         //Starts the search for addresses on the I²C Bus and adds found
              addresses to a list.
107         protected override void OnNavigatedTo(NavigationEventArgs e)
108         {
109             base.OnNavigatedTo(e);
110
111             try
112             {
113                 var task = Dispatcher.RunAsync(CoreDispatcherPriority.Normal,
                    async () =>
114                 {
115                     IEnumerable<byte> address = await FindDevicesAsync();
116
117                     if (address.Count() > 0)
118                     {
119                         Status.Text = string.Format("{0} device(s) found.",
                        address.Count());
120
121                         for (int i = 0; i < address.Count(); i++)
122                         {
123                             listView1.Items.Add(string.Format("Device {0}:
                        Adress: 0x{1:X}", i + 1, address.ElementAt(i)));
124                         }
125                     }
126
127                     SetTimer();
128                     setRtcTime.Visibility = Visibility.Visible;
129                 });
130             }
131             catch (Exception ex)
132             {
133                 Status.Text = string.Format("Error: {0}", ex);
134             }
135         }
136
137         //Sets and starts the timer with a 1 second interval.
138         private void SetTimer()
139         {
140             // Create a timer with a 1 second interval.
141             aTimer = new System.Timers.Timer(1000);
142             // Hook up the Elapsed event for the timer.
143             aTimer.Elapsed += OnTimedEvent;
144             aTimer.AutoReset = true;
145             aTimer.Enabled = true;
146         }
147
148         //Every 1 second, update the date and time
149         private async void OnTimedEvent(Object source, ElapsedEventArgs e)
150         {
151             string aqs = I2cDevice.GetDeviceSelector();
152             var dis = await DeviceInformation.FindAllAsync(aqs).AsTask();
```

...s\MASS_Platinen_Test\MASS_Platinen_Test\I2cPage.xaml.cs 4

```
153             var settings = new I2cConnectionSettings(0x68);
154             settings.BusSpeed = I2cBusSpeed.FastMode;
155             settings.SharingMode = I2cSharingMode.Shared;
156
157             I2cDevice device = await I2cDevice.FromIdAsync(dis[0].Id,
                  settings);
158
159             byte[] readBuffer = new byte[7];
160             device.WriteRead(new byte[] { 0x00 }, readBuffer);
161
162             DateTime rtcDateTime = ConvertByteBufferToDateTime(readBuffer);
163
164             var task = Dispatcher.RunAsync(CoreDispatcherPriority.Normal, ()
                  => {
165                 i2cClock.Text = string.Format("Real Time Clock on 0x68: {0}",
                      rtcDateTime);
166             });
167         }
168
169         //The following 4 methods are converting the values to appropriate
              types, else the values couldnt be used like intended.
170         private int BinaryCodedDecimalToInteger(int value)
171         {
172             var lowerNibble = value & 0x0F;
173             var upperNibble = value >> 4;
174
175             var multipleOfOne = lowerNibble;
176             var multipleOfTen = upperNibble * 10;
177
178             return multipleOfOne + multipleOfTen;
179         }
180
181         private DateTime ConvertByteBufferToDateTime(byte[] dateTimeBuffer)
182         {
183             var second = BinaryCodedDecimalToInteger(dateTimeBuffer[0]);
184             var minute = BinaryCodedDecimalToInteger(dateTimeBuffer[1]);
185             var hour = BinaryCodedDecimalToInteger(dateTimeBuffer[2]);
186             var dayofWeek = BinaryCodedDecimalToInteger(dateTimeBuffer[3]);
187             var day = BinaryCodedDecimalToInteger(dateTimeBuffer[4]);
188             var month = BinaryCodedDecimalToInteger(dateTimeBuffer[5]);
189             var year = 2000 + BinaryCodedDecimalToInteger(dateTimeBuffer[6]);
190
191             return new DateTime(year, month, day, hour, minute, second);
192         }
193
194         private byte IntegerToBinaryCodedDecimal(int value)
195         {
196             var multipleOfOne = value % 10;
197             var multipleOfTen = value / 10;
198
199             // convert to nibbles
200             var lowerNibble = multipleOfOne;
201             var upperNibble = multipleOfTen << 4;
202
203             return (byte)(lowerNibble + upperNibble);
204         }
```

...s\MASS_Platinen_Test\MASS_Platinen_Test\I2cPage.xaml.cs 5

```
205
206         private byte[] ConvertTimeToByteArray(DateTime dateTime)
207         {
208             var dateTimeByteArray = new byte[8];
209
210             dateTimeByteArray[0] = 0;
211             dateTimeByteArray[1] = IntegerToBinaryCodedDecimal
                  (dateTime.Second);
212             dateTimeByteArray[2] = IntegerToBinaryCodedDecimal
                  (dateTime.Minute);
213             dateTimeByteArray[3] = IntegerToBinaryCodedDecimal(dateTime.Hour);
214             dateTimeByteArray[4] = IntegerToBinaryCodedDecimal((byte)
                  dateTime.DayOfWeek);
215             dateTimeByteArray[5] = IntegerToBinaryCodedDecimal(dateTime.Day);
216             dateTimeByteArray[6] = IntegerToBinaryCodedDecimal
                  (dateTime.Month);
217             dateTimeByteArray[7] = IntegerToBinaryCodedDecimal(dateTime.Year -
                     2000);
218
219             return dateTimeByteArray;
220         }
221
222         //Sets the real time clock at i2c address 0x68 to system time of
              Raspberry Pi3.
223         async void OnClickSetRtcTime(object sender, RoutedEventArgs e)
224         {
225             string aqs = I2cDevice.GetDeviceSelector();
226             var dis = await DeviceInformation.FindAllAsync(aqs).AsTask();
227             var settings = new I2cConnectionSettings(0x68);
228             settings.BusSpeed = I2cBusSpeed.FastMode;
229             settings.SharingMode = I2cSharingMode.Shared;
230
231             I2cDevice device = await I2cDevice.FromIdAsync(dis[0].Id,
                  settings);
232
233             device.Write(ConvertTimeToByteArray(DateTime.Now));
234         }
235
236         private static System.Timers.Timer aTimer;
237     }
238 }
239
```

...epos\MASS_Platinen_Test\MASS_Platinen_Test\I2cPage.xaml 1

```
<Page
    x:Class="MASS_Platinen_Test.I2cPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MASS_Platinen_Test"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="Transparent">

    <!-- Author: Daniel Heintze -->

    <Grid>
        <StackPanel HorizontalAlignment="Center" VerticalAlignment="Top"
          Margin="0,10,0,0">
            <TextBlock x:Name="Status" Text="Searching I²C..." Margin="10,50,10,10"
               TextAlignment="Center" FontSize="26" />
            <ProgressBar x:Name="ProgressValue" Minimum="0" Maximum="100"/>
            <ListView x:Name="listView1" Height="300"
          ScrollViewer.HorizontalScrollMode="Disabled"
          ScrollViewer.HorizontalScrollBarVisibility="Hidden"
          ScrollViewer.VerticalScrollMode="Enabled"
          ScrollViewer.VerticalScrollBarVisibility="Visible">
            </ListView>
        </StackPanel>

        <TextBlock x:Name="i2cClock" Text="Real Time Clock on 0x68: Please Wait..."
           VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Margin="10,10,10,10" TextAlignment="Left" FontSize="20" />
        <Button x:Name="setRtcTime" Click="OnClickSetRtcTime" ClickMode="Release"
          Content="Set RTC Time" FontSize="20" VerticalAlignment="Bottom"
          HorizontalAlignment="Right" Margin="10,10,10,10"
          Visibility="Collapsed"></Button>
    </Grid>
</Page>
```

...MASS_Platinen_Test\MASS_Platinen_Test\Rs232Page.xaml.cs 1

```
1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4  using System.Linq;
5  using System.Runtime.InteropServices.WindowsRuntime;
6  using System.Threading.Tasks;
7  using System.Timers;
8  using Windows.Devices.Enumeration;
9  using Windows.Devices.SerialCommunication;
10 using Windows.Foundation;
11 using Windows.Foundation.Collections;
12 using Windows.Storage.Streams;
13 using Windows.UI.Core;
14 using Windows.UI.Xaml;
15 using Windows.UI.Xaml.Controls;
16 using Windows.UI.Xaml.Controls.Primitives;
17 using Windows.UI.Xaml.Data;
18 using Windows.UI.Xaml.Input;
19 using Windows.UI.Xaml.Media;
20 using Windows.UI.Xaml.Navigation;
21
22 namespace MASS_Platinen_Test
23 {
24     /// <summary>
25     /// Rs232Page: This page contains the test for the RS232 interface.
26     /// It starts a loop which sends some data and shows if the data is
       received.
27     /// If no data can be send or received it will show an error message.
28     /// Author: Daniel Heintze
29     /// </summary>
30     public sealed partial class Rs232Page : Page
31     {
32         public Rs232Page()
33         {
34             this.InitializeComponent();
35         }
36
37         public async Task Serial()
38         {
39             try
40             {
41                 int i = 0;
42
43                 //Sets the visibility of the buttons, so the user only sees
                    the abort button, when the task is running.
44                 startbutton.Visibility = Visibility.Collapsed;
45                 abortbutton.Visibility = Visibility.Visible;
46
47                 string aqs = SerialDevice.GetDeviceSelector
                    ("UART0");                  /* Find the selector string for
                     the serial device   */
48                 var dis = await DeviceInformation.FindAllAsync(aqs).AsTask
                    ();          /* Find the serial device with our selector
                    string  */
49                 SerialPort = await SerialDevice.FromIdAsync(dis
                    [0].Id);                /* Create an serial device with our
```

...MASS_Platinen_Test\MASS_Platinen_Test\Rs232Page.xaml.cs 2

```
                    selected device */
                /* Configure serial settings */
                SerialPort.WriteTimeout = TimeSpan.FromMilliseconds(500);
                SerialPort.ReadTimeout = TimeSpan.FromMilliseconds(500);
                SerialPort.BaudRate =
                  9600;                                          /* mini
                  UART: only standard baudrates */
                //SerialPort.Parity =
                  SerialParity.None;                             /* mini
                  UART: no parities */
                //SerialPort.StopBits =
                  SerialStopBitCount.One;                      /* mini
                  UART: 1 stop bit */
                //SerialPort.DataBits = 8;

                Status.Text = "Test is running, please wait...";
                Status2.Text = "";

                while (i < 10)
                {
                    // Write a string out over serial
                    string txBuffer = "Hello Serial";
                    dataWriter = new DataWriter();
                    dataWriter.WriteString(txBuffer);
                    uint bytesWritten = await
                    SerialPort.OutputStream.WriteAsync(dataWriter.DetachBuffer
                    ());

                    // Read data in from the serial port
                    const uint maxReadLength = 1024;
                    dataReader = new DataReader(SerialPort.InputStream);
                    uint bytesToRead = await dataReader.LoadAsync
                    (maxReadLength);
                    string rxBuffer = dataReader.ReadString(bytesToRead);

                    Status2.Text = string.Format("{0} / Test No: {1}",
                    rxBuffer, i + 1);
                    i++;
                }
                Status.Text = "Test finished";
                Status2.Text = "Loop successful!";

                //Stops the timer and disposes the timer and the used serial
                  port.
                aTimer.Stop();
                aTimer.Dispose();
                SerialPort.Dispose();
            }
            catch (NullReferenceException ex)
            {
                Status2.Text = "Error: no reference";
                ErrorHandling.Text = string.Format("{0}", ex);
            }
            catch (Exception ex)
            {
                ErrorHandling.Text = string.Format("{0}", ex);
```

```
 94              }
 95              finally
 96              {
 97                  abortbutton.Visibility = Visibility.Collapsed;
 98                  startbutton.Visibility = Visibility.Visible;
 99              }
100          }
101
102          //When the start button is pressed, set and start the timer and then
               launch the serial initialization and loop.
103          async void OnClickStart(object sender, RoutedEventArgs e)
104          {
105              SetTimer();
106              await Serial();
107          }
108
109          //When the abort button is pressed, try to reset and dispose
               everything that is running.
110          void OnClickAbort(object sender, RoutedEventArgs e)
111          {
112              Status2.Text = "";
113              ErrorHandling.Text = "";
114
115              try
116              {
117                  dataWriter.Dispose();
118                  dataReader.Dispose();
119                  SerialPort.Dispose();
120                  aTimer.Stop();
121                  aTimer.Dispose();
122              }
123              catch (Exception ex)
124              {
125                  ErrorHandling.Text = string.Format("{0}", ex);
126              }
127
128              abortbutton.Visibility = Visibility.Collapsed;
129              startbutton.Visibility = Visibility.Visible;
130
131              Status.Text = "Abort successful";
132          }
133
134          //Sets and starts the timer with a 6 second interval.
135          private void SetTimer()
136          {
137              // Create a timer with a 6 second interval.
138              aTimer = new System.Timers.Timer(6000);
139              // Hook up the Elapsed event for the timer.
140              aTimer.Elapsed += OnTimedEvent;
141              aTimer.AutoReset = false;
142              aTimer.Enabled = true;
143          }
144
145          //When the 6 second were reached, try to dispose all running tasks and
               also stop and dispose the timer itself.
146          private void OnTimedEvent(Object source, ElapsedEventArgs e)
```

...MASS_Platinen_Test\MASS_Platinen_Test\Rs232Page.xaml.cs 4

```
147         {
148             var task = Dispatcher.RunAsync(CoreDispatcherPriority.Normal, ()
                  => {
149                 Status.Text = "Test aborted: maximum time reached";
150                 Status2.Text = "Dongle not found";
151                 try
152                 {
153                     dataWriter.Dispose();
154                     dataReader.Dispose();
155                     SerialPort.Dispose();
156                     aTimer.Stop();
157                     aTimer.Dispose();
158                 }
159                 catch (Exception ex)
160                 {
161                     ErrorHandling.Text = string.Format("{0}", ex);
162                 }
163                 finally
164                 {
165                     abortbutton.Visibility = Visibility.Collapsed;
166                     startbutton.Visibility = Visibility.Visible;
167                 }
168             });
169             aTimer.Stop();
170             aTimer.Dispose();
171         }
172 
173         //Activates the garbage collection, when the page is left.
174         protected override void OnNavigatedFrom(NavigationEventArgs e)
175         {
176             base.OnNavigatedFrom(e);
177             if (aTimer != null)
178             {
179                 try
180                 {
181                     dataWriter.Dispose();
182                     dataReader.Dispose();
183                     SerialPort.Dispose();
184                     aTimer.Stop();
185                     aTimer.Dispose();
186                 }
187                 catch (Exception ex)
188                 {
189                     ErrorHandling.Text = string.Format("{0}", ex);
190                 }
191             }
192             GC.Collect();
193         }
194 
195         protected override void OnNavigatedTo(NavigationEventArgs e)
196         {
197             base.OnNavigatedTo(e);
198         }
199 
200         private static System.Timers.Timer aTimer;
201         SerialDevice SerialPort;
```

...MASS_Platinen_Test\MASS_Platinen_Test\Rs232Page.xaml.cs 5

```
202             DataWriter dataWriter;
203             DataReader dataReader;
204         }
205     }
206
```

...os\MASS_Platinen_Test\MASS_Platinen_Test\Rs232Page.xaml 1

```
<Page
    x:Class="MASS_Platinen_Test.Rs232Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MASS_Platinen_Test"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="Transparent">

    <!-- Author: Daniel Heintze -->

    <Grid>
        <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
            <Button x:Name = "startbutton"
            Content = "Start RS232 test"
            Click = "OnClickStart"
            ClickMode = "Release"
            Margin = "10"
            Width = "150"
            HorizontalAlignment = "Center"
            Visibility="Visible"/>

            <Button x:Name = "abortbutton"
            Content = "Abort"
            Click = "OnClickAbort"
            ClickMode = "Release"
            Margin = "10"
            Width = "150"
            HorizontalAlignment = "Center"
            Visibility="Collapsed"/>

            <TextBlock x:Name="Status" Text="RS232 loop test" Margin="10,50,10,10" ↵
              TextAlignment="Center" FontSize="26" />
            <TextBlock x:Name="Status2" Text="" Margin="10,50,10,10" ↵
              TextAlignment="Center" FontSize="26" />
            <TextBlock x:Name="ErrorHandling" Text="" Margin="10,50,10,10" ↵
              TextAlignment="Center" FontSize="10" Visibility="Collapsed"/>
        </StackPanel>
    </Grid>
</Page>
```

...s\MASS_Platinen_Test\Mcp2515Can\Mcp2515CanController.cs 1

```
1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7  using Windows.Devices.Gpio;
8  using Windows.Devices.Spi;
9
10 namespace Mcp2515Can
11 {
12     enum OpCode : byte
13     {
14         Reset = 0xC0,
15         Read = 0x03,
16         ReadRxBuffer = 0x90,
17         Write = 0x02,
18     }
19
20     enum Register : byte
21     {
22         CanStat = 0x0E,
23     }
24
25     enum CanStateMode : byte
26     {
27         Normal = 0 << 5,
28         Sleep = 1 << 5,
29         Loopback = 2 << 5,
30         ListenOnly = 3 << 5,
31         Config = 4 << 5,
32         Mask = 0x7 << 5,
33     }
34
35     enum CanStatInterrupt : byte
36     {
37         None = 0 << 1,
38         Error = 1 << 1,
39         WakeUp = 2 << 1,
40         TXB0 = 3 << 1,
41         TXB1 = 4 << 1,
42         TXB2 = 5 << 1,
43         RXB0 = 6 << 1,
44         RXB1 = 7 << 1,
45         Mask = 7 << 1,
46     }
47
48     struct CanStat
49     {
50         byte mByte;
51         public CanStat(byte b)
52         {
53             mByte = b;
54         }
55
56         public CanStateMode Mode => (CanStateMode)((byte)CanStateMode.Mask &   ↵
```

...s\MASS_Platinen_Test\Mcp2515Can\Mcp2515CanController.cs 2

```
            mByte);
57          public CanStatInterrupt Interrupt => (CanStatInterrupt)((byte)
              CanStatInterrupt.Mask & mByte);
58      }
59
60
61      public class Mcp2515CanController
62      {
63          public static async Task<Mcp2515CanController> Create(SpiController
              spiCtrl, int spiCsPin, GpioController gpioCtrl, int interruptPin)
64          {
65              var ret = new Mcp2515CanController(spiCtrl, spiCsPin, gpioCtrl,
                  interruptPin);
66              await ret.Reset();
67              return ret;
68          }
69
70          public static async Task<Mcp2515CanController> Create(int spiCsPin,
              int interrruptPin)
71          {
72              var spiCtrl = await SpiController.GetDefaultAsync();
73              var gpioCtrl = await GpioController.GetDefaultAsync();
74              return await Create(spiCtrl, spiCsPin, gpioCtrl, interrruptPin);
75          }
76
77          public static Task<Mcp2515CanController> CreateRaspberryPi()
78          {
79              return Create(0, 4);
80          }
81
82          readonly SpiDevice mDev;
83          readonly GpioPin mInterrupt;
84          private Mcp2515CanController(SpiController spiCtrl, int spiCsPin,
              GpioController gpioCtrl, int interruptPin)
85          {
86              mDev = spiCtrl.GetDevice(new SpiConnectionSettings(spiCsPin)
87              {
88                  Mode = SpiMode.Mode0,
89                  DataBitLength = 8,
90                  SharingMode = SpiSharingMode.Exclusive,
91              });
92              mInterrupt = gpioCtrl.OpenPin(interruptPin,
                  GpioSharingMode.SharedReadOnly);
93          }
94
95          async Task Reset()
96          {
97              mDev.Write(new byte[] { (byte)OpCode.Reset });
98              CanStat stat = new CanStat();
99              for (int i = 0; i < 100; i++)
100             {
101                 stat = GetCanStat();
102                 if (stat.Mode == CanStateMode.Config)
103                     break;
104                 await Task.Delay(1);
105             }
```

...s\MASS_Platinen_Test\Mcp2515Can\Mcp2515CanController.cs 3

```
106             if (stat.Mode != CanStateMode.Config)
107                 throw new Exception("CAN controller did not transition to
                      config mode.");
108             //Debug.Write("asdf");
109             //Calling my dispose method.
110             Dispose();
111         }
112
113         CanStat GetCanStat()
114         {
115             return new CanStat(ReadRegister(Register.CanStat));
116         }
117
118         byte ReadRegister(Register reg)
119         {
120             var sendBuf = new byte[] { (byte)OpCode.Read, (byte)reg };
121             var recBuf = new byte[1];
122             mDev.TransferSequential(sendBuf, recBuf);
123             return recBuf[0];
124         }
125
126         //Needed new method to dispose after use, otherwise it wouldnt work a
              second time.
127         public void Dispose()
128         {
129             mDev.Dispose();
130             mInterrupt.Dispose();
131         }
132     }
133 }
134
```

...s\MASS_Platinen_Test\MASS_Platinen_Test\CanPage.xaml.cs 1

```
1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4  using System.Linq;
5  using System.Runtime.InteropServices.WindowsRuntime;
6  using Windows.Foundation;
7  using Windows.Foundation.Collections;
8  using Windows.UI.Xaml;
9  using Windows.UI.Xaml.Controls;
10 using Windows.UI.Xaml.Controls.Primitives;
11 using Windows.UI.Xaml.Data;
12 using Windows.UI.Xaml.Input;
13 using Windows.UI.Xaml.Media;
14 using Windows.UI.Xaml.Navigation;
15 using Mcp2515Can;
16
17 namespace MASS_Platinen_Test
18 {
19     /// <summary>
20     /// CanPage: This page contains the test for the Mcp2515 can controller.
21     /// It calls an external class "Mcp2515Can" which initializes the
         controller and if successful it will show a message that it was found.
22     /// If the controller cannot be initialized, it will throw an exception and
          show a message, that it was not found.
23     /// Author: Daniel Heintze
24     /// </summary>
25     public sealed partial class CanPage : Page
26     {
27         public CanPage()
28         {
29             this.InitializeComponent();
30         }
31
32         //Starts the initialization of the Mcp2515CanController and shows if it
             is successfully initialized or not.
33         private async void CanCheck()
34         {
35             try
36             {
37                 Status.Text = string.Format("Test running...");
38                 Status2.Text = string.Format("");
39                 startbutton.Visibility = Visibility.Collapsed;
40                 abortbutton.Visibility = Visibility.Visible;
41                 var canCtrl = await Mcp2515CanController.CreateRaspberryPi();
42                 Status2.Text = string.Format("CAN Controller successfully
                     initialized");
43             }
44             catch (NullReferenceException ex)
45             {
46                 Status2.Text = string.Format("Error: No reference/CAN not
                     found");
47                 ErrorHandling.Text = string.Format("{0}", ex);
48             }
49             catch (Exception ex)
50             {
51                 Status2.Text = string.Format("Error: CAN Controller not found
```

...s\MASS_Platinen_Test\MASS_Platinen_Test\CanPage.xaml.cs 2

```
                    ");
52                  ErrorHandling.Text = string.Format("{0}", ex);
53              }
54              finally
55              {
56                  Status.Text = string.Format("Test finished");
57                  abortbutton.Visibility = Visibility.Collapsed;
58                  startbutton.Visibility = Visibility.Visible;
59                  GC.Collect();
60              }
61          }
62
63          //If the button is pressed, first it collects all unused ressources and ↵
               then starts the test.
64          void OnClickStart(object sender, RoutedEventArgs e)
65          {
66              GC.Collect();
67              CanCheck();
68          }
69
70          //If this button is pressed, it resets everything and you can start the ↵
               test again.
71          void OnClickAbort(object sender, RoutedEventArgs e)
72          {
73              Status.Text = string.Format("Test aborted");
74              Status2.Text = string.Format("");
75              ErrorHandling.Text = string.Format("");
76              abortbutton.Visibility = Visibility.Collapsed;
77              startbutton.Visibility = Visibility.Visible;
78              GC.Collect();
79          }
80
81          //Activates the garbage collection, when the page is left.
82          protected override void OnNavigatedFrom(NavigationEventArgs e)
83          {
84              base.OnNavigatedFrom(e);
85              GC.Collect();
86          }
87
88          protected override void OnNavigatedTo(NavigationEventArgs e)
89          {
90              base.OnNavigatedTo(e);
91          }
92      }
93  }
94
```

...epos\MASS_Platinen_Test\MASS_Platinen_Test\CanPage.xaml 1

```
<Page
    x:Class="MASS_Platinen_Test.CanPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MASS_Platinen_Test"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="Transparent">

    <!-- Author: Daniel Heintze -->

    <Grid>
        <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
            <Button x:Name = "startbutton"
            Content = "Start"
            Click = "OnClickStart"
            ClickMode = "Release"
            Margin = "10"
            Width = "150"
            HorizontalAlignment = "Center"
            Visibility="Visible"/>

            <Button x:Name = "abortbutton"
            Content = "Abort"
            Click = "OnClickAbort"
            ClickMode = "Release"
            Margin = "10"
            Width = "150"
            HorizontalAlignment = "Center"
            Visibility="Collapsed"/>

            <TextBlock x:Name="Status" Text="CAN Controller Test"
              Margin="10,50,10,10" TextAlignment="Center" FontSize="26" />
            <TextBlock x:Name="Status2" Text="" Margin="10,50,10,10"
              TextAlignment="Center" FontSize="26" />
            <TextBlock x:Name="ErrorHandling" Text="Test" Margin="0,38,20,22"
              TextAlignment="Center" FontSize="10" Visibility="Collapsed" />
        </StackPanel>
    </Grid>
</Page>
```

...\MASS_Platinen_Test\MASS_Platinen_Test\MainPage.xaml.cs 1

```
1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4  using System.Linq;
5  using System.Runtime.InteropServices.WindowsRuntime;
6  using System.Threading;
7  using System.Timers;
8  using Windows.Foundation;
9  using Windows.Foundation.Collections;
10 using Windows.UI.Core;
11 using Windows.UI.Xaml;
12 using Windows.UI.Xaml.Controls;
13 using Windows.UI.Xaml.Controls.Primitives;
14 using Windows.UI.Xaml.Data;
15 using Windows.UI.Xaml.Input;
16 using Windows.UI.Xaml.Media;
17 using Windows.UI.Xaml.Media.Imaging;
18 using Windows.UI.Xaml.Navigation;
19
20 namespace MASS_Platinen_Test
21 {
22     /// <summary>
23     /// MainPage: This is the start page, it will show when the program is
        started.
24     /// It contains an image for visuals and all buttons to navigate to the
        corresponding pages, which contain the tests.
25     /// Also it starts the timer for the date and time field on the top left.
26     /// Author: Daniel Heintze
27     /// </summary>
28     public sealed partial class MainPage : Page
29     {
30         public MainPage()
31         {
32             this.InitializeComponent();
33             timeBlock.Text = string.Format("System Date & Time: {0}",
                localDate);
34             SetTimer();
35         }
36
37         //Navigation to specific page, when the corresponding button is
            pressed.
38         void OnClickGpio(object sender, RoutedEventArgs e)
39         {
40             gpiobutton.Content = ChangeImage(gpioPressed);
41             i2cbutton.Content = ChangeImage(i2cUnpressed);
42             rs232button.Content = ChangeImage(rs232Unpressed);
43             canbutton.Content = ChangeImage(canUnpressed);
44             subFrame.Navigate(typeof(GpioPage), null);
45         }
46
47         void OnClickI2c(object sender, RoutedEventArgs e)
48         {
49             i2cbutton.Content = ChangeImage(i2cPressed);
50             gpiobutton.Content = ChangeImage(gpioUnpressed);
51             rs232button.Content = ChangeImage(rs232Unpressed);
52             canbutton.Content = ChangeImage(canUnpressed);
```

...\MASS_Platinen_Test\MASS_Platinen_Test\MainPage.xaml.cs 2

```
53              subFrame.Navigate(typeof(I2cPage), null);
54          }
55
56          void OnClickRs232(object sender, RoutedEventArgs e)
57          {
58              rs232button.Content = ChangeImage(rs232Pressed);
59              gpiobutton.Content = ChangeImage(gpioUnpressed);
60              i2cbutton.Content = ChangeImage(i2cUnpressed);
61              canbutton.Content = ChangeImage(canUnpressed);
62              subFrame.Navigate(typeof(Rs232Page), null);
63          }
64
65          void OnClickCan(object sender, RoutedEventArgs e)
66          {
67              canbutton.Content = ChangeImage(canPressed);
68              gpiobutton.Content = ChangeImage(gpioUnpressed);
69              i2cbutton.Content = ChangeImage(i2cUnpressed);
70              rs232button.Content = ChangeImage(rs232Unpressed);
71              subFrame.Navigate(typeof(CanPage), null);
72          }
73
74          //Activates the Garbage Collector and exits the program.
75          void OnClickExit(object sender, RoutedEventArgs e)
76          {
77              exitbutton.Content = ChangeImage(exitPressed);
78              gpiobutton.Content = ChangeImage(gpioUnpressed);
79              i2cbutton.Content = ChangeImage(i2cUnpressed);
80              rs232button.Content = ChangeImage(rs232Pressed);
81              canbutton.Content = ChangeImage(canUnpressed);
82              aTimer.Stop();
83              aTimer.Dispose();
84              subFrame.Navigate(typeof(ExitPage), null);
85              GC.Collect();
86              MASS_Platinen_Test.App.Current.Exit();
87          }
88
89          //Loads a new image and updates the existing one.
90          private Image ChangeImage(String destination)
91          {
92              Image myImage = new Image();
93              BitmapImage biImage = new BitmapImage();
94              biImage.UriSource = new Uri(destination, UriKind.Absolute);
95              myImage.Stretch = Stretch.Fill;
96              myImage.Margin = new Thickness(-25, -25, -25, -25);
97              myImage.Source = biImage;
98              return myImage;
99          }
100
101         //Sets and starts the timer with a 1 second interval.
102         private void SetTimer()
103         {
104             // Create a timer with a 1 second interval.
105             aTimer = new System.Timers.Timer(1000);
106             // Hook up the Elapsed event for the timer.
107             aTimer.Elapsed += OnTimedEvent;
108             aTimer.AutoReset = true;
```

...\MASS_Platinen_Test\MASS_Platinen_Test\MainPage.xaml.cs 3

```
109             aTimer.Enabled = true;
110         }
111
112         //Every 1 second, update the date and time.
113         private void OnTimedEvent(Object source, ElapsedEventArgs e)
114         {
115             var task = Dispatcher.RunAsync(CoreDispatcherPriority.Normal, ()
                  => {
116                 localDate = DateTime.Now;
117                 timeBlock.Text = string.Format("System Date & Time: {0}",
                      localDate);
118             });
119         }
120
121         DateTime localDate = DateTime.Now;
122         private static System.Timers.Timer aTimer;
123
124         //All image paths are stored here and can be changed.
125         private static string gpioPressed = "ms-appx:///Buttons/
              gpioPressed.png";
126         private static string gpioUnpressed = "ms-appx:///Buttons/
              gpioUnpressed.png";
127         private static string i2cPressed = "ms-appx:///Buttons/
              i2cPressed.png";
128         private static string i2cUnpressed = "ms-appx:///Buttons/
              i2cUnpressed.png";
129         private static string rs232Pressed = "ms-appx:///Buttons/
              rs232Pressed.png";
130         private static string rs232Unpressed = "ms-appx:///Buttons/
              rs232Unpressed.png";
131         private static string canPressed = "ms-appx:///Buttons/
              canPressed.png";
132         private static string canUnpressed = "ms-appx:///Buttons/
              canUnpressed.png";
133         private static string exitPressed = "ms-appx:///Buttons/
              exitPressed.png";
134     }
135 }
136
```

...pos\MASS_Platinen_Test\MASS_Platinen_Test\MainPage.xaml 1

```
<Page
    x:Class="MASS_Platinen_Test.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MASS_Platinen_Test"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <!-- Author: Daniel Heintze -->

    <Grid>
        <Grid.Background>
            <ImageBrush ImageSource="backPlate.png" Stretch="Fill"/>
        </Grid.Background>
        <TextBlock x:Name="timeBlock"  HorizontalAlignment="Left"
          VerticalAlignment="Top" Text="System Time: " TextAlignment="Left"
          Margin="10,10,0,0">
        </TextBlock>
        <StackPanel HorizontalAlignment="Left" VerticalAlignment="Top"
          Margin="0,115,0,0">
            <Button x:Name = "gpiobutton"
            Click = "OnClickGpio"
            ClickMode = "Release"
            Margin = "5,50,5,20"
            Width = "70"
            Background="Transparent">
                <Image Source="Buttons/gpioUnpressed.png" Stretch="Fill"
                  Margin="-25,-25,-25,-25"></Image>
            </Button>

            <Button x:Name = "i2cbutton"
            Click = "OnClickI2c"
            ClickMode = "Release"
            Margin = "5,5,5,20"
            Width = "70"
            Background="Transparent">
                <Image Source="Buttons/i2cUnpressed.png" Stretch="Fill"
                  Margin="-25,-25,-25,-25"></Image>
            </Button>

            <Button x:Name = "rs232button"
            Click="OnClickRs232"
            ClickMode = "Release"
            Margin = "5,5,5,20"
            Width = "70"
            Background="Transparent">
                <Image Source="Buttons/rs232Unpressed.png" Stretch="Fill"
                  Margin="-25,-25,-25,-25"></Image>
            </Button>

            <Button x:Name = "canbutton"
            Click = "OnClickCan"
            ClickMode = "Release"
            Margin = "5,5,5,20"
```

...pos\MASS_Platinen_Test\MASS_Platinen_Test\MainPage.xaml 2

```
            Width = "70"
            Background="Transparent">
                <Image Source="Buttons/canUnpressed.png" Stretch="Fill"
                  Margin="-25,-25,-25,-25"></Image>
            </Button>

            <Button x:Name = "exitbutton"
            Click = "OnClickExit"
            ClickMode = "Release"
            Margin = "5,5,5,20"
            Width = "70"
            Background="Transparent">
                <Image Source="Buttons/exitUnpressed.png" Stretch="Fill"
                  Margin="-25,-25,-25,-25"></Image>
            </Button>
        </StackPanel>
        <Frame x:Name="subFrame" Margin="75,50,0,0" Width="715" Height="420">
            <Image x:Name="Platine" Source="gpio-platine.png"/>
        </Frame>
    </Grid>
</Page>
```

Literaturverzeichnis

[1] Krügel, Dennis: *„3 Beispiele für Raspberry Einplatinencomputer in der Industrie“*, URL: https://www.expertenderit.de/blog/3-beispiele-fuer-raspberry-einplatinencomputer-in-der-industrie (Aufgerufen am 19. August 2019)

[2] MASS GmbH: *„Intelligenz im richtigen Outfit – Industriecomputer nach MASS“*, URL: https://www.mass.de/de/produkte/raspberry-pi-systeme.html (Aufgerufen am 19. August 2019)

[3] Raspberry Pi Foundation: *„About Us“*, URL: https://www.raspberrypi.org/about/ (Aufgerufen am 19. August 2019)

[4] Raspberry Pi Foundation: „*Downloads“*, URL: https://www.raspberrypi.org/downloads/ (Aufgerufen am 19. August 2019)

[5] Raspberry Pi Foundation: *„Raspberry Pi 4“*, URL: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/ (Aufgerufen am 19. August 2019)

[6] Raspberry Pi Foundation: *„What you will need“*, URL: https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up/2 (Aufgerufen am 19. August 2019)

[7] Microsoft: *„Release Notes for Raspberry Pi 3B+“*, URL: https://docs.microsoft.com/de-de/windows/iot-core/release-notes/insider/rpi3bp (Aufgerufen am 20. August 2019)

[8] Raspberry Pi Foundation: *„Raspberry Pi 3 Model B“*, URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ (Aufgerufen am 20. August 2019)

[9] Göhler, David: *„Was bietet Windows 10 IoT Core für Raspberry-Pi & Co.?“*, URL: https://www.expertenderit.de/blog/was-bietet-windows-10-iot-core-f%C3%BCr-raspberry-pi-co (Aufgerufen am 20. August 2019)

[10] Microsoft: *„Windows 10 IoT Core Dashboard“*, URL: https://docs.microsoft.com/de-de/windows/iot-core/connect-your-device/iotdashboard (Aufgerufen am 20. August 2019)

[11] Microsoft: *„An overview of Windows 10 IoT“*, URL: https://docs.microsoft.com/de-de/windows/iot-core/windows-iot (Aufgerufen am 20. August 2019)

[12] Microsoft: *„Visual Studio Community“*, URL: https://visualstudio.microsoft.com/de/vs/community/ (Aufgerufen am 20. August 2019)

[13] Microsoft: *„Schnelles Auffinden von Fehlern“*, URL: https://visualstudio.microsoft.com/de/vs/features/debugging-and-diagnostics/ (Aufgerufen am 20. August 2019)

[14] Microsoft: *„Überblick über C#“*, URL: https://docs.microsoft.com/de-de/dotnet/csharp/tour-of-csharp/ (Aufgerufen am 21. August 2019)

[15] Microsoft: *„Übersicht über XAML“*, URL: https://docs.microsoft.com/de-de/dotnet/framework/wpf/advanced/xaml-overview-wpf (Aufgerufen am 21. August 2019)

[16] Rasppishop: *„Raspberry Pi 7“ Touchscreen Display“*, URL: https://www.rasppishop.de/Raspberry-Pi-7-Touchscreen-Display (Aufgerufen am 22. August 2019)

[17] Elektronik Kompendium: *„Raspberry Pi: GPIO – General Purpose Input Output“*, URL: https://www.elektronik-kompendium.de/sites/raspberry-pi/2002191.htm (Aufgerufen am 23. August 2019)

[18] Mikrocontroller.net: *„I²C“*, URL: https://www.mikrocontroller.net/articles/I%C2%B2C (Aufgerufen am 23. August 2019)

[19] Kompendium by Infotip Service GmbH: *„RS-232 – Die Serielle Schnittstelle“*, URL: https://kompendium.infotip.de/rs-232-die-serielle-schnittstelle.html (Aufgerufen am 23. August 2019)

[20] Mikrocontroller.net: *„Serial Peripheral Interface“*, URL: https://www.mikrocontroller.net/articles/Serial_Peripheral_Interface (Aufgerufen am 23. August 2019)

[21] ME-Meßsysteme GmbH: *„Grundlagen zum CAN Bus“*, URL: https://www.mikrocontroller.net/attachment/6819/canbus.pdf (Aufgerufen am 23. August 2019)

[22] Microsoft: *„Hardwarekompatibilitätsliste“*, URL: https://docs.microsoft.com/de-de/windows/iot-core/learn-about-hardware/hardwarecompatlist (Aufgerufen am 26. August 2019)

[23] Microsoft: *„Raspberry Pi 2 & 3 Pin Mappings - Serial UART“*, URL: https://docs.microsoft.com/en-us/windows/iot-core/learn-about-hardware/pinmappings/pinmappingsrpi (Aufgerufen am 7. August 2019)

[24] Velladurai, Ravishankar: *„How To Control An LED Using Raspberry Pi 3, Windows IoT Core, And Visual Studio UWP“*, URL: https://www.c-sharpcorner.com/article/led-on-and-off/ (Aufgerufen am 31. Juli 2019)

[25] Microsoft: *„GpioPin Class“*, URL: https://docs.microsoft.com/en-us/uwp/api/windows.devices.gpio.gpiopin (Aufgerufen am 31. Juli 2019)

[26] Tutorialspoint: *„XAML – CheckBox“*, URL: https://www.tutorialspoint.com/xaml/xaml_checkbox.htm (Aufgerufen am 7. August 2019)

[27] Tutorialspoint: *„XAML – ToggleButton“*, URL: https://www.tutorialspoint.com/xaml/xaml_togglebutton.htm (Aufgerufen am 7. August 2019)

[28] Microsoft: *„Image.Source Property“*, URL: https://docs.microsoft.com/de-de/dotnet/api/system.windows.controls.image.source?view=netframework-4.8 (Aufgerufen am 13. August 2019)

[29] Porrey, Daniel: *„Discover i2c Devices on the Raspberry Pi“*, URL: https://www.hackster.io/porrey/discover-i2c-devices-on-the-raspberry-pi-84bc8b (Aufgerufen am 14. August 2019)

[30] WPF Tutorial: *„The ProgressBar Control“*, URL: https://www.wpf-tutorial.com/misc-controls/the-progressbar-control/ (Aufgerufen am 15. August 2019)

[31] Microsoft: *„Listenansicht und Rasteransicht“*, URL: https://docs.microsoft.com/de-de/windows/uwp/design/controls-and-patterns/listview-and-gridview (Aufgerufen am 15. August 2019)

[32] Microsoft: *„Timer Class“*, URL: https://docs.microsoft.com/de-de/dotnet/api/system.timers.timer?view=netframework-4.8 (Aufgerufen am 15. August 2019)

[33] Lindsay, Jeremy: *„How to use the DS1307 Real-Time clock with C# and the Raspberry Pi 3“*, URL: https://jeremylindsayni.wordpress.com/2016/06/18/how-to-use-the-ds1307-real-time-clock-with-c-and-the-raspberry-pi-3/ (Aufgerufen am 15. August 2019)

[34] Microsoft: *„DateTune.Now Property“*, URL: https://docs.microsoft.com/de-de/dotnet/api/system.datetime.now?view=netframework-4.8 (Aufgerufen am 15. August 2019)

[35] Wise, Austin: *„Windows 10 IoT driver for the MCP2515 CAN controller (WIP)“*, URL: https://github.com/AustinWise/Mcp2515Can (Aufgerufen am 16. August 2019)

[36] Unbekannt: *„One xaml page for multiple pages?“*, URL: https://stackoverflow.com/questions/33487368/one-xaml-page-for-multiple-pages (Aufgerufen am 16. August 2019)