Zusammenfassung der Arbeit
Abstract of Thesis

Fachbereich: Department:	**Electrical Engineering and Computer Science**
Studiengang: University course:	**Information Technology**
Thema: Subject:	**Raspberry Pi Programming with Windows IoT using C# and Xamarin.Forms – Hardware connections with I^2C**

Zusammenfassung:
Abstract:

The aim of this thesis is to investigate the possibilities and limitations of Raspberry Pi programming with Windows IoT operating system and Xamarin.Forms – a cross-platform development toolkit in C#. The thesis was conducted in May 2018 at Fachhochschule Lübeck. This topic is of interest because the Raspberry Pi, Windows IoT and Xamarin.Forms are new technologies for software development. This study did several programming experiments. These programs were connected to I^2C devices to fulfil their tasks such as run time comparison to Python, converting Windows Forms applications to Xamarin.Forms applications, automation control program for temperature and balance, and remote control with Android smartphone. In addition, some C# external libraries like “Microcharts” for chart display and “MicroTimer” for high-performance timer were also investigated and used in the programs. The results show that Raspberry Pi with Windows IoT is capable of performing the automation control as expected, the “Microcharts” and “MicroTimer” can be easily implemented by Xamarin.Forms. Furthermore, the conversion of the old C# applications as well as the cross-platform programming in Xamarin.Forms is considerably convenient. Despite of its limitation of compatibility problems and lack of UI elements, the Xamarin.Forms with Raspberry Pi and Windows IoT has shown their potential and they may be a suitable platform for software development. This thesis may be useful in providing an alternative way of Raspberry Pi programming and cross-platform application development.

Verfasser: Author:	**Qiwen Gu**
Betreuender Professor/in: Attending Professor:	**Prof. Dr. Jörg Bayerlein**
WS / SS:	**SS <2018>**

Table of Contents

1 Introduction

In this paper, the motivation as well as the problem area will be addressed, and the introduction of Raspberry Pi, Xamarin.Forms and other technologies will be proposed. The details of setting Raspberry Pi and how it was used with other devices will be introduced. In addition, several experiments will be mentioned, including the hardware connection and functional realization by C# programming language. In the end, the result of the experiment and further improvements will be evaluated.

1.1 Motivation

Nowadays, Computers play various important roles in industry and society all over the world and they also contribute to the need of interconnected electronic devices and Internet of Things (IoT).With the development of hardware technology and protocols such as I^2C, computers and micro controllers start to transfer into smaller size and better performance. Raspberry pi, for instance, is a small-size computer, which is designed by the Raspberry Pi Foundation in Britain [1]. It is low-cost, low-power and compatible with many operating system. In that case, it is designed for educational use on computer science and practical experiments [1]. In this case, Raspberry Pi might be a common hardware platform for software development in the future. Therefore, further research on the Raspberry Pi is necessary.

1.2 Goal

Preciously, several experiments have been conducted to implement Raspberry pi into practical use like remote storage, automatic control for small program [2], sensor detection [3] and so on. In most cases the functionalities are realized by using Python or C++ as programming language, because they are easier for the developers to understand and maintain the code. Moreover. They are extensible with many convenient libraries, which makes the program more extensible.

However, there still exists blanks in the current research on the Raspberry pi programming. The problem area is addressed at the programming language, operating system and programming platform on the Raspberry Pi. Normally, python or C++ is used as programming language and programs are developed directly by the corresponding integrated development environment (IDE). Considering the fact that many engineering programs are also developed in C#, it is still unknown whether C# programs can also run on Raspberry Pi. In addition, a new programming platform needs to be investigated which supports C# programming and makes software development more convenient.

In this paper, several programs were developed in C# using Xamarin.Forms, which is a new programming platform introduced by Microsoft. The programs were executed on Windows 10 Universal Windows Platform (UWP) on the Raspberry Pi to perform specific functionalities in combination with I^2C devices and electrical circuit. This paper aims at finding out the possibility of developing applications using Xamarin.Forms on the Raspberry Pi installed with Windows 10 IoT operating system. Moreover, the limitations of Raspberry Pi and Xamarin.Forms will be evaluated after the experiment.

1.3 Organization

This paper is composed of 4 main sections. In Section 2, the basic knowledge of the required hardware and software is introduced. The Section 3 provides the primary design of the applications to be developed in the thesis including software specification and hardware setup. In Section 4, the implementation of the applications and several research about the topic is documented. In the end, in Section 5, the evaluation and conclusion about the Raspberry Pi, Windows IoT and Xamarin.Forms is provided, and suggestions for further research is mentioned.

2 Basic information and technologies

This section introduces the basic knowledge about the hardware and software used in the paper in order to provide a primary understanding of the technology.

2.1 Hardware and development environment

In this paper, a laptop was chosen as the programming tool. The laptop was equipped with Intel® Core™ i7-4710HQ CPU @ 2.5GHz, 16 GB DDR3L RAM @1600MHz and the operating system was Windows 10 Home version 1709.

2.2 Background knowledge

In this section, the technical parameters and the characteristics about the Raspberry Pi will be provided and Xamarin.Forms will be introduced. Both of them are relatively new since Raspberry Pi was released in February 2012 [4] while the Xamarin.Forms was released on May 28, 2014 [5]. In addition, the principle of I Ȼ device will be introduced. Moreover, Microsoft Visual Studio was used as the integrated development environment (IDE) for coding. This section enables the reader to have a primary understanding of the technologies used in this paper.

2.2.1 Raspberry Pi

Raspberry Pi is a computer which is in the size of a card [6]. It can be seen as a computer providing elementary functions. Raspberry Pi can be install with many open-source operating systems such as Raspbian [7], which supports all kinds of functions and is easily usable by people. So it becomes a widely used device for learning programming in education institutions and for personal purpose as well [6]. In this paper, the Raspberry Pi 3 Model B was used and it was installed with Windows 10 IoT operating system, version 10.0.16299.15.

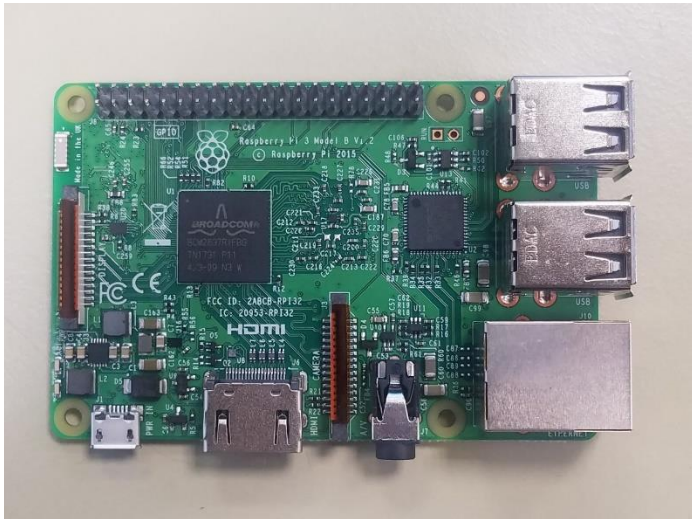


Figure 2-1: The structure of the Raspberry Pi 3 Model B

According to the Figure 2-1, the Raspberry Pi consists of I/O, CPU/GPU, RAM, USB hub, Ethernet plot, TF card slot and so on, which is a typical von Neumann architecture. The Raspberry Pi 3 Model B has 40 GPIO pins which are used for communication with other electronic devices via reading and writing the digital signals [8]. For the purpose of I ℃ communication, the GPIO #2 and GPIO#3 was used. The GPIO #2 is serial data-line (SDA), which reads and write 8 bit data with other, and the GPIO #3 is serial clock-line (SCL), which sends the clock signal for synchronization [8].

2.2.2 I ℃ devices

Nowadays, Inter-Integrated Circuit (I ℃) devices play an important role in the industry. In 1982, it was invented by Philips Semiconductor, which now becomes the NXP semiconductors [9] [10]. As is shown in Figure 2-2 [11], the I ℃ devices are connected in bus topology serially. There are one master device and multiple slave devices on the line to communicate with each other.

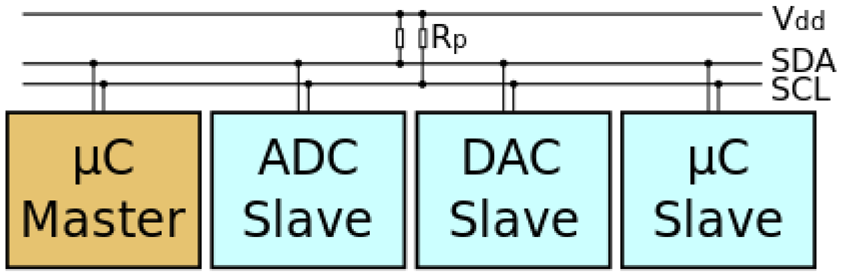


Figure 2-2: Inter-Integrated Circuit (I ℃) connection [11]

The I ℃ devices have 4 input pins. First, the VCC and GND pins are used as +5V or +3.3V power supply. Second, the serial clock-line (SCL) and serial data-line (SDA) are used for data transmission. Sensors, processors and microcontrollers are common peripherals which support the lows-peed, short distance and intra-board communication [12].

In order to make use of the I^{2}C devices, registration and configuration need to be done in advance. The I^{2}C devices contains 7-bit address space and 1-bit read/write signal. They are expressed in hexadecimal form and managed by registers [10]. The configuration of the I^{2}C devices includes bus speed, running mode and so on [13]. In addition, in Xamarin.Forms, the I^{2}C address is reduced into 7 bits, and the read/write actions are realized by specific methods.

2.2.3 Xamarin.Forms and C#

As is introduced by Microsoft (2015), C# supports the application development on .Net Framework. It is one of the object-oriented languages and is widely used in development for instance: Windows client applications, distributed applications, client-server applications and so on [14].

In addition, developers who have the experience of C, C++ or java programming will find it easy to get familiar with C# syntax. It is supported by many integrated development environments such as Visual Studio and develop platform for example Xamarin.Forms [14]. Considering these advantages, C# was selected as the programming language in this paper.

Generally speaking, there exist 3 main platforms for the mobile devices: Android, iOS and Universal Windows Platform (UWP). However, the cross-platform application demands many efforts because developers have to transfer one programming language to another. Therefore, Xamarin.Forms is a general and open-source platform which introduced one possible solution to overcome this problem. Petzold (2016) describes that Xamarin.Forms enables the developers to "use the same programming language, APIs and one single IDE to build cross-platform applications based on a shared C# codebase." [15]. Developers can create applications for iOS, Android, UWP and debug, update, maintain them at the same time.

As is shown in Figure 2-3, Xamarin.Forms makes use of the .NET Framework class library and the code is platform independent [15]. So when the project solution is created, it can be chosen as a Shared Asset Project (SAP), whose code and the program files can be directly used by other projects; or a Portable Class Library (PCL), which transfers the important and frequently used code to a Dynamic-Link Library (DLL), so other project can refer to the DLL and reuse the code [15].

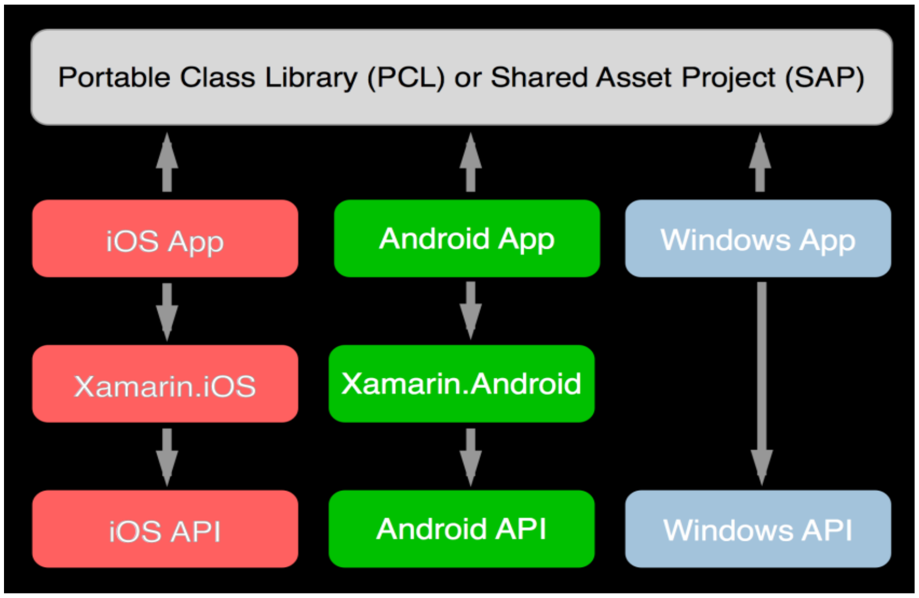


Figure 2-3: Xamarin C# libraries bind to native OS SDKs [15]

Considering the different way of programming between SAP and PCL, PCL is much more suitable for programming complex cross-platform applications [16]. Since the general functionalities are described in the interface and realized in independent platforms, it is more convenient for code debugging, maintenance and functional extension.

In 2017, the new version of the Xamarin.Forms supports .Net Standard project, which is a replacement of PCL [19]. In this paper, the Xamarin.Forms was used to program the C# code and the .Net Standard project was chosen as the standard project type.

2.2.4 Microsoft Visual Studio

The Visual Studio is an interactive development environment (IDE), it supports many programming languages, including JavaScript, C++, C# and so on [20]. It also allows the code editor and debugger, which enables developers to view and edit various kinds of code, and then debug, build, and publish applications for Android, iOS, Windows, the web, and the cloud [20]. Nowadays it is widely used for programming because it is usable for nearly all kinds software development for instance games, webs and mobile applications [20]. In addition, it can be extended with many plugins which supports other functions. Considering these advantages, Microsoft Visual Studio Professional 2017 (Version 15.6.6) was used in this paper. Because the Professional version supports more functionalities for instance emulator plugins, method reference indication, chart edition and so on. The Professional version is much more convenient for coding and debugging.

3 Design of the main applications

This section introduces the primary design of the applications and the specification of it. In addition, information about the general connection of Raspberry Pi with other devices is provided.

3.1 Functional requirements of the main applications

The functional requirements describes the main functionalities of the applications, which enable the Raspberry Pi to fulfill specific tasks.

3.1.1 Analog and digital convertor voltage measurement

The program called "ADC-DAC" should provide a page for initializing and configuring the 2 ADCs and 2 DACs. In addition, the user should be able to control the output voltage of both DACs and observe the input voltage of both ADCs.

3.1.2 Chart display

The application should provide a single chart page which shows curves of the input voltage of the ADCs in real-time. The real-time chart display should refresh itself every 1 second.

3.1.3 Temperature automation control

The application should provide a page which provides the functionality of temperature automation control. The user can set the desired temperature by the potentiometer on the "Hand Dryer" hardware and the Raspberry Pi controls the output voltage of DACs to meet the desired temperature. The input voltage of ADCs and the output control voltage of DACs should be shown on the screen every 1 second.

3.1.4 Android smartphone remote control

This is another application which is run both on Xamarin.Forms.Android and Xamarin.Forms.UWP at the same time. The main functionality is same as the temperature automation control. Additionally, the user can use the Android smartphone to control the desired temperature and the user can observe the current temperature, DAC output control voltage and temperature curves on the Android smartphone.

3.1.5 Balance automation control

The application should have a child page called "Balance control". The Raspberry Pi runs the program to automatically control the balance of a specific balancing device.

3.1.6 Page navigation

Except the Android smartphone remote control application, other programs should be integrated as a single application. The master page enables user to switch to the desired page conveniently. The detailed page includes analog and digital converter voltage measurement, chart display, temperature control and Balance control.

3.2 Non-functional requirements of the main applications

The non-functional requirements describes the properties and desired requirements of the applications.

3.2.1 Usability

The applications mainly focus their target users on electrical engineers. They have the experience of working with analog and digital converters, electric circuit, and I^{2}C devices. They are able to use electronic devices and tools to conduct experiments. After 30-minutes instruction, the user can be familiar with the applications and use them to conduct the experiments within 30-minutes instruction.

3.2.2 Reliability

The applications should use some methods to avoid initialization and configuration mistakes caused by careless users or errors which caused by external environment for instance poor network connection.

3.2.3 Compatibility

The applications should be able to run on the platforms which are supported by Xamarin.Forms, including Android and UWP.

3.2.4 Programming and human interface design

The applications code should have a clear structure and contains necessary description. It should also follow the coding convention of C#. The human interface design should provide necessary assistance and information to the users for the experiments.

3.2.5 Extensibility

The application files should be named clearly. The detailed description of the visual elements' names should be documented for the reader and developer. In addition, basic methods should be clearly implemented so that the extension and modification of the applications is convenient.

3.3 Device connection design for experiment

The Figure 3-1 shows the design of the required devices for the thesis. The applications in this paper are based on the I^2C devices, so the connection of the Raspberry Pi with these devices needs to be introduced in detail.

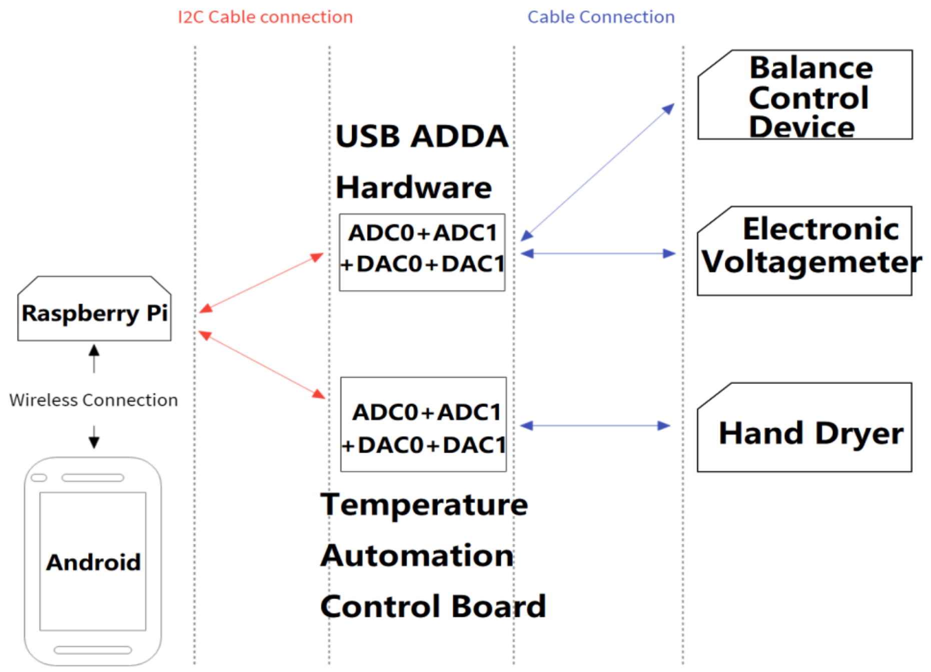


Figure 3-1: Device connection design

As can be seen, the Raspberry Pi was connected with 2 I^2C devices by I^2C cables. Each device had 2 analog to digital converters (ADC), 2 digital to analog converters (DAC) and other electric components. The I^2C devices were connected to different electrical devices by electrical cables to perform specific tasks. In addition, the Android smartphone can remotely control the Raspberry Pi.

3.4 Software structure

At the beginning of the implementation phase, the software structure was first designed by means of mockups, use case diagram and class diagram.

3.4.1 Human interface design

Before the application development, the human interface design was first conducted and the mockup of each page was primarily designed. However, since some programs were based on the original programs on Windows Forms [17] and other programs' human interface were deeply related to the functionalities, the description of the human interface interaction will be separated into each task parts in Section 4. Additionally, the mockups of the applications is provided in Appendix 10.1.

3.4.2 Use case diagram

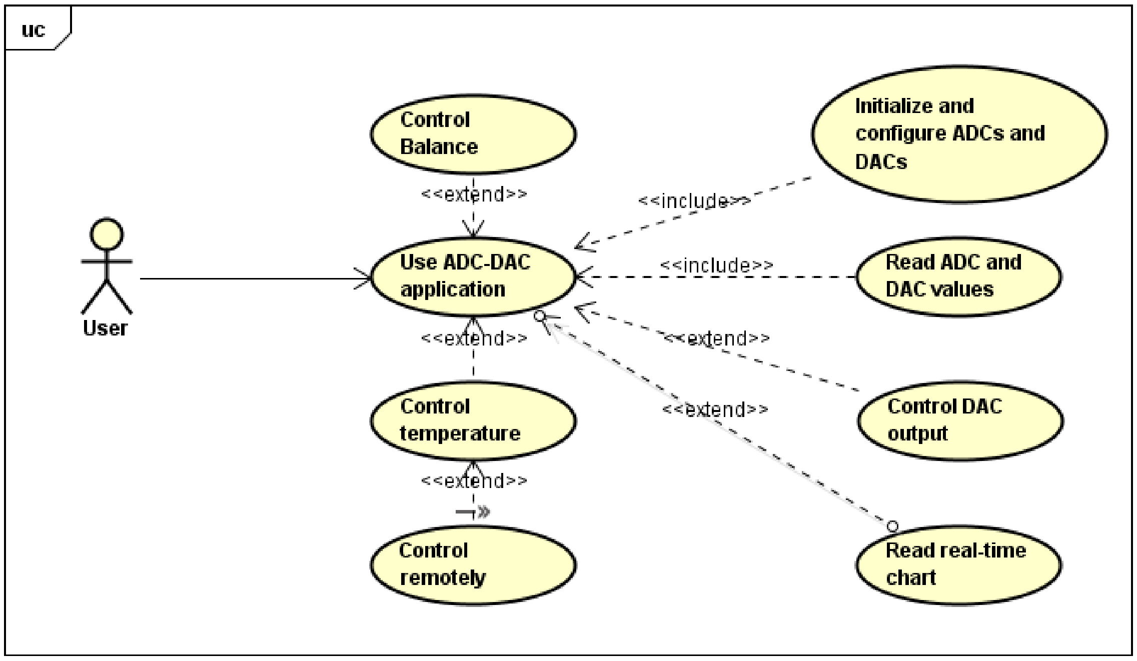


Figure 3-2: Use case diagram of the applications

As was mentioned earlier, there were 2 separate applications: “I2CADDA” was used for “ADC-DAC” program, chart display, temperature control and balance control, while “TempCon” was used for remote temperature control. According to the use case diagram, there are 8 use cases, which are explained in detailed below:

1. The user can enter the “ADC-DAC” page to use the analog to digital converters (ADC) and digital to analog (DAC) converters.
2. When the user is in “ADC-DAC” page and wants to use the ADCs and DACs, the initialization and configuration of these converters should be done at first.
3. The user can read the ADC and DAC values in the “ADC-DAC” page.
4. In the “ADC-DAC” Page, the user can control the DAC output value to output desired voltage on the device.
5. In “I2CADDA” application, the user can use Raspberry Pi to control the temperature on “Hand Dryer”
6. In “I2CADDA” application, the user can use Raspberry Pi to control the balance of the balancing device.
7. Alternatively, the user can use Android smartphone to control the temperature on “Hand Dryer” in “TempCon” application.

8. In both “I2CADDA” and “TempCon” application, the user can read real-time ADC inputs in “Micro Charts Display” page.

3.4.3 Class diagram

The class diagram describes the structure of the classes and libraries which are necessary for realizing the functionalities. The main structure of the applications kept similar to the original Windows Forms applications [17].

According to Figure 3-3, “I2CADDA” was composed of 2 platform projects: the PCL project and UWP project. In PCL project, the interface “IReadWriteI2C” was added in file “MainPage.xaml.cs”, it was used for managing I^{2}C devices as well as “MicroTimer”. Then in the UWP project, the class “ReadWriteI2C” implemented the interface in “MainPage.xaml.xs”, it also included “MicroLibrary” class to initialize and manage the “MicroTimer”. Specifically, the parameters in “ReadWriteI2C” were responsible for the control algorithm of temperature control and balance control.

In addition, the PCL project, the master page “HomePage” and 4 child page were created. As is shown in Figure 3-4, the “MainPage” was implemented as the application “ADC-DAC”, the “MicroChartsView” was used for chart display of the ADC inputs. Then “TempCon” and “BalCon” were responsible for temperature automation control and balance automation control. Moreover, the “MainPage”, “BalCon” and “TempCon” all called to the interface “IReadWriteI2C” to perform their tasks in “MicroTimer”.

The structure of “TempCon” application was very similar to the “I2ADDA” because the functionality of the temperature automation control was almost the same. According to Figure 3-5 and Figure 3-6, an Android platform project was created additionally and the interface “IDAL” was added into the PCL project for database communication. The UWP and Android both implement “IDAL” in their “DAL” file. What’s more, the “MainPage” and “MicroChartsView” both called to “IDAL” to communicate with the database and I^{2}C devices.

Finally after the design phase, the implementation of the applications could be started according to these specifications.

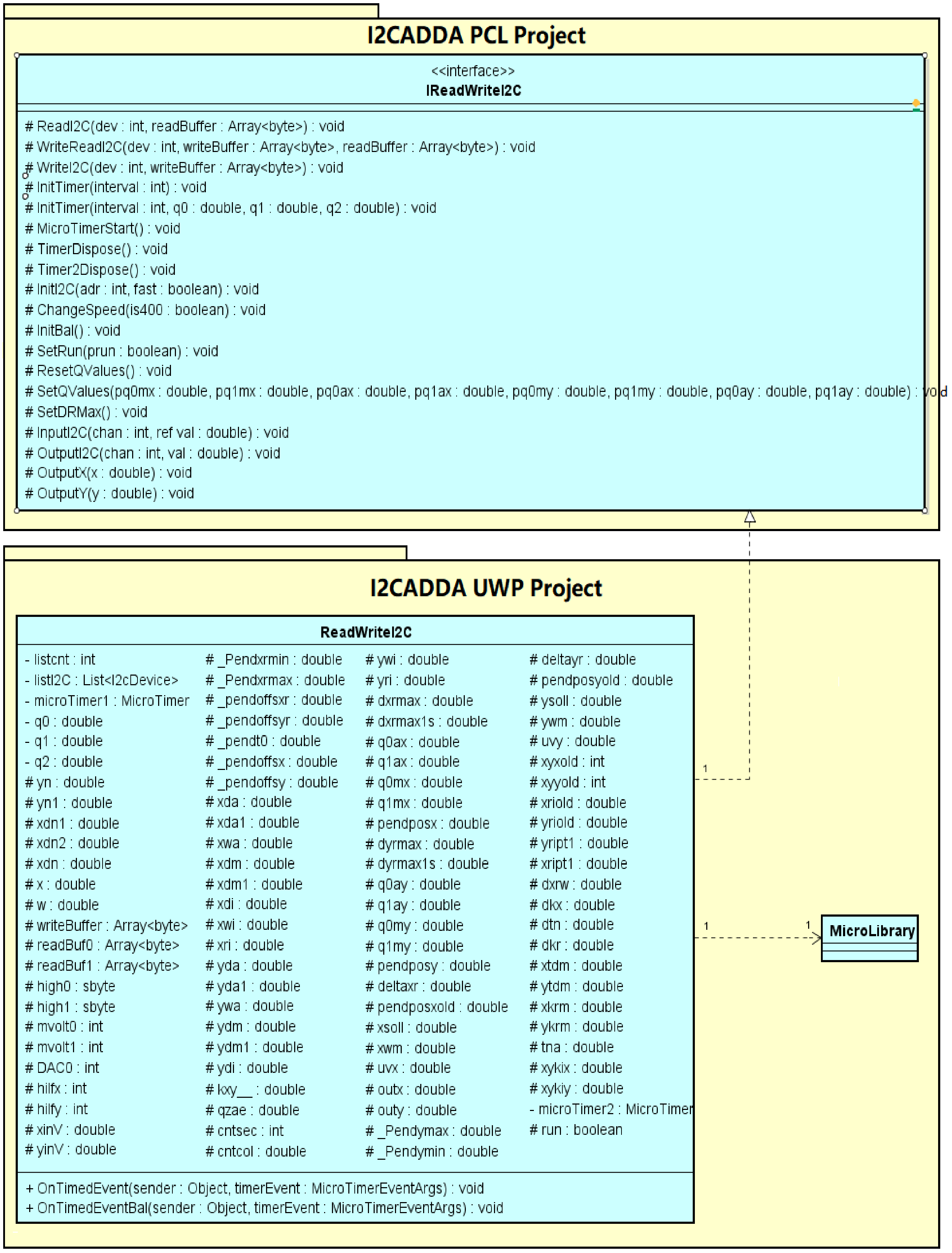


Figure 3-3: Class diagram of "I2CADDA" UWP project

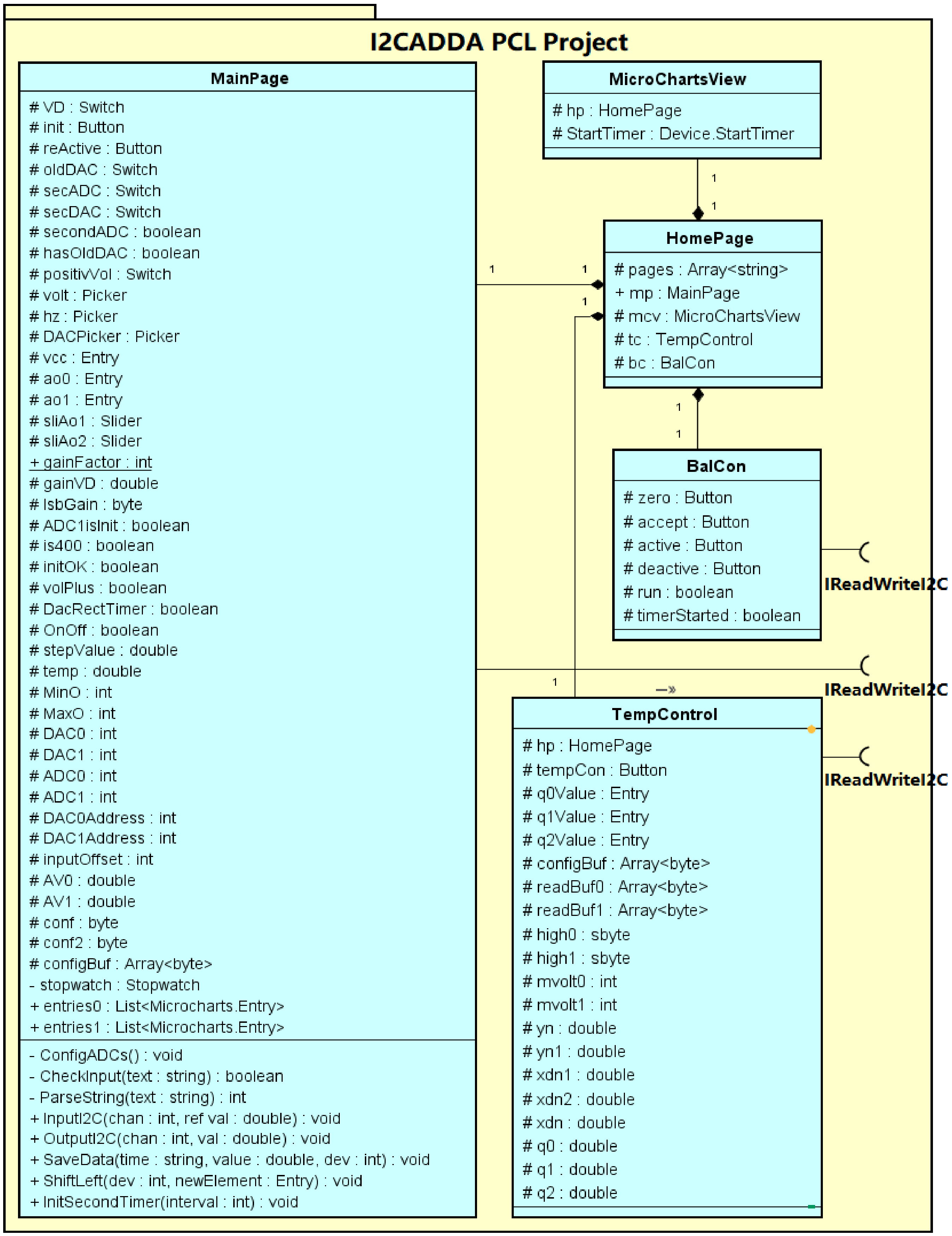


Figure 3-4: Class diagram of "I2CADDA" PCL project

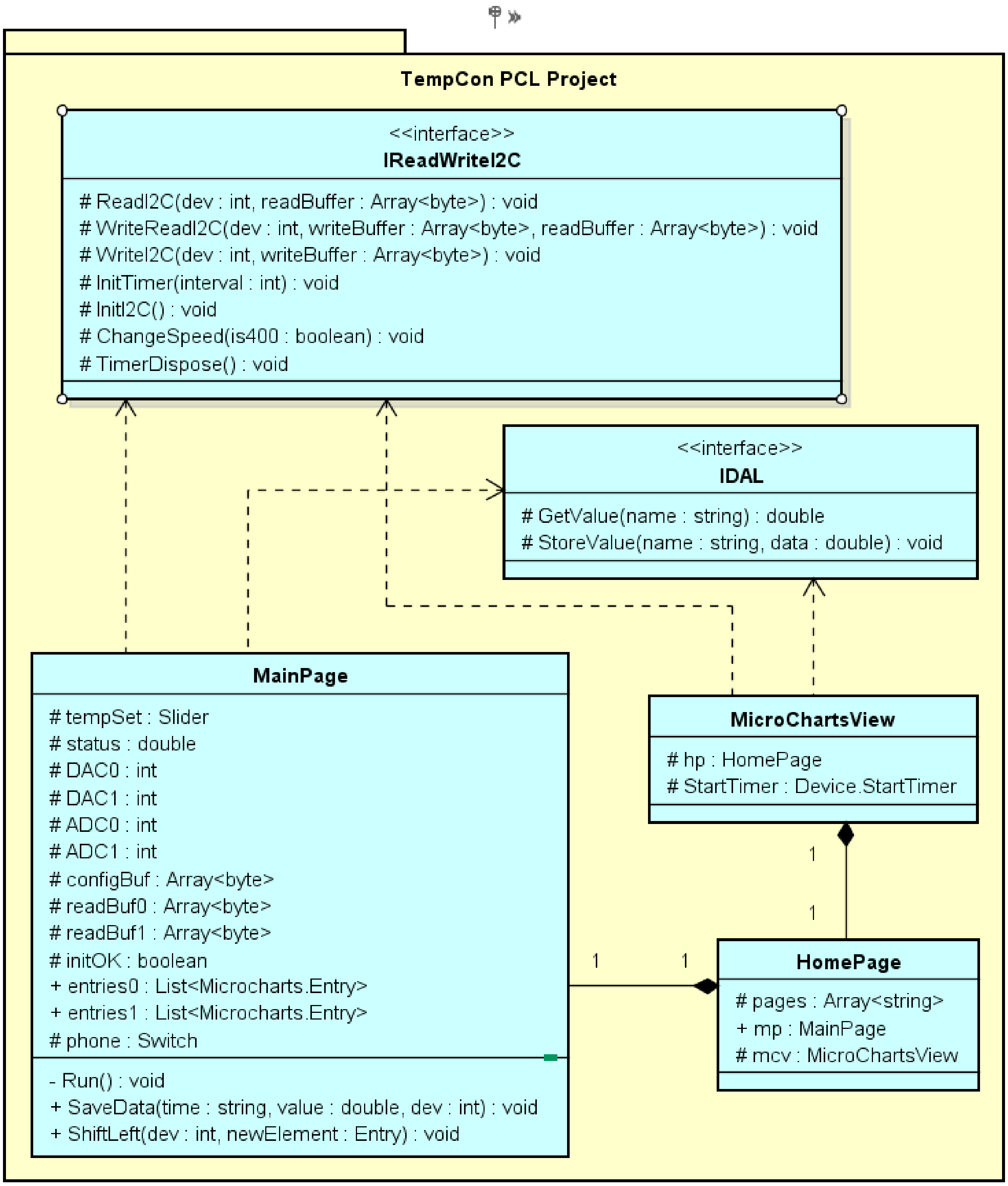

TempCon PCL Project
<<interface>>
IReadWriteI2C
# ReadI2C(dev : int, readBuffer : Array<byte>) : void
# WriteReadI2C(dev : int, writeBuffer : Array<byte>, readBuffer : Array<byte>) : void
# WriteI2C(dev : int, writeBuffer : Array<byte>) : void
# InitTimer(interval : int) : void
# InitI2C() : void
# ChangeSpeed(is400 : boolean) : void
# TimerDispose() : void
<<interface>>
IDAL
# GetValue(name : string) : double
# StoreValue(name : string, data : double) : void
MainPage
# tempSet : Slider
# status : double
# DAC0 : int
# DAC1 : int
# ADC0 : int
# ADC1 : int
# configBuf : Array<byte>
# readBuf0 : Array<byte>
# readBuf1 : Array<byte>
# initOK : boolean
+ entries0 : List<Microcharts.Entry>
+ entries1 : List<Microcharts.Entry>
# phone : Switch
- Run() : void
+ SaveData(time : string, value : double, dev : int) : void
+ ShiftLeft(dev : int, newElement : Entry) : void
MicroChartsView
# hp : HomePage
# StartTimer : Device.StartTimer
HomePage
# pages : Array<string>
+ mp : MainPage
# mcv : MicroChartsView
1
1
1
1

Figure 3-5: Class diagram of remote application "TempCon" PCL project

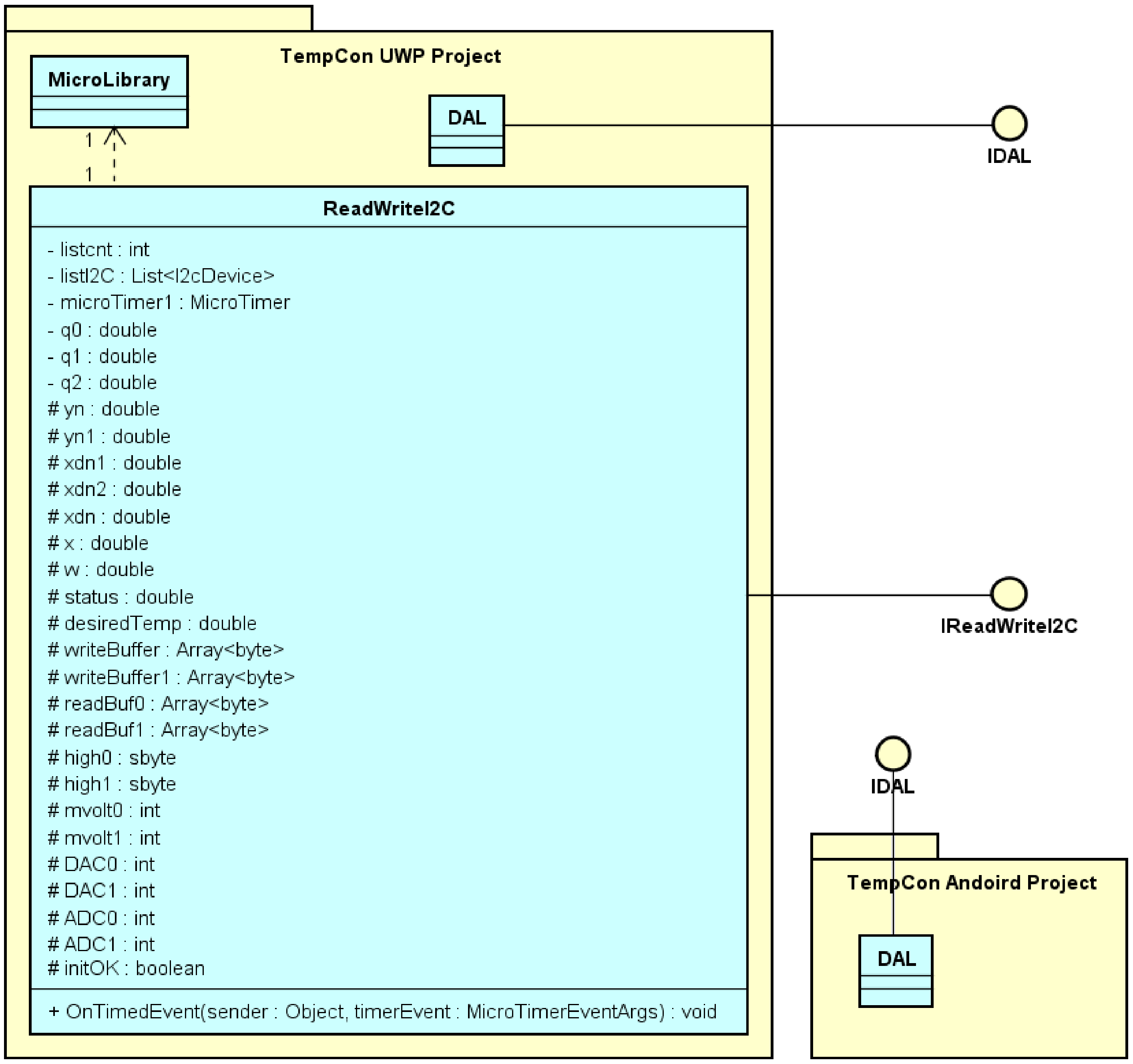


Figure 3-6: Class diagram of remote application "TempCon" UWP project

4 Implementation of the applications

In this section, the development phase of Xamarin.Forms C# applications will be introduced. The sections are divided into 8 parts according to the task list of the thesis. Additionally, some programs are transferred from the original Windows Forms application [17] and some functionalities were modified from an existing Xamarin.Forms example application from Prof. Dr. Bayerlein [29] [42].

4.1 Setting up Raspberry PI 3

The first step of utilizing the Raspberry Pi is setting up the hardware and installation of the necessary operating system. Then the Raspberry Pi can be connected to the computer for application development.

4.1.1 Cable connection with peripherals

The cable connection of the Raspberry Pi is shown in Figure 4-1. The Raspberry Pi provides many ports to connect to all kinds of devices and peripherals. Moreover, the Raspberry Pi has Ethernet ports for cable connection to the network or computer, and it supports Wi-Fi connection as well.

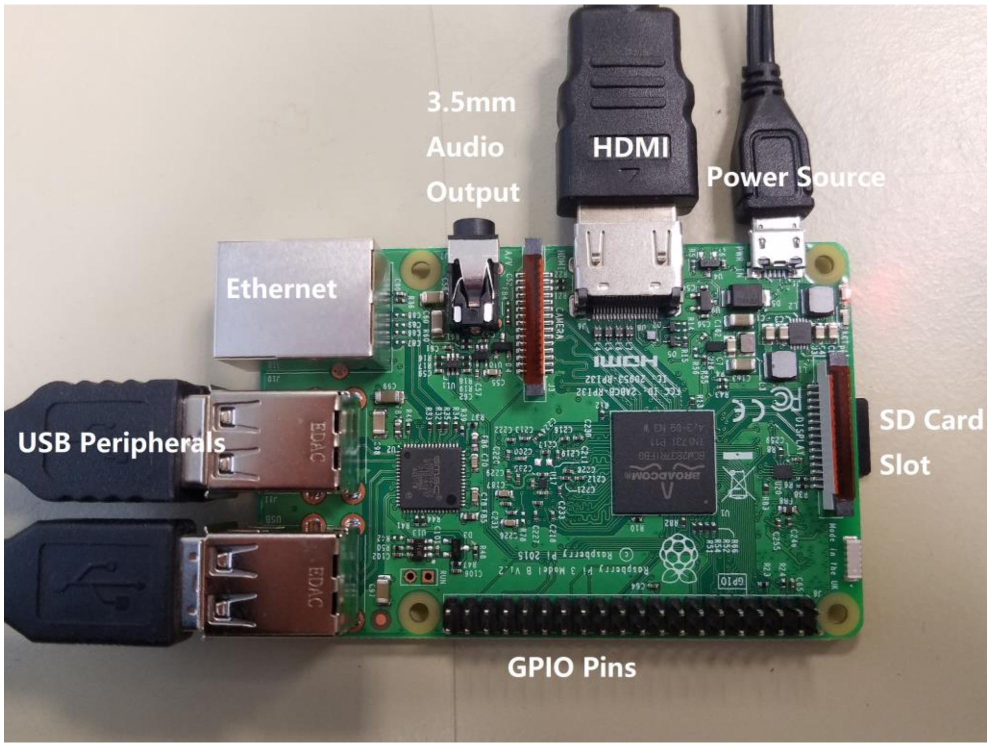


Figure 4-1: The cable connection of Raspberry Pi

4.1.2 Windows 10 IoT installation and software configuration

First, the computer with Windows 10 operating system should be installed with Windows IoT Core Dashboard. This is an administration and configuration software for Windows 10 IoT devices [21].

Second, the TF card should be plugged into the computer. The Windows IoT Core Dashboard would install the Windows 10 IoT operating system according to the instruction provided by Microsoft [21]. In addition, a WAVESHARE 10.1” HDMI LCD (H) touch screen in Figure 4-2 was connected to the Raspberry Pi by HDMI and USB port. Specifically, the resolution information should be written into the “config.txt” file which was located in the root of the TF card. The code is shown in Listing 4-1.

```
max_usb_currrent=1
hdmi_group=2
hdmi_mode=1
hdmi_mode=87
hdmi_cvt 1024 600 60 6 0 0 0
```

Listing 4-1: Screen resolution information

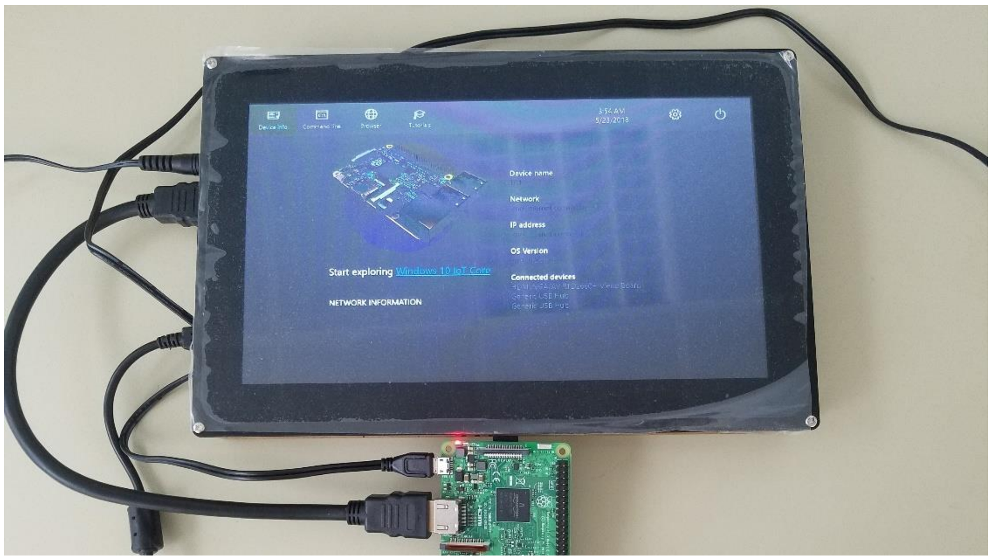


Figure 4-2: Touch screen connected with Raspberry Pi

Third, The Raspberry Pi should be connected by Ethernet or Wi-Fi to the network. In this paper, the Wi-Fi connection was used because wireless connection required fewer cables. The Raspberry Pi was connected to the laptop’s mobile hotspot for communication. The mobile hotspot is supported by Windows 10 and can be activated easily if the computer is equipped with wireless network card [22].

Finally, the administration and the setup of the Raspberry Pi could be done with the “Windows Device Portal” [21]. In Windows 10 IoT Core Dashboard, if the user goes to “My devices” page and double clicks the detected Raspberry Pi, a link called "open in device portal" will be shown at the bottom. "Windows Device Portal" is opened with a browser and needs login process. According to Figure 4-3, in the Windows Device Portal, device settings, application management and many other configurations can be done. User can also see the running process and real-time performance of the Raspberry Pi. In this thesis, the name of Raspberry Pi was "*TR1*", the user name was "*administrator*" and password was "*123*".

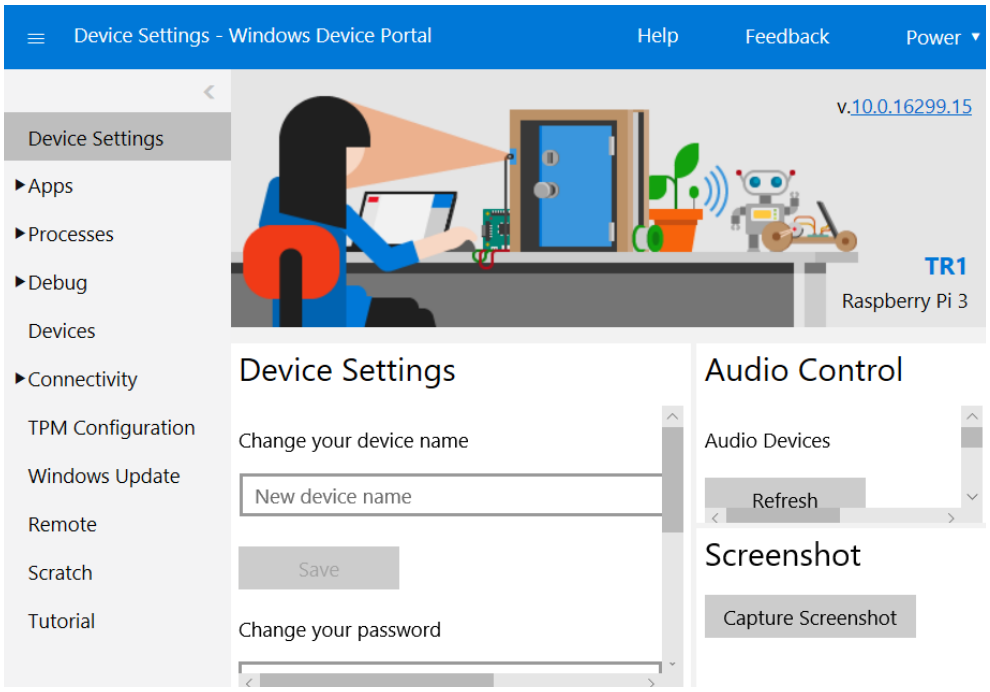
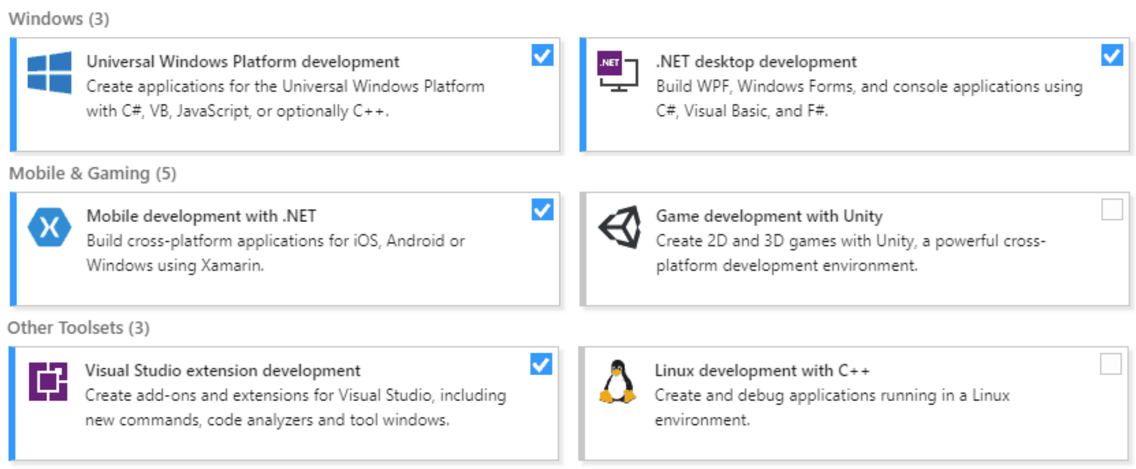


Figure 4-3: Window of "Windows Device Portal"

4.1.3 Visual Studio configuration

If the setup process of the Raspberry Pi is finished, the Visual Studio should be installed with corresponding development components. The necessary components is shown in Figure 4-4.



Windows (3)

Universal Windows Platform development ✓
Create applications for the Universal Windows Platform with C#, VB, JavaScript, or optionally C++.

.NET desktop development ✓
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F#.

Mobile & Gaming (5)

Mobile development with .NET ✓
Build cross-platform applications for iOS, Android or Windows using Xamarin.

Game development with Unity ☐
Create 2D and 3D games with Unity, a powerful cross-platform development environment.

Other Toolsets (3)

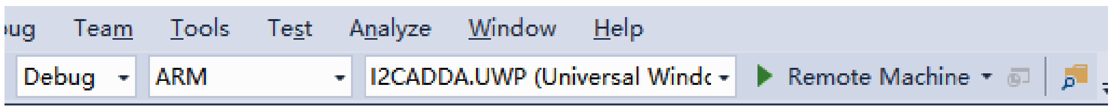
Visual Studio extension development ✓
Create add-ons and extensions for Visual Studio, including new commands, code analyzers and tool windows.

Linux development with C++ ☐
Create and debug applications running in a Linux environment.

Figure 4-4: Development component installed in Visual Studio

Then, the Visual Studio can be used for software development. The project should be created as "Mobile App (Xamarin.Forms)" in "Cross-Platform" section in "Visual C#" folder. Then the target platforms and ".Net Standard" (PCL) is selected to create the new project.

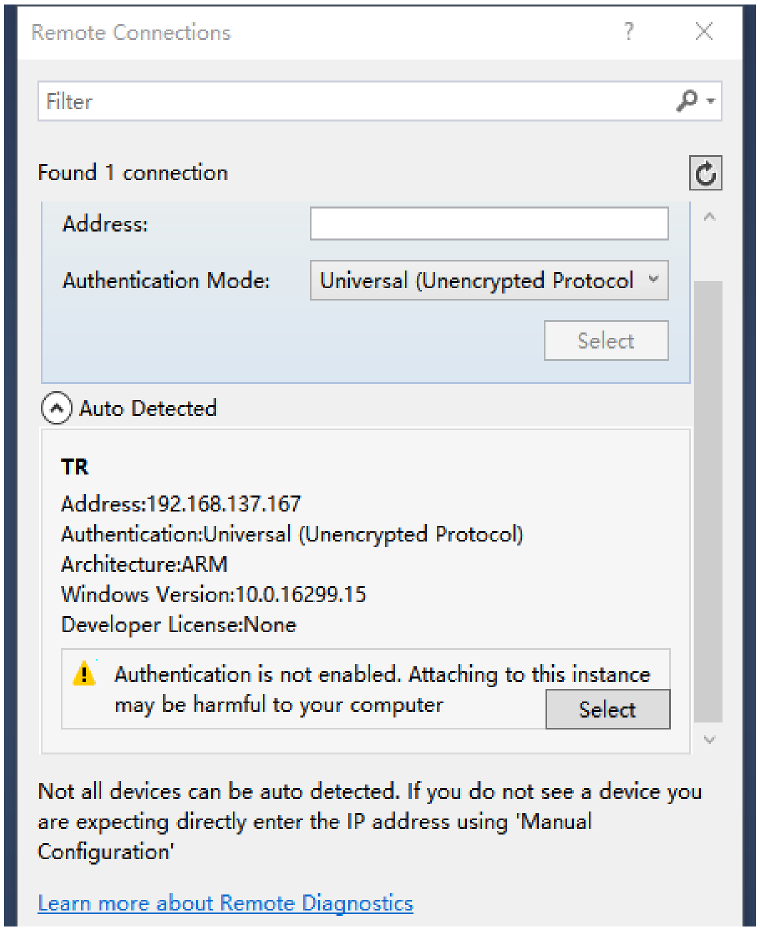
After programming in Visual Studio, the program can be run either on local machine (computer) or remote machine (Raspberry Pi). In this paper, since most applications run on UWP operating system, the UWP was set as the startup project first.



Debug ▾ ARM ▾ I2CADDA.UWP (Universal Windc ▾ ▶ Remote Machine ▾

Figure 4-5: Visual Studio debug with local machine

According to Figure 4-5, if the application is debugged on Raspberry Pi, the "ARM", and "Universal Windows Platform" and "Remote Machine" should be selected. As is shown in Figure 4-6, the auto-detected Raspberry Pi is be selected. If the Raspberry Pi is undetectable, the IP address should be typed in to set up a manual connection. In this paper, the Raspberry Pi as the remote machine was set as the default hardware platform for application debug.


Remote Connections

Filter

Found 1 connection

Address:

Authentication Mode: Universal (Unencrypted Protocol

Auto Detected

TR
Address:192.168.137.167
Authentication:Universal (Unencrypted Protocol)
Architecture:ARM
Windows Version:10.0.16299.15
Developer License:None

Authentication is not enabled. Attaching to this instance may be harmful to your computer

Not all devices can be auto detected. If you do not see a device you are expecting directly enter the IP address using 'Manual Configuration'

Learn more about Remote Diagnostics

Figure 4-6: Visual studio remote machine setup

4.1.4 Cable connection with I²C devices

On the Raspberry Pi, the GPIO #2 SDA pin and #3 SCL pin, together with the VCC and GND pins are connected to the I²C devices such as “USB ADDA” hardware. As can be seen in the Figure 4-7, the I²C pins must be connected correctly, otherwise, the hardware might be damaged.

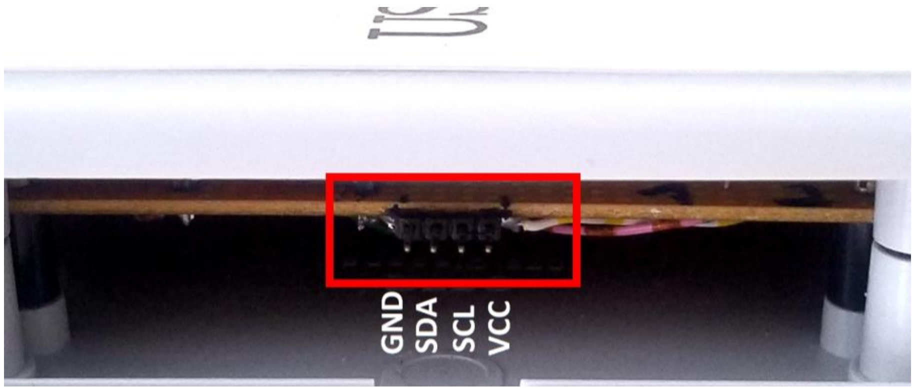


Figure 4-7: I²C pin connection with Raspberry Pi

4.2 Timing performance comparison

Generally, the Raspberry Pi applications are programmed in Python with Raspbian operating system. Raspbian is a common operating system for the Raspberry Pi. It is an open-source operating system and is optimized for the Raspberry Pi hardware [7]. And it has been installed with a Python IDE by default. For further application development, a timing test was conducted to find out the performance difference between Python and Win10- local machines. In that case, the developer can therefore find out which language is more suitable to use for development. The test result will be evaluated to determine performance of the Xamarin.Forms and Windows 10 IoT operating system.

4.2.1 Install Raspbian operating system on Raspberry Pi

In this task, the operating system Raspbian was installed on the Raspberry Pi in advance [23]. The Raspbian has been initially installed with Thonny. As was stated in the official website (2018), Thonny is “a Python IDE (Integrated Development Environment) designed for programming beginners” [24]. One major advantage of Thonny is that its human interface is user-friendly. It requires less than 10 minutes for a developer to get use of this IDE and program a simple application without instruction. In this section, the test program was simple and required few complex functions or external libraries. Considering these reasons, Thonny is a suitable IDE for this timing comparison test.

4.2.2 Timing performance application programming

In this task, 2 simple programs for the same mathematic calculation process were programmed in Python and C#, then they were run on both Raspbian and Windows 10 IoT operating system. The system timer was implemented in the applications to measure the time duration of these 2 programs to provide a comparison of the timing performance of the 2 programming languages.

First, a square calculation of a double number 1.002 was conducted for 100,000 times. In both C# and Python, a double number was initialized and a simple “for” loop was used. In the loop, the number was multiplied by itself.

Second, a string append operation was conducted for 100,000 times. In practice, Python treats the strings as array of chars [25], while C# with Xamarin.Forms support different kinds of strings. For example, “String” is a class of the Framework, “string” is an alias for “String” in the .NET Framework in C# [26], and “StringBuilder” is a specific class of “System.Text” of .NET Framework in C# for creating and modifying strings [27]. In this test, these classes were all tested and compared to the string in Python. The detailed string append functions in C# is provided in Listing 4-2.

```
String str = "";
string str2 = "";
StringBuilder sb = new StringBuilder(5);
for (int i = 0; i < 100000; i++)
    {
     str = str + "a";  //String(System) append
     str2 = str2 + "a";  //string(.Net) append
     sb.Append("a");  //StringBuilder append
    }
```

Listing 4-2: String append codes in C#

As was mentioned above, both programs used system timer to measure the general running time. In Python, the "time" library was imported and it provided the function "*time.time ()*" to get the current time. The run time duration could be calculated by minus the end time by start time. And the unit was second. In C#, the namespace "System.Diagnostics" [43] was used to initialize the "Stopwatch" as a timer. The "Stopwatch" provided "*Start ()*", "*Stop ()*", "*Rest ()*" methods to control the timer. At last, by means of "*ElapsedMilliseconds ()*", the duration of time could be achieved in millisecond.

4.2.3 Result of timing performance comparison

The purpose of this test was to investigate the timing difference of the Raspberry Pi with C# application and Python application. The run time of 2 applications were obtained and recorded in the following Table 4-3. It is apparent from this table that the double calculation in C# was much faster than that in Python. As can be seen, the double calculation took only 5 milliseconds in C#, while the Python took about 162 milliseconds to finish the calculation. What is interesting about the data in this table is that for both normal string append, including "string" in .Net Framework and "String" in "System", C# took more than 24,000 milliseconds, which was much slower than Python, however, by means of "StringBuilder" methods, the duration was only 4 about milliseconds. This class had improved the performance extremely when modifying the strings, as a result, it was almost 50 times faster than Python.

	C# + Windows 10 IoT	Python + Raspbian
Double calculation	5 ms	162 ms
String append(System)	24,366 ms	194 ms
string append(.NET)	25,079 ms	
StringBuilder append(System.Text)	4 ms	

Table 4-1: Run time comparison between C# and Python

The comparison of the 2 application run time revealed the obvious difference between the compilation, debugging and the treatment of data in Python and C# on the Raspberry Pi.

One possible explanation for this timing difference might be the running platform. Since the Python application was based on Thonny, it took considerably more time for compiling and registering itself in memory space and it had less priority while requesting the resources for execution than other processes in CPU. On the other hand, C# program had fewer steps for executing the program because it was run directly by the CPU as a single process, so its priority was considerably higher than Python application. This may explain why the double calculation in C# was typically faster than that in Python.

It is difficult to explain why the duration of string append in C# varied surprisingly, but it might be related to the different ways of string processing in Python and C#.

As indicated previously, Python was developed in C and it uses character arrays to represent strings. Therefore, when a string is defined in Python, it is allocated to a piece of memory space. If the string is modified, a new memory space is assigned to the string In addition, Python internally has the concept of a buffer pool [28]. In this test, the character "a" was appended after the target string for 100,000 times, so the character "a" was stored in the pool to avoid having a new space for "a" each time, therefore duration of normal string append was much quicker than C# string append.

One reason why the time of string process in C# varied surprisingly may be that C# supports different types of string. As was explained earlier, In C#, the "string" is an alias for "String" and it changes or improves some string operations. When the "string" is used the compiler will compile it to "String" automatically [26]. This may explain why "string" append was slightly slower than "String" append in C#. Moreover, modifying "String" takes time to create a new object and register it in a new address [26]. Thus, the string modification in C# was nearly 130 times slower in comparison with Python. Alternatively, the "StringBuilder" supports the string modification by expanding memory to accommodate the modified string instead of creating a new "String" object [30]. By means of "*append ()*" method, new string is appended to the original string. As a result, the duration time was reduced greatly, which consumed only 4 milliseconds.

This test compared the performance of both C# and Python on Raspberry Pi. By measuring the duration time of double calculation and string append, the result indicated that Xamarin.Forms with C# performed considerably better than Python. However, only simple function was tested in the experiment, and many other functions remained untested, for instance: calculation and processing of other data type, control structures and sentences, complex functions or instance process and thread, network transmission and so on. Further research should have a more complex and comprehensive test about these 2 programming languages in order to have a better understanding of the difference between C# and Python.

4.3 Timer component in Raspberry Pi

In many applications, a timer is needed to control the looping functionality in a fixed time duration. According to the definition provided by Microsoft (2018), a timer is a class which can “create and recur an event according to the set interval” [31]. In some cases, a high performance timer is essential for the control system for special functional requirements. Previous studies made use of different system timers to only fulfill the tasks but they paid little attention on the performance of the timer components. There is still uncertainty, however, whether there exists such kind of timer, which provides the high-performance timing functionality and is available for the Xamarin.Forms in C# on the Raspberry Pi. Therefore, this study first investigated the default system timer components and evaluated their performance, then a third-party open-source timer library for C# was investigated and tested by running it on Raspberry Pi to verify its performance.

4.3.1 Problem and timer components in C#

The .NET Framework Class Library offers 4 classes of timer. In most cases, the “System.Timers.Timer” and “System.Threading.Timer” are used in Xamarin.Forms [32].

Microsoft (2018) introduces that the “System.Timers.Timer” class is server-based and designed for a multithreaded environment. The “System.Timers.Timer” execute the “*OnTimedEvent ()*” method according to the set interval property [31].However, this class is unavailable for Universal Windows Platform. In that case, the “System.Threading.Timer” class is used instead. The use of “System.Threading.Timer” is similar to the “System.Timers.Timer”, a timer “*TimerCallback*” class as well as its event handler method should be initialized and implemented [32].

These 2 timers might be sufficient for normal application, but there exists limitation about the resolution of the timer. As noted by Ken (2013):“The limitation of the normal system timer is that Windows is unable to provide a high-performance timer which has the interval of less than 15 millisecond accurately. If the interval is set less than the resolution of the system clock, the “*TimerCallback*” method will execute according to the system clock interval, which is about 15 milliseconds in Windows” [33]. Therefore, a new timer should be used to solve the problem, and the solution may be the “MicroTimer”.

4.3.2 Principle of the “MicroTimer”

The “MicroTimer” is a third party library for timing functionality. It is developed in C# and provides the library file and can be easily used without knowing the exact code. It provides a timer which has the accuracy approximately 1 microsecond [33].

The calling of “MicroTimer” is similar to other system timers. The definition of the time interval is in microseconds and the “MicroTimer” execute the “*OnTimedEvent*” function to perform the desired functionality. However, the performance depends highly on the operating system as well.

If the operating system shifts the resources to other threads or processes, the “MicroTimer” may encounter timing delay [33].

In the next part of the section, the detailed implementation of “MicroTimer” in Xamarin.Forms will be described and the performance will be tested to find out whether “MicroTimer” performs as well as it is designed to be on Raspberry Pi.

4.3.3 Implementation of the “MicroTimer” with “DependencyService”

The use of the “MicroTimer” was simple. First, the ZIP file of the “MicroTimer” library should be downloaded and extracted as file [33]. Then in Visual Studio, the “MicroLibrary.cs” file was added as an existing file into the UWP project. And the setup process was already finished.

The main codes were placed at the PCL project, and the “MicroLibrary.cs” was added in UWP project because the “MicroTimer” was only available on UWP platform. Since they were in different platform, the access of the parameters and methods from PCL project to UWP project needed a special way, that was, the “DependencyService”. Microsoft (2017) describes that “the ‘DependencyService’ enables the application to call the platform-specific methods from shared code” [34]. In practice, the Xamarin.Forms application needs 5 steps to use “DependencyService”.

First, in order to implement the “MicroTimer”, an interface called “IReadWriteI2C” was created in the “MainPage.xaml.cs” in PCL project. In the interface, an initialization method called “*InitTimer (int interval)*” was added.
Second, in the UWP project, the implementation of the interface “IReadWriteI2C” was done in “ReadWriteI2C” class in the “MainPage.xaml.cs” file. There the method “*InitTimer (int interval)*” was implemented, a new “MicroTimer” object was created and the desired interval of the timer was set. Additionally, the timer was assigned a “MicroTimerElapsedEventHandler” called “*OnTimedEvent*” to fulfill the tasks and the timer could be started.
Third, the “*OnTimedEvent*” method was implemented in the same file to realize the functionality of the timer tick event.
Fourth, the “ReadWriteI2C” are registered with metadata code *“[assembly: Xamarin.Forms.Dependency(typeof (I2CADDA.UWP.ReadWriteI2C))]”* at the file header. So the general PCL project can find the correct implementation of the interface in each platform project [34].
Later, when the “*InitTimer (int interval)*” method was needed in PCL project, an instance of the “DependencyService” object was created. Therefore the methods call from the PCL project to the UWP project could be realized.
The whole steps of “DependencyService” is established by the Figure 4-8 [34], as can be seen, the implementation of the interface is done in separate platforms and the PCL project only needs to call the methods from the interface to fulfill the tasks. Even though the implemented code, imported libraries and used components may be different from each other, the functionality and behavior of

each platform project keeps the same. "DependencyService" separates the platform specific code into different project and makes the development and maintenance of the application much easier.

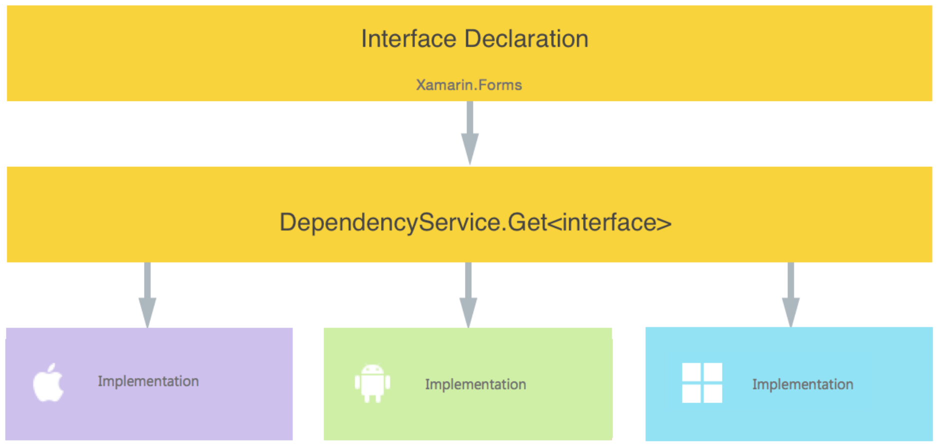


Figure 4-8: Steps of implementation of "DependencyService" [34]

In this test, a simple test of the "MicroTimer" was conducted on the Raspberry Pi. The code of the "DependencyService" and "MicroTimer" were referred and modified from the example from Prof. Dr. Bayerlein [29]. The interval of the "MicroTimer" was set to "1" which meant the period of the timer was 1 milliseconds. In the "*OnTimedEvent*", the "USB ADDA" device produced an output voltage of 0 volt or 1.25 volt each time (the use of the digital and analog converters in Xamarin.Forms will be also introduced in section 4.4).

4.3.4 Hardware connection with voltage meter

In this test, the "USB ADDA" was connected to the Raspberry Pi via I ₢ cables. The "USB ADDA" was composed of 2 analog to digital converters (ADC), 2 digital to analog converters (DAC) and other electric components, it provided many output and input ports for different range of voltage. According to the Figure 4-9, A and B are analog output ports from 0 volt to +5 volt, while C and D are analog output ports from -10 volt to +10 volt. Additionally, E and F are analog input ports from – 10 volt to +10 volt, G and H are analog input ports from – 20 volt to +20 volt.

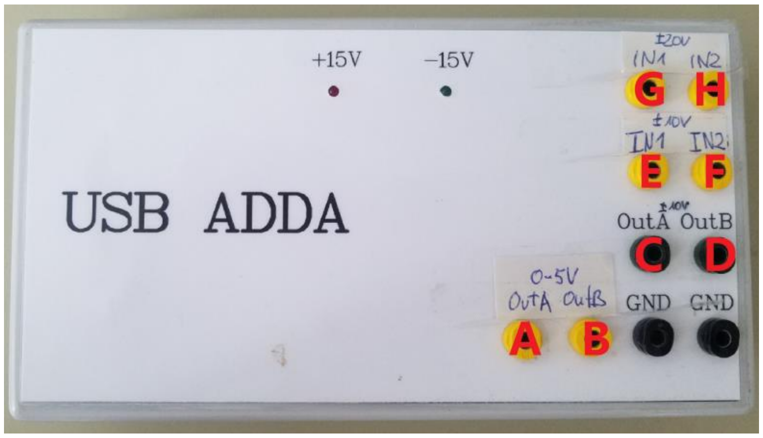


Figure 4-9: Ports of "USB ADDA" device

In order to observe the accurate output, the DAC 0 (port C) and GND ports were connected to "RIGOL DS1074B" digital oscilloscope to display the output curve on the screen. The curve would indicate whether the timer tick really happened in each 1 millisecond.

4.3.5 Result of "MicroTimer" output

The result of the test was show in the Figure 4-10. As can be seen, the DAC 0 generated the voltage curve from 0 volt to 1.25 volt. It is apparent in the figure that the scale of the curve was 500 microseconds and each impulse took 2 scales, which meant that the interval was exactly 1 milliseconds.

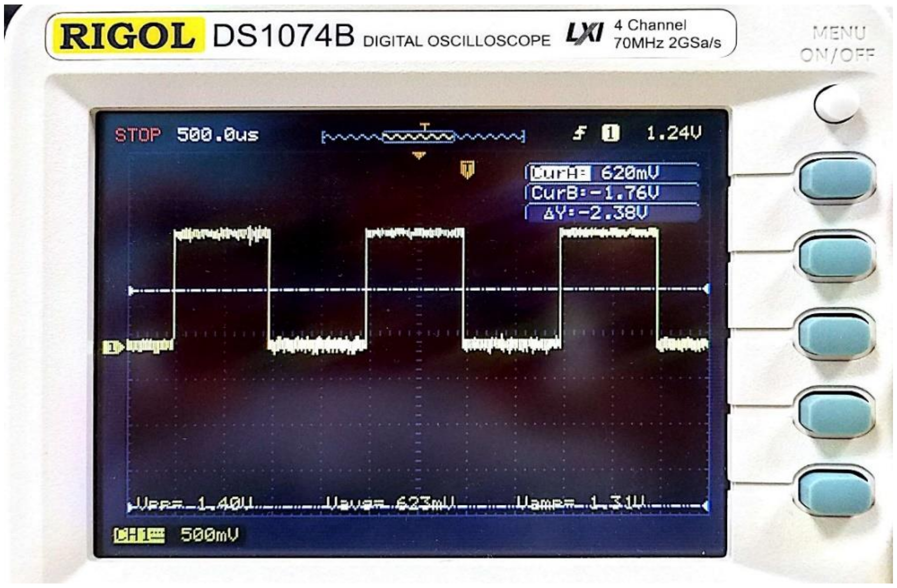


Figure 4-10: "MicroTimer" minimum interval test result

According to the result, the interval of the "MicroTimer" running on Windows 10 IoT on the Raspberry Pi was indeed 1 millisecond. This test has proved that the "MicroTimer" also supported the development of Xamarin.Forms application and on Raspberry Pi hardware. The performance of it remained the same as the Windows local applications.

As a result, the Raspberry Pi could make use of the “MicroTimer” library to develop UWP applications and realize high-performance timing functionalities. In this project, the “MicroTimer” was used in temperature control application and the balance control application, because both of them required an accurate timer.

4.4 Xamarin.Forms programming with 2 ADCs and 2 DACs hardware

In this section, a new Xamarin.Forms UWP application was programmed. The application was based on an original application which was programmed on Windows Forms in C# [17]. The main goal was to program an UWP application “ADC-DAC” which had the similar user interface and the same functionality. The new application should be run on the Raspberry Pi. This task is of interest because questions remains to be figured out whether Raspberry Pi is capable of interacting with human and other I^2C devices. In this task, the functionalities as well as the human interface were modified to improve the usability and reliability. In addition, the code compatibility of the application was investigated.

4.4.1 Hardware connection with Raspberry Pi

According to Figure 4-9, the Raspberry Pi was connected to the “USB ADDA” hardware. The analog output ports C and D were connected to the voltage meter “PM 2525 multimeter” to testify the exact output voltage. The analog input ports E and F were connected to the analog output nodes C and D directly, which means they shared a same voltage, so the input of the ADCs could be checked easily. The detailed connection of the Raspberry Pi is shown in Figure 4-11.

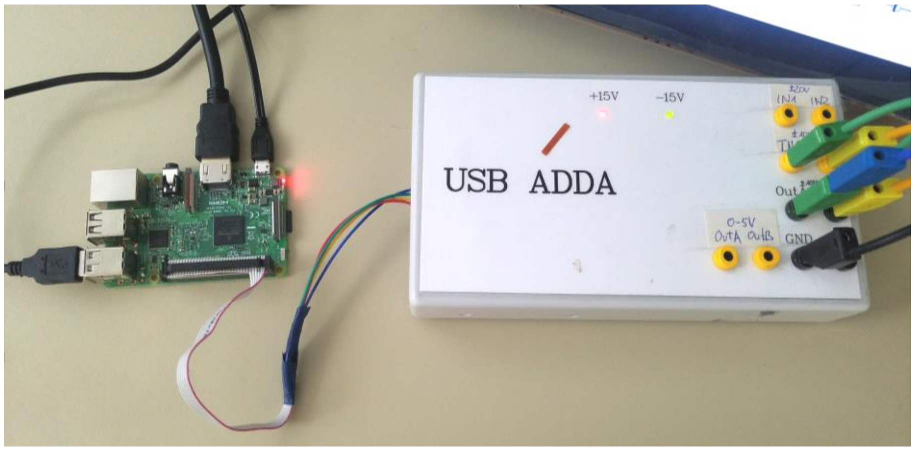
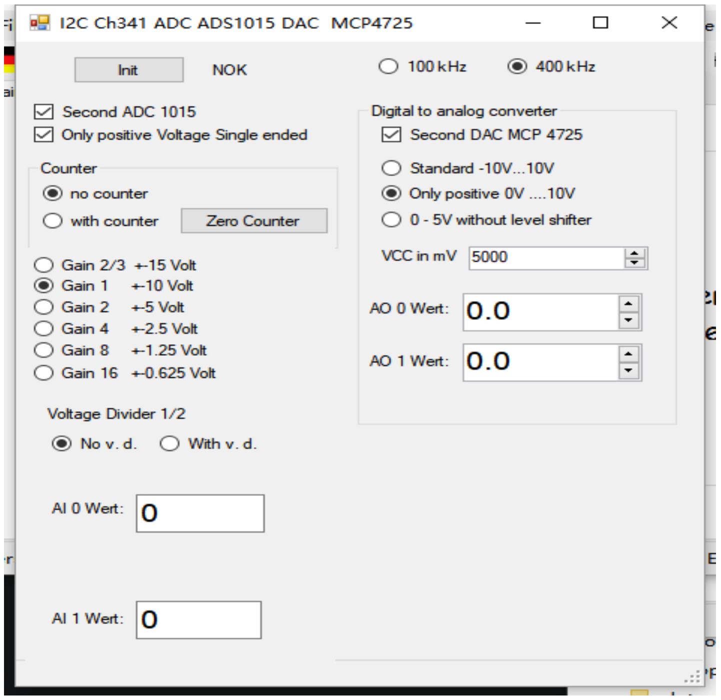


Figure 4-11: Raspberry Pi connection with "USB ADDA"

4.4.2 Original application introduction

As is shown in the Figure 4-12, the original application was based on Windows Forms platform, which contained several component: buttons, check boxes, radio buttons, text input fields and so on. It initialized, configured and controlled the output of the DACs and monitored the input of the DACs.


I2C Ch341 ADC ADS1015 DAC MCP4725
Init
Zero Counter
Counter
Digital to analog converter
Voltage Divider 1/2

NOK

100 kHz [✓ 400 kHz]

[✓ Second ADC 1015]
[✓ Only positive Voltage Single ended]

Counter: [✓ no counter] with counter

Gain 2/3 +-15 Volt [✓ Gain 1 +-10 Volt] Gain 2 +-5 Volt Gain 4 +-2.5 Volt Gain 8 +-1.25 Volt Gain 16 +-0.625 Volt

Voltage Divider 1/2: [✓ No v. d.] With v. d.

AI 0 Wert: 0

AI 1 Wert: 0

Digital to analog converter:
[✓ Second DAC MCP 4725]
Standard -10V...10V [✓ Only positive 0V ....10V] 0 - 5V without level shifter

VCC in mV: 5000

AO 0 Wert: 0.0

AO 1 Wert: 0.0

Figure 4-12: The original Windows Forms application

4.4.3 Human interface programming

In Xamarin.Forms, the content page had a child of grid view and was divided into 14 rows and 8 columns. Each element was added into the grid and could be automatically adapted to the changing resolution and window size.

In the content page, elements for example Button, Switch, Entry, Slider and Picker were used. In particularly, the Picker took place of the Radio Button in Windows Forms and the Switch took place of the check box, because the Xamarin.Forms supported limited types of visual elements. The number plus and minus buttons of the “AO” (analog output) were replaced by slider, so that it was easier to change the value. If the user change the number value in the “AO” (analog output), the value of the slider will also be changed and vice versa. Besides, the “Counter” and the “VCC” was used for reset and calibration of the application, which was irrelevant to the new application.

As is shown in Figure 4-13, the new layout of the UWP application “ADC-DAC” was very similar to the original Windows Forms application. The “Counter” was deleted, and some new switches and button were added according to the user’s requirements. What to be emphasized is that a switch called “Include 0xC4 ADC on old board” was added into the page, because the UWP application might control both old temperature automation control device and new “USB ADDA” device. However, the DAC addresses in these 2 devices were different. As a result, it would be inconvenient to use the devices. After adding a switch to choose the correct device, it was unnecessary for the user to change the addresses in the code and therefore it improved the usability and avoided errors.

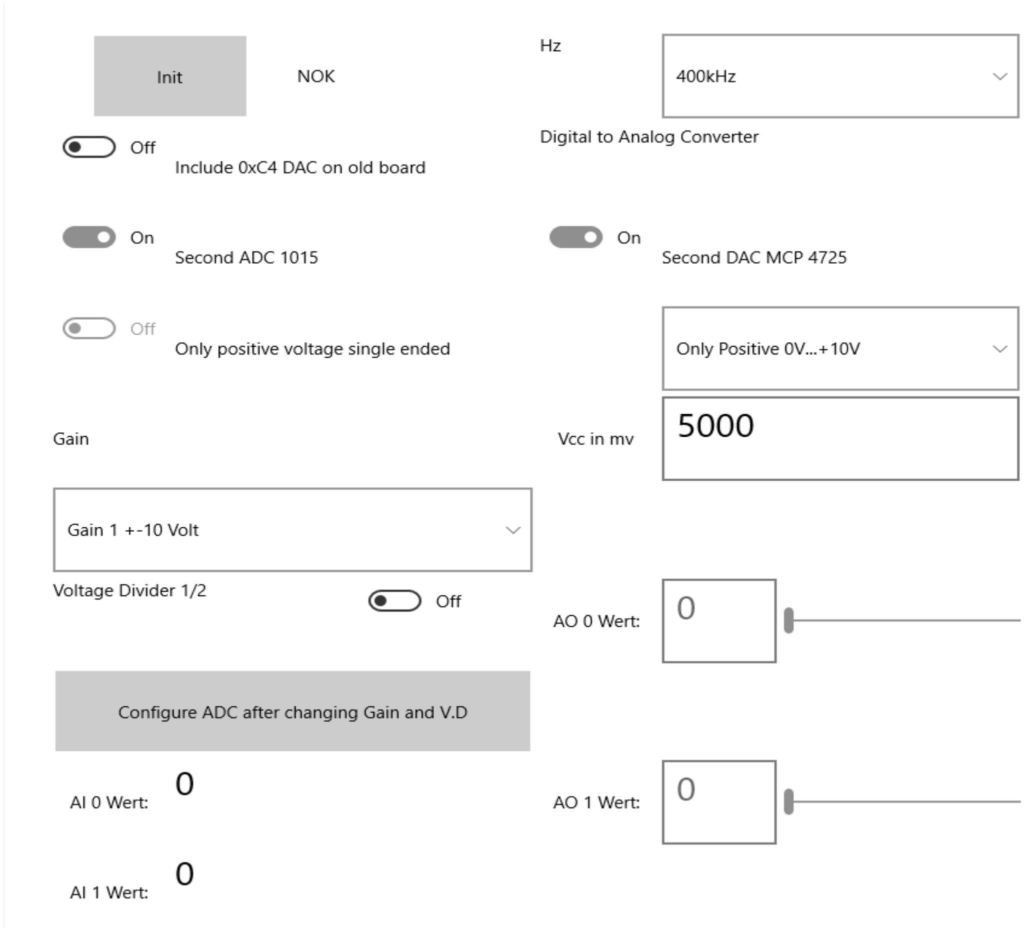
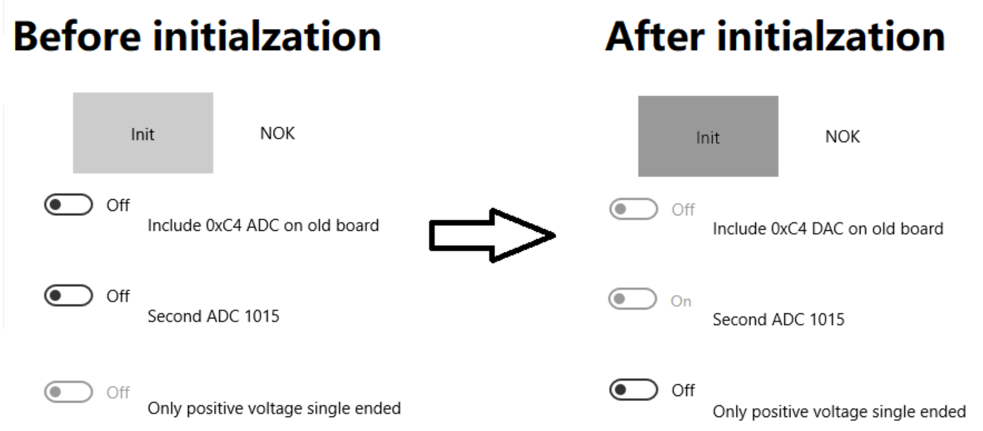

Init NOK
Hz: 400kHz
Off — Include 0xC4 DAC on old board
Digital to Analog Converter
On — Second ADC 1015
On — Second DAC MCP 4725
Off — Only positive voltage single ended
Only Positive 0V...+10V
Gain: Gain 1 +-10 Volt
Vcc in mv: 5000
Voltage Divider 1/2 — Off
AO 0 Wert: 0
Configure ADC after changing Gain and V.D
AI 0 Wert: 0
AO 1 Wert: 0
AI 1 Wert: 0

Figure 4-13:The new UWP application of ADC DAC

In the original application, the “Init” button must be clicked again to reconfigure the ADCs. However, it might cause severe errors and crash the application, and it reduced usability of the application. In order to solve this problem, the reconfiguration method was assigned to each element, so that after the setting was changed, ADCs would be automatically reconfigured again. Besides, the “Configure ADC after changing gain ad V.D” button was added into the content page to change the gain factor of the ADCs specifically.

In order to avoid conflicts and configuration errors caused by careless users, according to the Figure 4-14, if the “Init” button was clicked, the switches such as "Include 0xC4 ADC on old board" and

“Second ADC 1015” as well as the “Init” button would be disabled. Moreover, the “Only positive voltage single ended” switch would become enabled because it was used for ADC configuration especially for the temperature control device. With the help of these action, the robustness of the application was improved and the operation logic was clear and understandable.



Before initialzation:
Off Include 0xC4 ADC on old board
Off Second ADC 1015
Off Only positive voltage single ended

After initialzation:
Off Include 0xC4 DAC on old board
On Second ADC 1015
Off Only positive voltage single ended

Figure 4-14: Disable and enable the switches

Generally speaking, even though the layout of the application changed according to the requirements and target platform, the basic operating step and the functionalities remained similar to the previous one. So the users should find it easy to get familiar with the new UWP application.

4.4.4 Functional programming

In this section, the realization of the functionalities of the “ADC-DAC” application will be described. The functionalities of the application could be divided into 3 main categories: human interface interaction, I Ꞓ device management and ADC/DAC communication.

First, the human interface interaction part means the event handler of the visual elements. For instance, the event handler of the “sliAo0”, “sliAo1” (Sliders of DAC 0/ DAC 1) was realized to get the value of the slider and show them in “Ao1” and “Ao1” (Entries of DAC 0/ DAC 1). These kind of methods were responsible for human interface interaction, setting or getting the parameters for the program. And they played a minor role of the functional action. A table of these methods are provided in Appendix 10.2.1.

Second, the I Ꞓ device management methods were responsible for the I Ꞓ devices’ initialization, configuration and management. In this section, the I Ꞓ devices were the 2 analog to digital converter ADC 1, ADC 0 and 2 digital to analog converter DAC 0, DAC 1. In this part, the “DependencyService” was used again because only the UWP application supported the I Ꞓ device communication. For example, the method “*void InitI2C (int adr, bool fast)*” was added into the interface “IReadWriteI2C” and it was implemented in the UWP project. It got the address and bus speed of the I Ꞓ devices, initialized them and added them into the I Ꞓ device list for further operation. The responsibilities of these method are described in Appendix 10.2.2. What needs to be mentioned is that the methods of I^2C initialization and configuration were referred and modified from the example provided by Prof. Dr. Bayerlein [42].

Among these methods, attention needs to be paid on the I^2C initialization and configuration methods. In fact, an error called “Index out of range exception” happens most likely during the initialization process of the ADCs and DACs. There may be 2 feasible approaches to this problem, one solution is initialize and configure the ADCs first and at last initialize the DACs. Another solution was suspend the configuration for one second using code “*System.Threading.Thread.Sleep (1000)*”. Finally the initialization threads could have enough time to finish their work and the configuration could start successfully.

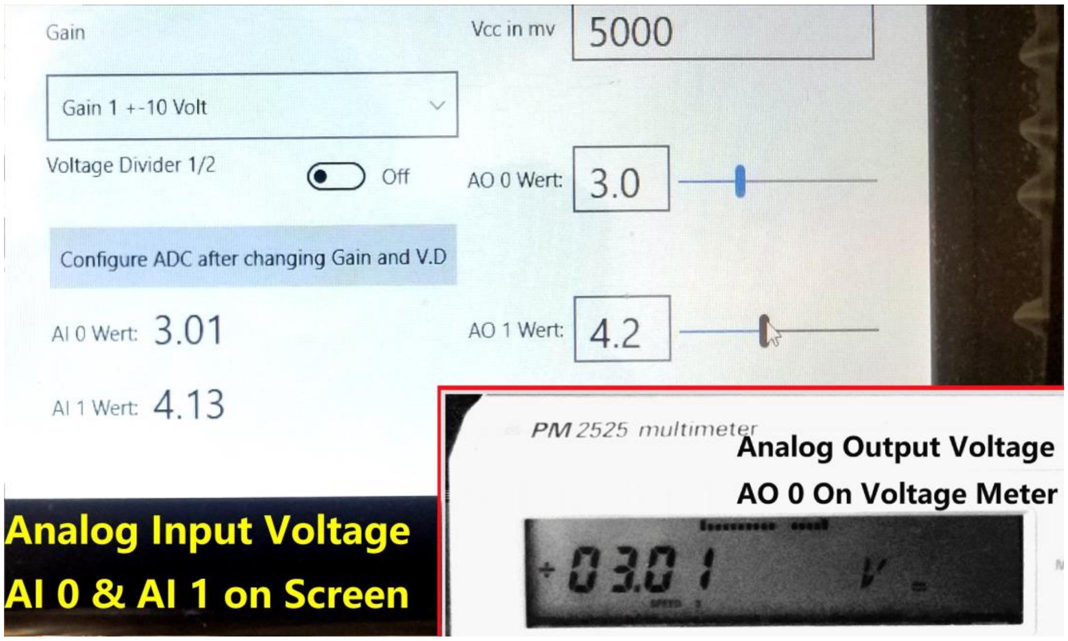
At last, the ADC/DAC communication methods were programmed to write and read the I^2C devices and fulfill specific functionalities. The basic read and write I^2C devices methods were added into the “IReadWriteI2C” and implemented. Later in the PCL project, task-specific methods were created to handle data and display them in the human interface. The methods of ADC/DAC input and output were referred to the example program from Prof. Dr. Bayerlein [42]. Moreover, the detailed tasks of the methods could be seen in Appendix 10.2.3.

Especially, the “*void InitSecondTimer(int interval)*” needs to be emphasized. As noted above, the .NET Framework Class Library provides the “System.Threading.Timer” for Xamarin.Forms UWP. However, if this timer was used to write data into the visual elements such as Label and Entry, a threading error called “The application called an interface that was marshalled for a different thread” would occur. To overcome this problem, another timer called “Device.StartTimer” in Xamarin.Forms was used [41]. Considering the fact that the timer was only used for observing the data, the time interval of 1 second was enough. As a result, the “Device.StartTimer” was adequate for this purpose.

4.4.5 Result of “ADC-DAC” application

This task aims at converting an existing Windows. Form application to the Xamarin.Forms UWP application. The new application initialized and configured the ADCs and DACs, and changed the output voltage of DACs (“AO 0 Wert” and “AO 1 Wert”) by means of touch screen.

The Figure 4-15 illustrates the final result of the UWP application. As can be seen, the voltage meter displayed the output voltage form ADC 0 and ADC 1 (port C and D), and the touch screen displayed the input voltage at the Label “AI 0 Wert” and “AI 0 Wert”. This result proved that the new application worked correctly as expected.


Gain: Gain 1 +-10 Volt

Vcc in mv: 5000

Voltage Divider 1/2: Off

AO 0 Wert: 3.0

Configure ADC after changing Gain and V.D

AI 0 Wert: 3.01

AO 1 Wert: 4.2

AI 1 Wert: 4.13

Figure 4-15: UWP application “ADC-DAC” test result

According the result, Xamarin.Forms provided a relatively good support for the UWP programming. With help of the “DependencyService”, the UWP application was able to interact with other hardware. However, parameters and elements in one platform project seemed to be inaccessible by other platform projects in normal access way. Therefore the same methods had to be programmed again in other platform, which might consumed more resources in the memory.

In brief, the “ADC-DAC” UWP application was capable of controlling 4 I Ꞓ devices at the same time, and the usability as well as robustness were improved. Moreover, the Xamarin.Forms platform improved the cross-platform programming and made large program more maintainable indeed.

4.5 Chart display in Raspberry Pi

Nowadays, many applications provide various diagrams and chart to improve the usability and enable the user to better understand about the status or the functionalities of the application. Jensen and Anderson (1992) stated that charts represent the data graphically, in other words, charts can be used to display information in a comprehensible way [35]. Previous study mainly focused on the programming technology about cross-platform software development. In the contrary, few researches have been done about the visual elements for Xamarin.Forms. So it is interesting to find out whether chart display is also supported by Raspberry Pi and can be integrated into the Xamarin.Forms applications. In this section, the “Microcharts” was evaluated and used for displaying the real-time input voltage curve on the screen.

4.5.1 Problem and solution of chart display on Raspberry Pi

One requirement of this task was that the chart display plugin should be open-source. In that case, 3 third-party plugins were investigated during the test: “OxyPlots”, “XLabs” and “Microcharts”. They are all supports C# application development.

Unfortunately problems exists on both “XLabs” and “OxyPlots”. According to the website of “XLabs” (2017), the plugin is obsolete and unsupported by the latest version of the Xamarin.Forms [36]. In addition, the “XLabs” seems to support only XAML initialization, which means the creation of the “XLabs” charts in C# code may be impossible.

Moreover, the “OxyPlots” has the same problem with the “XLabs”, even though the “OxyPlots” supports the in-code creation of chart views [37], the “.Net Standard” version is obsolete and is incompatible to the Xamarin.Forms.

The last solution is “Microcharts”. Aloisdeniel (2017) introduced that “Microcharts” was an external chart library especially for UWP, Xamarin.Forms, Xamarin.iOS, Xamarin.Android and so on [38]. For instance, “Microcharts” supports bar chart, point chart, line chart, donut chart, radial gauge chart and radar chart. According to the document, “Microcharts” can only be added into the human interface by XAML code, however, it may be the only plugin which is still supported by the .Net Standard and it provides clear and beautiful chart display in human interface. Considering these reasons, the “Microcharts” was used as the chart library for the real-time chart display on the Raspberry Pi.

4.5.2 Implementation of the “Microcharts”

First of all, according to the requirement from the user, a navigation bar was needed for integrating multiple programs into one general application. This could be done by the “MasterDetailPage”. It could have multiple detailed child pages and a master page as a navigation bar on the left. In the project “I2CADDA”, a C# content page called “HomePage.cs” was created in PCL project folder and extended the “MasterDetailPage” class. If one page title is clicked in the navigation bar, the corresponding the child page will be shown on the right hand side. The implementation was referred and modified from the example from Prof. Dr. Bayerlein [42].

Second, the “I2CADDA” solution should be installed with the “Microcharts” packages. In the “Manage NuGet Packages for Solution” window, the “Microcharts” and “Microcharts.Forms” were downloaded and installed on every platform. The packages also included “SkiaSharp” library for specific color display.

Third, in the “MicroChartsView.xaml”, the name space of the “Microcharts” and its chart view was added into the content page. Then in the “MicroChartsView.xmal.cs” file, another “Device.StartTimer” was created to initialize the chart type, add the data and set the property of the chart every 1 second.

Specifically, the home page can pass itself as an argument to its child pages to initialize them. Therefore, if the child page has the instance of the home page, it can access other child pages within the same project folder. In this task, the "MicroChartsView" needed the data of analog input voltage for chart display, but the data was stored as a list of "Microcharts.Entry" in "MainPage.xaml.cs" ("ADC-DAC" program). So the "MicroChartsView" accessed the parameter by the path of "HomePage". The code example for this task is shown in Listing 4-3.

```
//HomePage.cs
//child page initialization in home page
MainPage mp = new MainPage();//ADCDAC Page initialization
MicroChartsView mcv = new MicroChartsView(this);//Micro Charts View initialization with HomePage class
TempControl tc = new TempControl(this);//Temperature Control View initialization with HomePage class
BalCon bc = new BalCon();//Balance Control View
```

```
<!-- MicroChartsView.xaml-->
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:forms="clr-namespace:Microcharts.Forms;assembly=Microcharts.Forms"
             x:Class="I2CADDA.MicroChartsView">
    <ContentPage.Content>
        <StackLayout Orientation="Vertical">
            <forms:ChartView x:Name="Chart1" HeightRequest="200"/>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

```
//MicroChartsView.xaml.cs
public partial class MicroChartsView : ContentPage
    {
        HomePage hp;
        public MicroChartsView(HomePage h)
        {
            hp = h;
            InitializeComponent();
            Device.StartTimer(TimeSpan.FromMilliseconds(1000), () =>
            {
                Chart1.Chart = new LineChart { Entries = hp.mp.entries0, LineMode = LineMode.Straight, MaxValue = 10.0F, MinValue = -10.0F }; //get chart data from MainPage via HomePage
                return true; // True = Repeat again, False = Stop the timer
            });
        }
    }
```

Listing 4-3: Implementation of "MasterDetailPage" and "Microcharts"

One limitation of the “Microcharts” is that the line chart can only have one line at one time. If the user wants to have multiple lines in one chart to compare the value, it is impossible to realize it by adding another data list. One possible solution is to use the “Grid” layout in the XAML and overlay 2 different charts at the same position. If the opacity of the charts are set as 0.5, both lines can be seen clearly and the value can be compared. The disadvantage of this solution is that the labels are also overlaid and unreadable. The two-line chart realized by “Microcharts” is shown in Figure 4-16. It was used in temperature control program which will be discussed in Section 4.6.

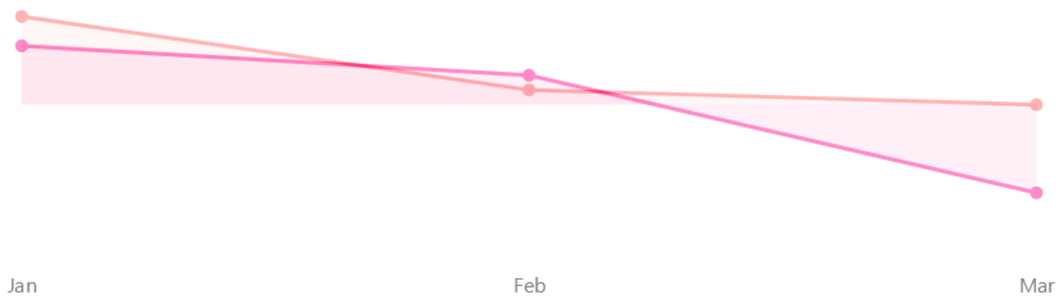


Figure 4-16: Two-line chart realized manually by “Microcharts”

4.5.3 Result of “Microcharts” display on Raspberry Pi

After the implementation of “Microcharts” in “I2CADDA” application, the “ADC-DAC” program was first started and it stored the analog input voltage into 2 lists of “Microcharts.Entry” every 1 second. Finally, the child page of “Micro Charts Display” could show the real-time input voltage of ADC 0 and ADC 1. According to Figure 4-17, the Y axis was voltage value and the X axis was timer information. The curve of the 2 ADCs could be easily readable and comprehensible to the user.

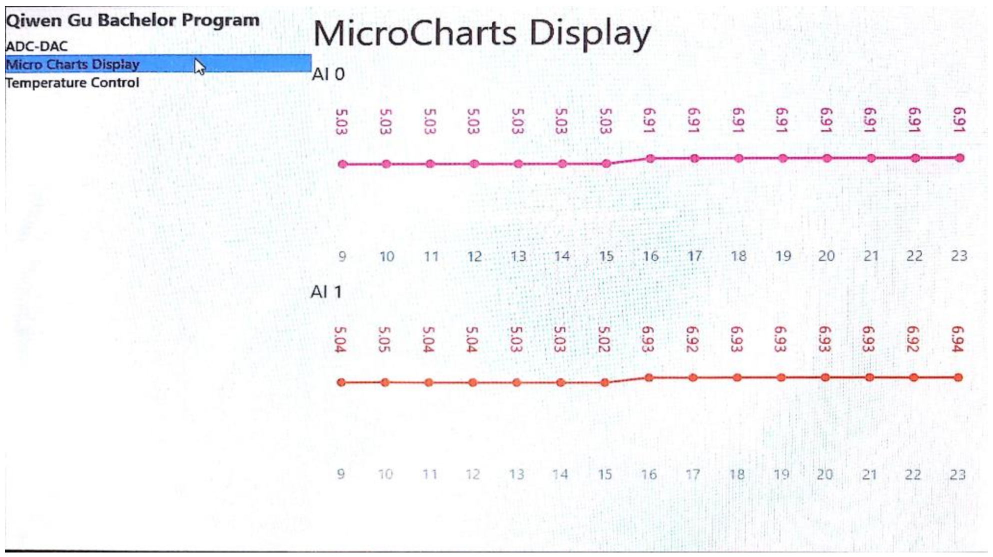


Figure 4-17: “Microcharts” display on Raspberry Pi

Meanwhile, there are 2 disadvantages of the “Microcharts”. First, it can only be added into the content page by XAML, which is against the preference that the application should be realized by C# code as much as possible. Second, the charts types provided by “Microcharts” are limited. For example, the two-line chart is still under development by the author. Despite of these limitations, the “Microcharts” is still one of the most suitable open-source chart libraries available on Xamarin.Forms.

In general, therefore, it seems that the “Microcharts” makes real-time chart display possible on Raspberry Pi. The implementation process is also simple. As a result, “Microcharts” was later used again for the temperature automation control program and Android remote control application as well. In addition, the functionalities of “Microcharts” can be investigated more in further research, for instance, the zooming methods and the in-code creation of the charts needs to be explored.

4.6 Temperature automation control program

Automation control is one of the most important technologies in electrical engineering. Groover (2014) describes that automation is a technology which can control a system and the assistance from human is unnecessary for this process [39]. Moreover, previous research conducted by Singhpannu, Ansari and Gupta (2015) has established that Raspberry Pi with Raspbian operating system was able to realize an autonomous car driving system [2]. However, the possibility of automation control on Raspberry Pi with Windows 10 IoT operating system remains uninvestigated now. This section aimed at converting an existing control system written in C# Windows Forms into Xamarin.Forms UWP and running it on Raspberry Pi. The program interacted with a “Hand Dryer” hardware to control the temperature. During this investigation, the application was first developed and then the ability of Raspberry Pi was evaluated to check whether it was able to control an electrical control system.

4.6.1 Raspberry Pi hardware connection

This program required 2 hardware devices. The first one is the old temperature control device which was mentioned in Section 4.4. As can be seen in Figure 4-18, the temperature control device had 2 ADCs and 2 DACs. A new Raspberry Pi was installed on the device and it was connected with the I Ꞓ devices via the I Ꞓ cables. The temperature control device had 2 analog input nodes, 2 analog output nodes and one ground node, which is illustrated in Figure 4-19.

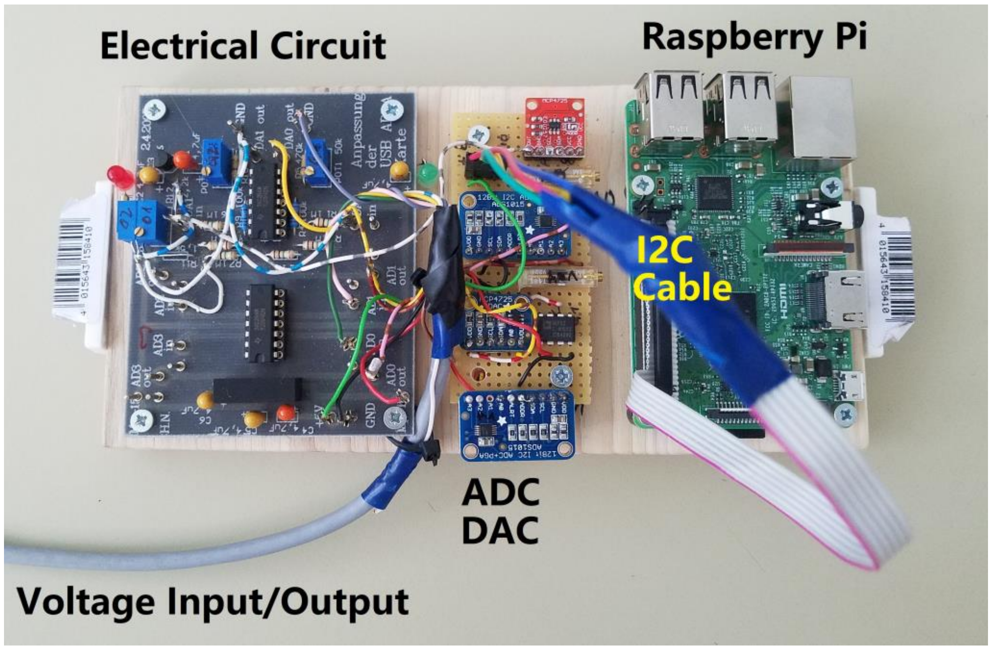


Figure 4-18: Structure of temperature control device

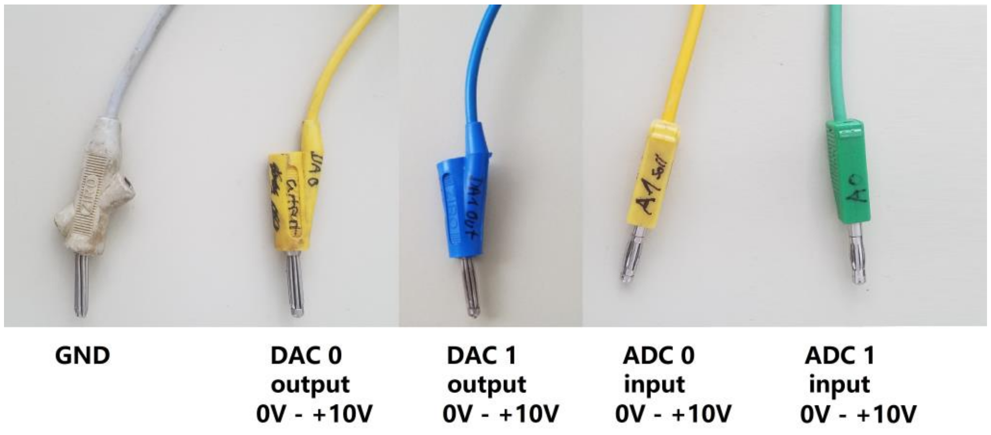


Figure 4-19: The input nodes and output nodes of temperature control device

The structure of the "Hand Dryer" is shown in Figure 4-20, it was composed of an air blower, a heater, a temperature sensor and many other electronic components such as voltage amplifiers and level shifters. According to the requirement, the user could set the desired temperature by the potentiometer on the "Hand Dryer" hardware, then the nodes of the temperature control device were connected to the "Hand Dryer" hardware to read temperature status and output the voltage to control the power of heater. The detailed cable connection can also be seen in Figure 4-20. In addition, the "Hand Dryer" also an electrical feedback circuit, if user toggle the switch to "Analog Mode", the "Hand Dryer" could control itself automatically without Raspberry Pi.

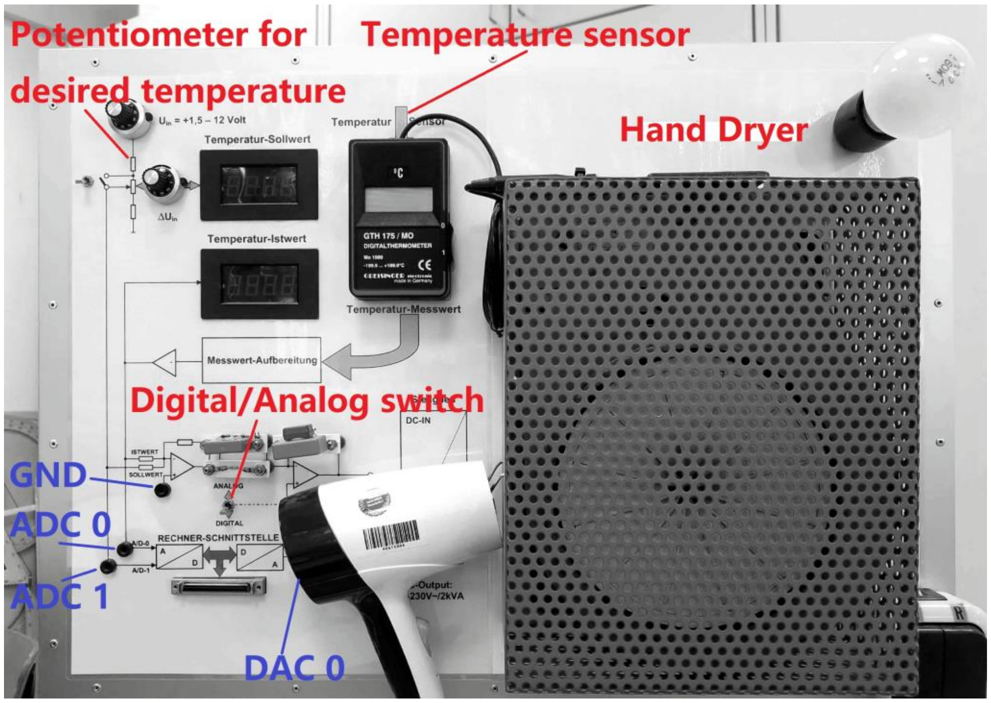


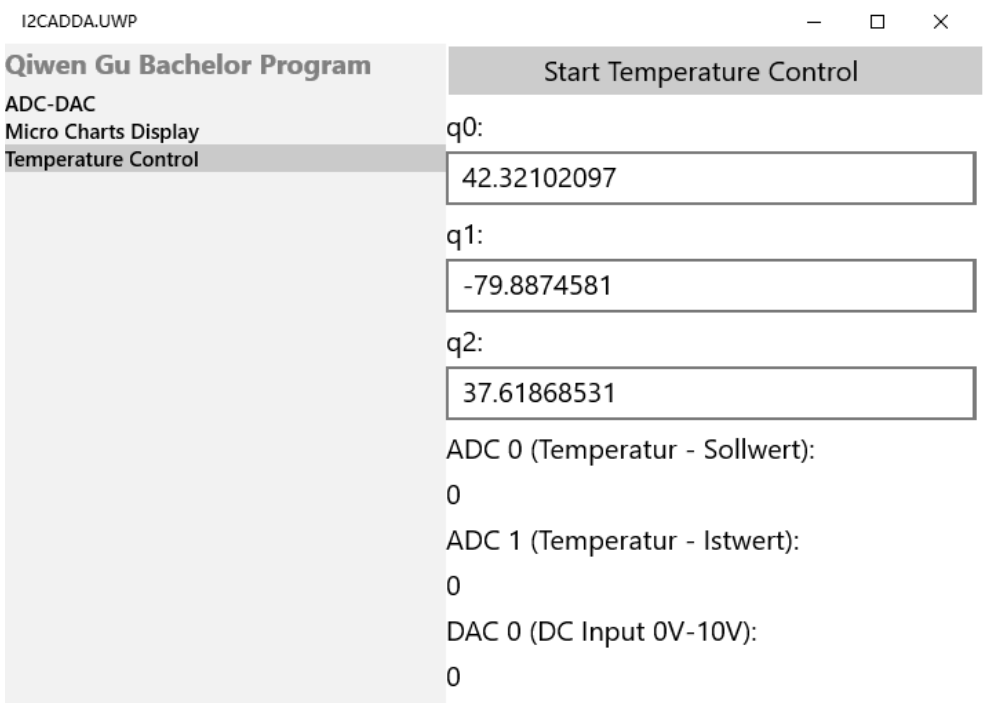
Figure 4-20: Structure and cable connection of "Hand Dryer" device

After the setup process, the program was developed to fulfill the functionality of temperature automation control.

4.6.2 Temperature control program development

The temperature control program “TempControl” was based on the “I2CADDA” application. A new child page was added to the “HomePage.cs”. The “TempControl” could also configure and communicate with the I ²C devices using “DependencyService”. In addition, the “MicroTimer” was modified to perform the automation control according to the requirement.

First, 3 “Entry” fields and a start button were added in the content page of “TempControl.cs”. The “Entry” fields were used for setting the PID (proportional–integral–derivative controller) parameters for the automatic control algorithms. If the start button was clicked, the ADCs would be reconfigured and the “MicroTimer” would be started. Then the Raspberry Pi read ADCs’ input and displayed the input voltage on the screen. The detailed layout of the “TempControl” is shown in Figure 4-21.



q0: 42.32102097
q1: -79.8874581
q2: 37.61868531
ADC 0 (Temperatur - Sollwert): 0
ADC 1 (Temperatur - Istwert): 0
DAC 0 (DC Input 0V-10V): 0

Figure 4-21: Layout of temperature automation control program

Second, the "MicroTimer" in UWP project was modified. The initialization methods got 3 arguments set previously by the user and started the "MicroTimer". The interval was set to 100 milliseconds. In the "*OnTimedEvent*" method, the Raspberry Pi read the current temperature signal from the temperature sensor (ADC 1) and compared it to desired temperature signal (ADC 0) from the potentiometer, then it calculated the output control voltage according to the automation control algorithm [40]. The algorithm is shown in Table 4-5, specifically, "qo", "q1", "q2" were PID (proportional–integral–derivative controller) parameters. "yn1", "xdn1", "xdn2" were used for saving the system status. And "yn" was the final output voltage.

Step	No.	Algorithm method	Description

Initialization	1	yn1 = 0, xdn1 = 0, xdn2 = 0
		yn=0, xdn=0	

Control algorithm loop	2	xdn = ADC 1 – ADC 0	Calculation of control difference
	3	yn = yn1 + q0 * xdn + q1 * xdn1 + q2 * xdn2	
	4	If yn < minimum or yn < maximum: yn = yn – (q1 + q1 + q2) * xdn	Anti-windup mechanism
	5	yn1 = yn, xdn2 = xdn1, xdn1 = xdn	Actualize variables for next loop

Table 4-2: Control algorithm steps

Third, the temperature control device could output the control voltage “yn” via DAC 0 (0V - +10V) to change the power of the heater and keep pace with the desired temperature. What needs to be noticed is that the output voltage of DAC 0 should be limited to 0 volt to +10 volt, and the Step 5 in the table must be placed before limiting the DAC 0 value. Otherwise, error may occur and the automation control program may fail to perform its functionality.

After the development process, the new application was run on Raspberry Pi. The ADC 0, ADC 1 and DAC 0 should be first initialized in page “ADC-DAC”, then the temperature control program could be started in “Temperature Control” page. Later the control procedure was observed for evaluation.

4.6.3 Result of temperature control on Raspberry Pi

This section aims at investigating the ability of the Raspberry Pi for automation control functionality. The time duration of the adjusting procedure was recorded between the 2 stable statuses. During the test, time duration was measured by an electronic stopwatch. And the performance of Raspberry Pi was compared to that of the analog feedback circuit on “Hand Dryer”. As a result, the evaluation of the performance of Raspberry Pi and the application was achieved.

According to the Table 4-6, the Raspberry Pi took on average 25.28 seconds to adjust to the desired temperature, which was about 1 second slower than the analog electrical circuit. So the performance was close to some extent.

	Raspberry Pi	Analog feedback circuit
Average time duration	25.28 s	24.16 s

Table 4-3: Time comparison of temperature control

The results, as shown in Table 4-6, indicated that the Raspberry Pi installed with Windows 10 IoT was able to run an automation control system written in C# on Xamarin.Forms UWP platform. The performance of the application was considerably close to that of the electrical circuit. In addition, the size of Raspberry Pi was small, which might make Raspberry Pi as well as Xamarin.Forms to be the suitable platforms for small electrical engineering projects or experiments.

4.7 Remote Raspberry Pi control with Android smartphone

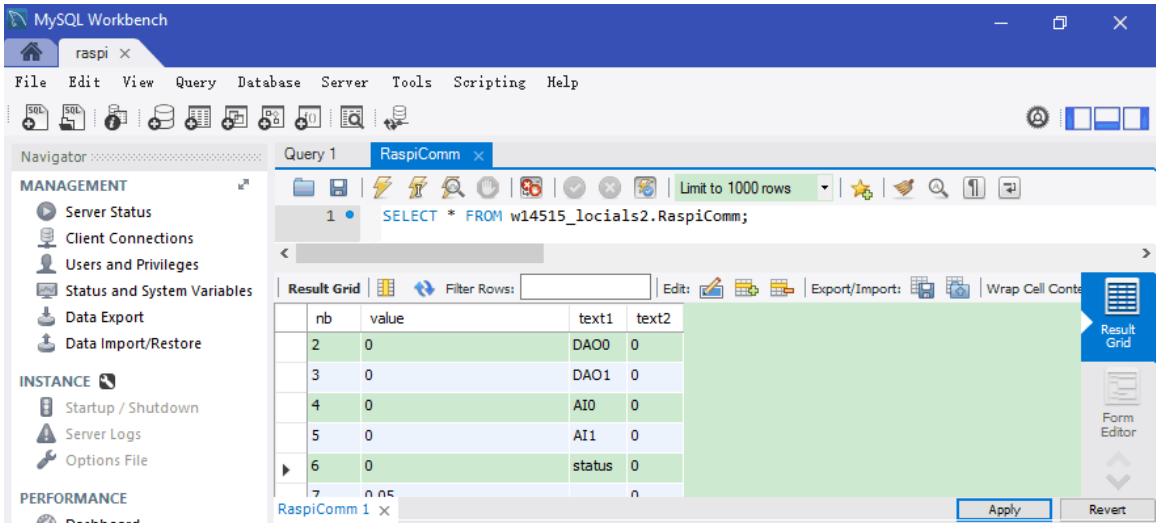
Nowadays, Android smartphone and iPhone occupy the most percentage of the smartphone users, and UWP is also one of the dominant platforms for portable devices. Recently, many applications are released on these popular platforms, so one important requirement for them is that the basic functionalities on every platform should be same. What’s more, applications on different platforms should be able to communicate with each other. Although Xamarin.Forms supports the cross-platform programming, it is still unknown whether the UWP application running on Raspberry Pi can communicate with other platforms' application. In this task, the temperature automation control application was modified and transferred to Android platform. So the Android smartphone can control the Raspberry Pi remotely to adjust the temperature on "Hand Dryer" hardware.

4.7.1 Database connection

In this paper, the Raspberry Pi communicated with Android smartphone via network. Both devices were connected to the Wi-Fi and accessed a shared database for data transmission. Therefore the Android smartphone could send the desired value to the Raspberry Pi.

The setup of the database and the communication with the database was referred to the bachelor thesis of Liu (2017) [18]. Specifically, in PCL project, the interface "IDAL.cs" was added for the "DependencyService" for database connection. In Android project, "MySql.Data.CF" was added as a reference while in UWP project the "MySql.Data.RT" was added. Each platforms implemented the "IDAL.cs" methods in their own "DAL.cs" classes.

In order to use the database, 5 data rows were added into the table "RaspiComm" in MySQL Workbench, including "DAO0", "DAO1", "AI0", "AI1" and "status". According to the Figure 4-22, column "text1" stored the name of data, and column "value" stored the data as double. Column "nb" and "text2" were unnecessary to this application.



```
SELECT * FROM w14515_locials2.RaspiComm;
```

nb	value	text1	text2
2	0	DAO0	0
3	0	DAO1	0
4	0	AI0	0
5	0	AI1	0
6	0	status	0
7	0.05		0

Figure 4-22: Structure of data table "RaspiComm" in MySQL Workbench

As can be seen in Figure 4-23, if the "status" is 0, it means that the Raspberry Pi gets desired temperature from "Hand Dryer" hardware and controls itself. If "status" is 1, the Raspberry Pi is controlled by the smartphone and gets desired temperature from database, that is, the value of "DAO1".

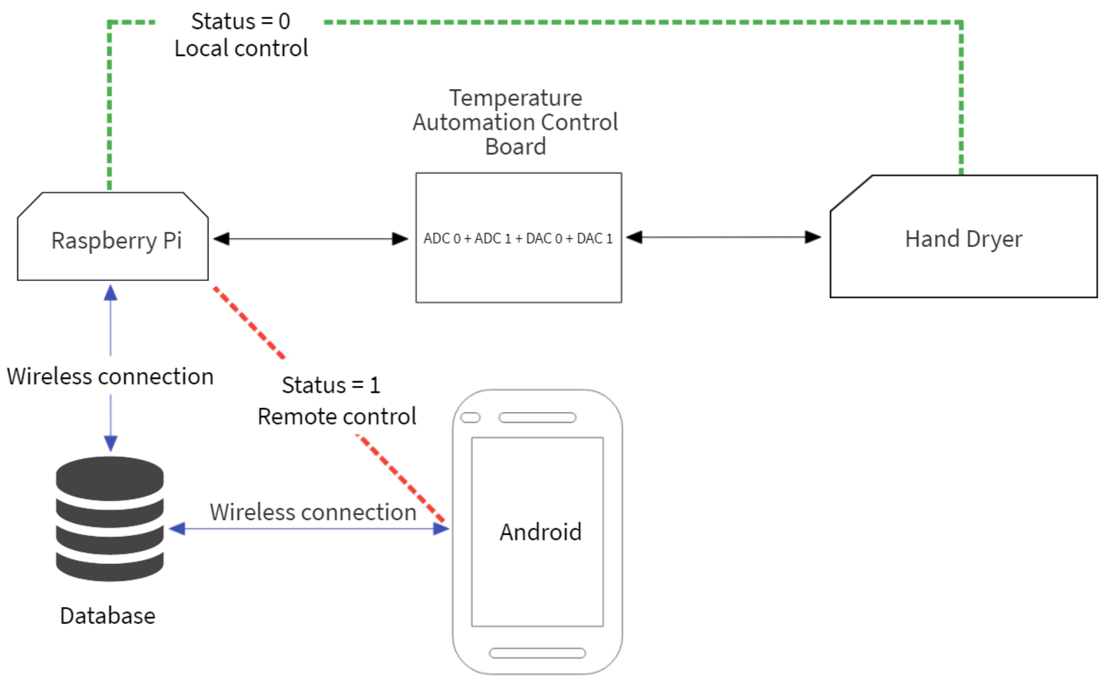


Figure 4-23: Local control and remote control of Raspberry Pi

Furthermore, in the original "DAL.cs" class, the SQL sentence simply added new data to the database and read the latest added data from it. However, if the application requires multiple data, this approach is infeasible because the program is unable to distinguish different data. On the contrary, in this application, the "*GetValue*" and "*StoreValue*" methods in "DAL.cs" were changed, the name of the data was past to the methods as an argument and stored in the database together with the value. So the application could store and get the data according to the name, the specific code part for "*void StoreValue(string name, double data)*" is available in Listing 4-4. Additionally, "*MySqlException*" was implemented in "DAL.cs" to prevent database connection failure caused by poor network signal.

```
public void StoreValue(string name, double data)
        {
            System.Text.EncodingProvider ppp;
            ppp = System.Text.CodePagesEncodingProvider.Instance;
            Encoding.RegisterProvider(ppp);
            string connsqlstring = "Server='ServerName';Port=3306;User Id=UserId;Password=Password;Database=Database1;charset=utf8";
            MySqlConnection mySqlConnection = new MySqlConnection(connsqlstring);
            try
            {
                mySqlConnection.Open();
                string sqlstring = string.Format("UPDATE RaspiComm SET value ={0} WHERE text1='{1}'", data, name);//update values by names
                MySqlCommand cmd = new MySqlCommand(sqlstring, mySqlConnection);
                cmd.ExecuteNonQuery();
                mySqlConnection.Close();
            }
            catch (MySqlException e)
            {
                // perform some action here, and then throw a new exception.
            }
        }
```

Listing 4-4: Code example for “StoreValue”

After the setup of the database and implementation of the database functions, the setup of the hardware with Raspberry Pi will be introduced.

4.7.2 Hardware connection with Raspberry Pi

According to the requirement from the user, the temperature control device should be equipped on the “Hand Dryer” hardware. Since the Raspberry Pi can be controlled remotely and gets desired temperature from “DAO1” in database, it should also show this voltage on the “Hand Dryer” device. Therefore, on the temperature control device, the “DAC 1” was connected to the “ADC 1” on “Hand Dryer” so the desired temperature from the database could also be displayed on the hardware. The detailed connection is shown in Figure 4-24. After hardware connection, the Android application as well as the UWP application could be developed.

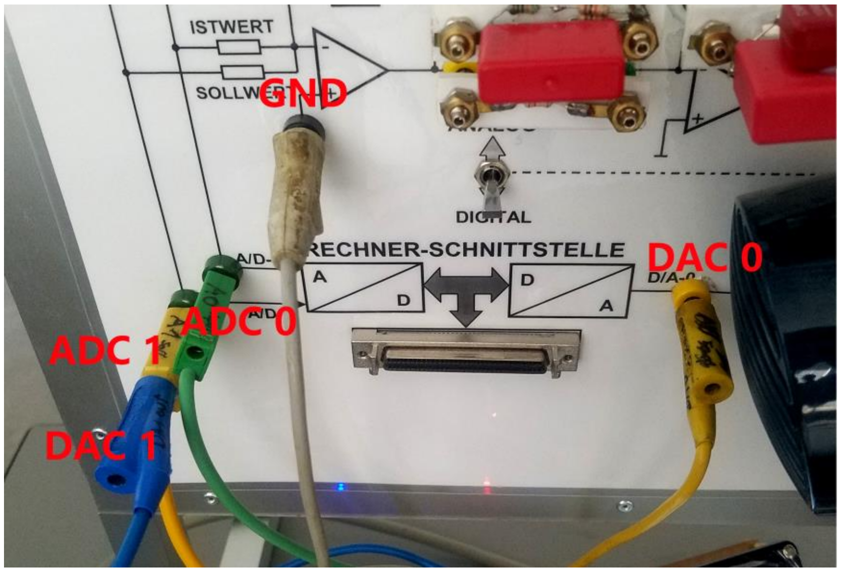


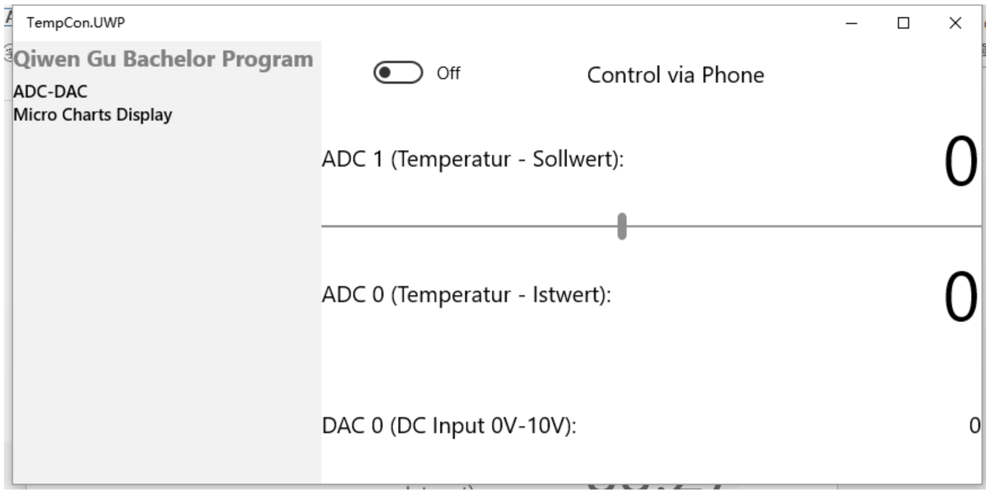
Figure 4-24: Cable connection for remote control

4.7.3 Temperature control application modification

A new project solution called “TempCon” was created, and the main functionalities were same as the previous application in “I2CADDA” project.

In human interface, 2 child pages was added into the “HomePage.cs”: the “ADC-DAC” page and the “Micro Charts Display” page which showed two-line chart about the real-time temperature curves.

According to Figure 4-25, in “ADC-DAC” page, a switch called “Control via phone” was added, if the switch was toggled, the slider under “ADC 1” would be enabled and the user could change the desired temperature value from 0 to 100. Then changed temperature was stored into the database in row “DAO1”



ADC 1 (Temperatur - Sollwert): 0

ADC 0 (Temperatur - Istwert): 0

DAC 0 (DC Input 0V-10V): 0

Figure 4-25: Layout of "ADC-DAC" page

In the “MainPage.xaml.cs”, the program first automatically initialized the I^{2}C devices and reset all the data in database to “0”, including the “status” value. Then it started the “MicroTimer” for temperature control and the “Device.StartTimer” for data display in human interface. In the “Device.StartTimer”, the data were read from the database every 1 second. And the temperature data was stored in lists for “Microcharts” display.

In UWP project, the initialization of “MicroTimer” and the “*OnTimedEvent*” were modified in “MainPage.xaml.cs”. According to Listing 4-5, during initialization, the “MicroTimer” started another “Device.StartTimer” for communication with the database every 1 second.

```
Device.StartTimer(TimeSpan.FromMilliseconds(1000), () =>
            {
                IDAL sqlOperation = DependencyService.Get<IDAL>();
                status = sqlOperation.GetValue("status");
                if (status == 1)  //remote control
                {
                    desiredTemp= sqlOperation.GetValue("DAO1");//get value from data base
                }
                sqlOperation.StoreValue("AI0", x);
                sqlOperation.StoreValue("AI1", w);
                sqlOperation.StoreValue("DAO0", yn);
                return true; // True = Repeat again
            });
```

Listing 4-5: Second "Device.StartTimer" for database communication in UWP application

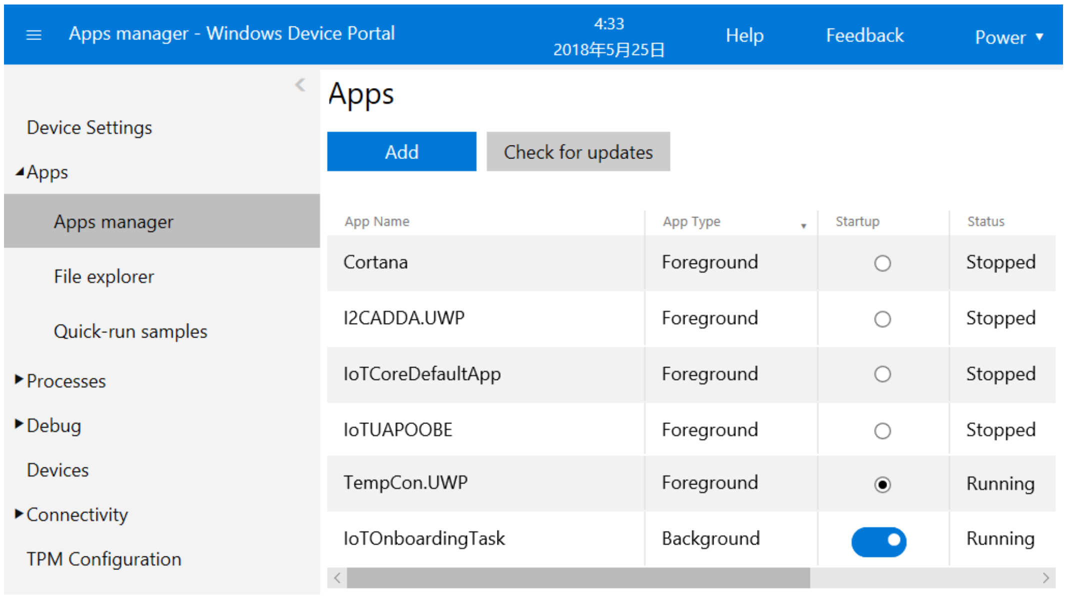
4.7.4 Android application programming

Since the Android smartphone was used only for database communication, it was impossible to be connected with any hardware devices. The Android application shared the basic methods from the PCL project, so after the creation of the Android project in Xamarin.Forms and the realization of database connection methods in “DAL.cs”, the Android application was already finished. The interface “IReadWriteI2C” was implemented only with empty methods to avoid errors.

For Visual Studio debugging, the Android smartphone installed with Android 7.0 was connected to the laptop by USB port. In Android operating system, “Developer Options” and the “USB debugging” should be switched on in the “Setting” page. Then the Visual Studio could detect the Android smartphone and debug the Android application on it.

4.7.5 Result of remote control application

After the development phase, the application was deployed both on Raspberry Pi and Android smartphone. According to the requirement from the user, the Raspberry Pi should automatically run the temperature control application after it starts up. The Figure 4-26 shows the setting of “TempCon.UWP” as the startup application in Windows Device Portal.



Apps manager - Windows Device Portal 4:33 2018年5月25日 Help Feedback Power ▾

Device Settings
◢Apps
Apps manager
File explorer
Quick-run samples
▸Processes
▸Debug
Devices
▸Connectivity
TPM Configuration

Apps

Add | Check for updates

App Name	App Type	Startup	Status
Cortana	Foreground	○	Stopped
I2CADDA.UWP	Foreground	○	Stopped
IoTCoreDefaultApp	Foreground	○	Stopped
IoTUAPOOBE	Foreground	○	Stopped
TempCon.UWP	Foreground	●	Running
IoTOnboardingTask	Background	●	Running

Figure 4-26: Setting of startup application in Windows Device Portal

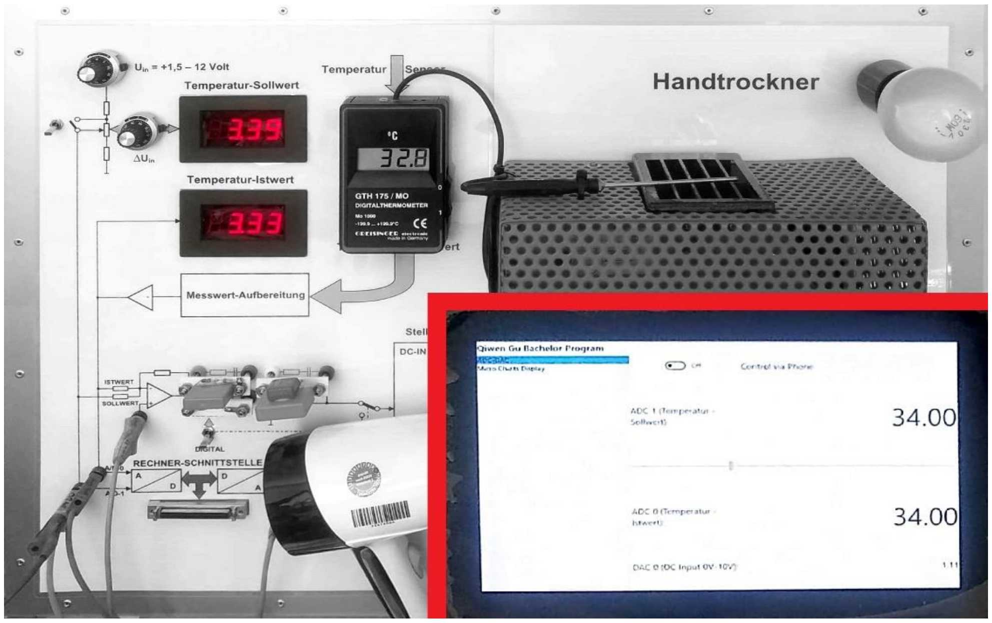
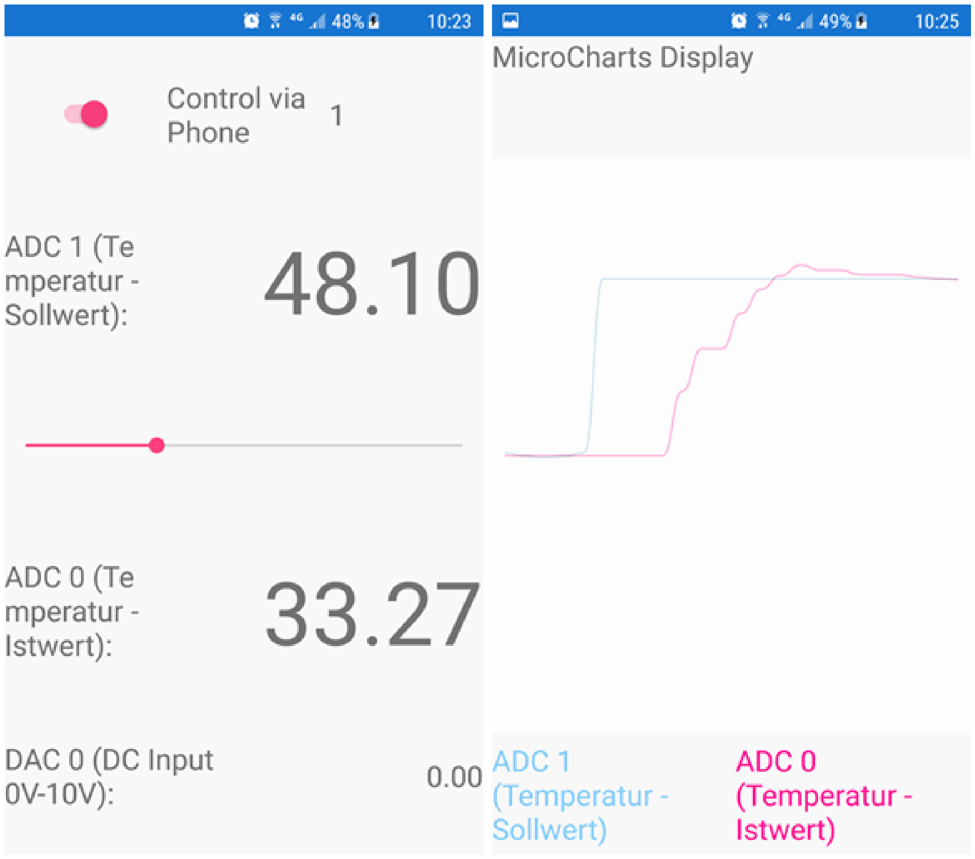


Figure 4-27: Temperature control by Raspberry Pi locally

Finally the remote control of the temperature control application could be tested. Both the Raspberry Pi and Android smartphone should run the application. According to Figure 4-27, in remote control mode, the temperature control was working, and the real-time temperature could be displayed on the touch screen. In addition, the Figure 4-28 shows that the “Microcharts” could also display the real-time temperature curve, which indicated that the temperate control application was working correctly.

However, there is one limitation for the remote control, that is, the performance of the database communication. As was stated by Ken (2013), the workload for the “MicroTimer” must be limited [33]. If “MicroTimer” executes too many SQL operations, the time duration will be longer than the time interval. As a result, performance of temperature control is influenced because “MicroTimer” is unable to calculate the control voltage within 1 millisecond.


Control via Phone 1

ADC 1 (Temperatur - Sollwert): 48.10

ADC 0 (Temperatur - Istwert): 33.27

DAC 0 (DC Input 0V-10V): 0.00

Figure 4-28: Temperature control by Android smartphone remotely

Despite of this disadvantage, the result has shown that the remote control of Raspberry Pi by Android smartphone is possible. If the workload of “MicroTimer” can be rationally designed, the performance of the new “TempCon” application is nearly identical to the previous temperature control application. In general, Xamarin.Forms supports the cross-platform programming as well as cross-platform communication by means of database. Alternatively, for remote control on Raspberry Pi, the way of Bluetooth connection and other methods can be further investigated in later experiments, and see whether these methods can improve the performance of the remote control.

4.8 Balance automation control program

In 2015, Microsoft introduced the Xamarin.Forms to reduce the effort for cross-platform application development. Nowadays many software and programs are developed in C, Python, C# and so on. Especially, in this paper, many original programs were based on Windows Forms in C#. For further application development, it is important to find out whether it is convenient to transfer a Windows Forms application to Xamarin.Forms application. In this section, a balance control program on Windows Forms [17] was transferred to Xamarin.Forms UWP application, and the developed time was recorded. Finally, the effort of the development phase was evaluated to achieve a result.

4.8.1 Hardware connection to the balance control device

First, the Raspberry Pi was connected to the “USB ADDA” device. Then the “USB ADDA” was connected to the balance control device according to Figure 4-29.

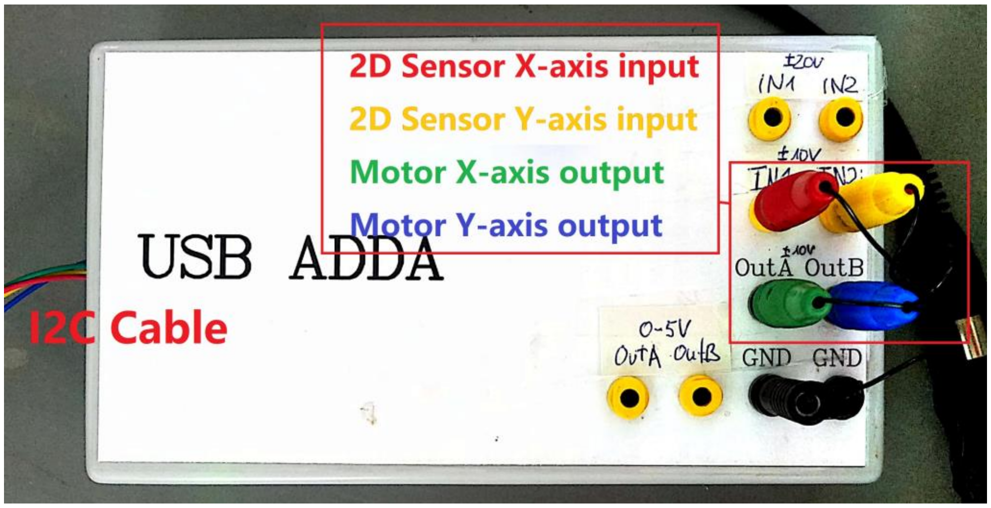


Figure 4-29: Cable connection between "USB ADDA" and balance control device

As can be seen in Figure 4-30, the balance control device had 2 main equipment for balance control: a two-dimensional angle sensor and a pair of motors for two-dimensional position movement. It detected the angle between the stick and the horizontal plane and moved the motors at X and Y axis to keep the stick balanced.

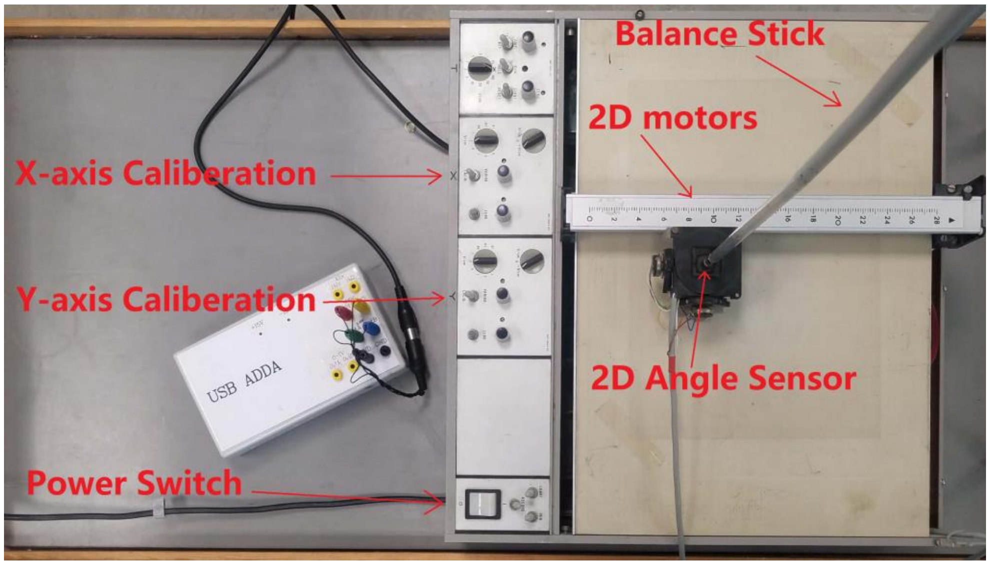


Figure 4-30: Structure of balance control device

4.8.2 Balance control program transformation to Xamarin.Forms

In the original application [17], there was only one C# file containing the codes. However, since the “MicroTimer” was responsible for the balance control and it was only available on UWP platform, the codes were separated into 2 files in Xamarin.Forms UWP project.

In the “I2CADDA” application, a new page called “Balance Control” was added into the “HomePage.cs” and a C# program called “BalCon.cs” was created. As is shown in Figure 4-31, it was responsible for the human interface interaction and system initialization.

On the other hand, the balance control algorithms and all the parameters were placed in “MainPage.xaml.cs” in the UWP project. Therefore, in the PCL project, the “DependencyService” was used again in “BalCon.cs” for setting parameters, reading ADC values and starting “MicroTimer” for balance control every 6 milliseconds.

Since this task aims at evaluating the difficulties of transferring one existing C# program to Xam arin.Forms platform, little emphasis will be put on the code, the detailed code can be found in the USB stick under Path *“./I2CADDA/I2CADDA/I2CADDA.UWP/MainPage.xaml.cs”*.

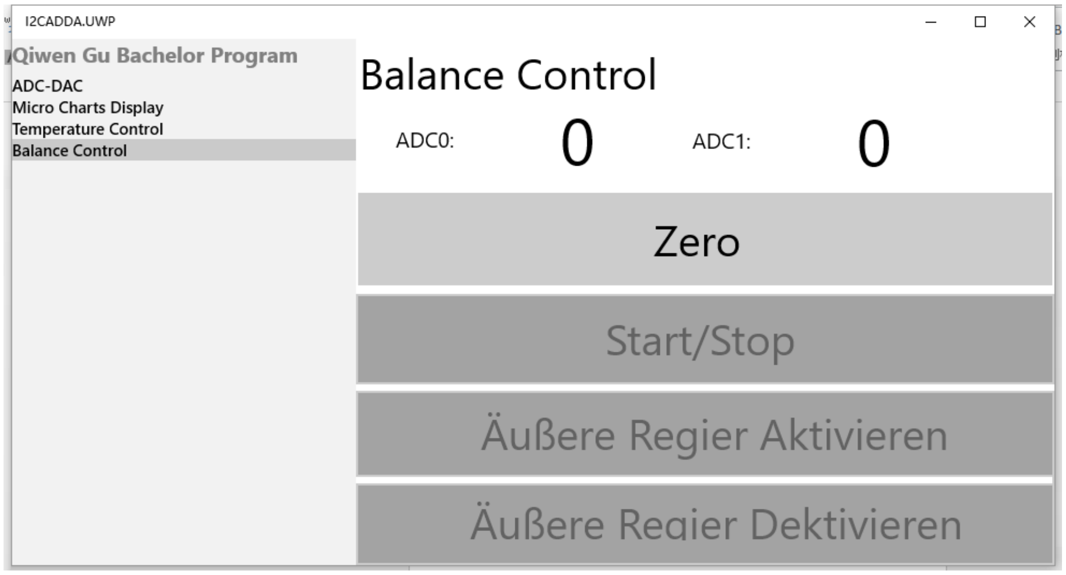


Figure 4-31: Layout of "BalCon"

After the transformation process, the application was deployed to the Raspberry Pi. First, the ADCs and the DACs should be initialized in “ADC-DAC” page. Second, in the “Balance Control” page, the “Zero” button was clicked to move the motors into the middle of the panel. Third, the “Start/Stop” button was clicked to initialize and start the “MicroTimer”. Then if the stick was kept balanced manually, the “Äußer Regier Aktivieren” could be clicked to start the automatic balance control program. Finally, the “Äußer Regier Deaktivieren” could stop the balance control program.

4.8.3 Result of balance control program transformation

The behavior of the balance control program on Raspberry Pi is shown in Figure 4-32. The Raspberry Pi could control the balance, but the oscillation of the motors were larger than original Windows Forms application. The behavior of the “BalCon” was different because the parameters or the automation control algorithm steps might differ from the original one. To fix this problem, additional configuration needs to be done. However, since this task aimed at investigating the difficulties of application transformation to Xamarin.Forms, the main focus was the programming time of the application instead of its functionality and performance. Therefore, the development phase of the balance control program was finished.

The time duration of the development phase was measured by a digital stopwatch. In the end, for an original program which had approximately 500 lines of codes without human interface part, it took nearly 6 hours for transforming, modifying and testing the program on Xamarin.Forms. Most problems might occur during the methods call by “DependencyService” and the parameter configuration for control algorithms.

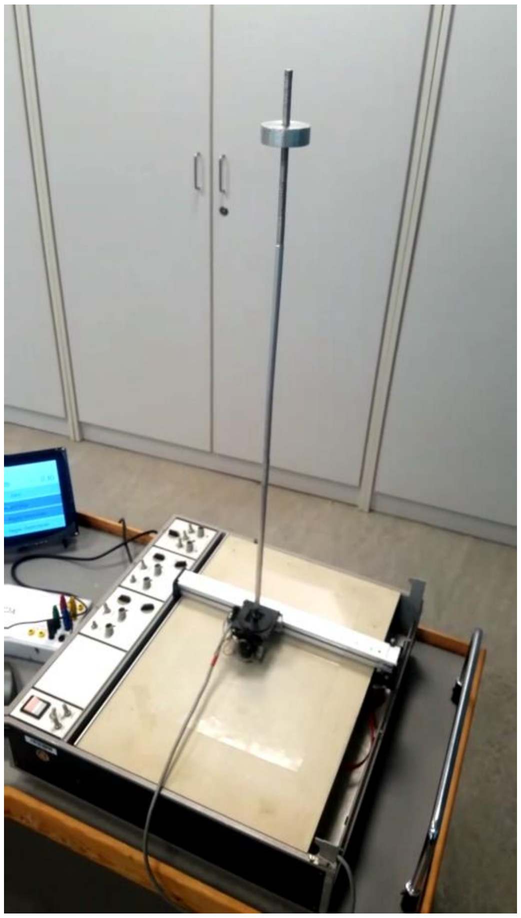


Figure 4-32: Balance control by Raspberry Pi

In Brief, this experiment has indicated that it was convenient to transfer a balance control program to Xamarin.Forms UWP application. However, since the human interface on Xamarin.Forms should be realized in C# code, extra effort is needed to create a new human interface. In addition, the implementation of “DependencyService” is also necessary on Xamarin.Forms. Considering these limitations, it seems that for a large cross-platform application which contains more than 500 lines of codes, the Xamarin.Forms might be a suitable platform for the transformation. The time duration may also depends on the experience of the developer. If the developer is familiar with the Xamarin.Forms, then the effort as well as time duration required for transformation will be considerably acceptable.

5 Summary

In this section, the conclusion as well as the outlook about the investigation of Raspberry Pi programming and Xamarin.Forms are introduced.

5.1 Conclusion and evaluation

The purpose of the this study is to determine the possibilities and limitations of Raspberry Pi programming with Windows 10 IoT operating system using C# and Xamarin.Forms. In this paper, several programs were developed and some practical tests were conducted to evaluate the Raspberry Pi and Xamarin.Forms.

The results of the tasks has shown that the Raspberry Pi and Xamarin.Forms are capable of controlling the I Ꞓ devices and performing complex functionalities such as temperature automation control and balance control. Their performance are similar to the original Windows Forms applications or the electrical circuits and they are better than traditional Python programs on the Raspbian, which was proved by comparing the timing performance to Python and testing the performance of “MicroTimer”.

Moreover, the task of remote Raspberry PI control has proved that Xamarin.Forms supports and improves the cross-platform programming and communication. And the transfer of a C# application from one old platform to Xamarin.Forms is also convenient for the developers. In addition, the implementation of the “Microcharts” in Xamarin.Forms also indicates that it is possible and convenient to extend the functionalities of the Xamarin.Forms by other C# libraries or plugins.

Specifically, from my perspective of view, Xamarin.Forms is a creative solution for application development. The learning process requires some effort, but later during the development phase, it may increase the efficiency of programming. According to the preference of the user, the content page should be initialized by C# code, but this may increase the workload during implementation because some visual elements or tutorials are only available in XAML. So in my opinion, the creation of visual elements by XAML needs further research since this approach supports the Model-View-ViewModel (MVVM) architectural model. It can separate the view from the data and controller, which seems to make the code much clearer and maintainable.

Therefore, the possibility of Xamarin.Forms development on Raspberry Pi with Windows IoT can be achieved.

- Xamarin.Forms offers another approach of cross-platform application development. It makes the code more readable, understandable and maintainable. It can also reduce the workload of the developers because it can separate the platform specific code and provide uniform functionalities among each platform.

- Xamarin.Forms supports C# programming, so this could be a good platform for converting the C# program form old platform and working on portable devices such as Windows 10 IoT.
- Raspberry Pi with Windows IoT can provide sufficient performance to run some complex programs such as automation control, wireless communication and I^{2}C device interaction. Considering the fact that the trend of IoT device is becoming popular, Raspberry Pi could be a suitable portable device for further practical use.
- Xamarin.Forms is extensible with many existing C# libraries and plugins, which can realize many alternative functionalities easily.

However, there still exist some limitations during the experiment.

- The initialization of I^{2}C devices is likely to have threading errors and crash the applications. Therefore specific initialization steps and codes need to be implemented.
- The Xamarin.Forms supports limited UI elements. And visual elements such as check box and radio button are unsupported right now. This may add difficulties during the development.
- Due to the version compatibility of Xamarin.Forms and Windows 10 IoT, some third party libraries or plugins are unsupported anymore.
- It seems to be impossible for Android, iOS and UWP project to access the PCL project and read parameters, so the difficulties of implementation is increased. Same methods may be implemented in several different projects, which consumed more resources and reduce the maintainability.

In conclusion, the evidence of this paper suggests that Raspberry Pi with Windows 10 IoT and Xamarin.Forms is a suitable platforms for cross-platform application development indeed. Even though this is a new platform, it has shown its future potential during the investigation. The findings from this study may make contributions to the developer to understand the characteristic of Raspberry Pi and Xamarin.Forms. The programs in this thesis are practical examples for programming engineering applications for I^{2}C communication and automation control. Finally, they may also provide suggestions about a new approach for software development.

5.2 Outlook

For further research, several factors still remain to be investigated. In addition, some approach in the experiment can be improved.

- The remote control of the Raspberry Pi needs deeper study, a new communication way between different platforms in Xamarin.Forms can be investigated. Right now, the remote control is realized by database communication, but due to the delay of the database connection, the performance of this approach is only suitable for simple applications. For high-performance remote control, other methods can be further investigated, for instance, Bluetooth or infrared ray.
- The remote control with iOS operating system can be implemented in further research to check whether the database commination used in this paper is also feasible on iOS.
- The performance of Raspberry Pi with Xamarin.Forms requires further research, for example, the complex calculation, control statements and database communication can be tested and compared to Python and Raspbian again, so a more detailed understanding about these technologies can be achieved and the developers can decide to use which technologies according to the requirements of the applications.
- The creation of content page in Xamarin.Forms can be studied more. Now some visual elements like "Microcharts" may only be created in XAML. So a comparison of the XAML and C# can be conducted to find out with approach of content page creation is more convenient and extensible.
- In further research, the system safety of the application can be taken into consideration, especially in remote control.

6 Acknowledgement

First, I want to express my sincere thankfulness to my supervisor Prof. Dr. Bayerlein for providing me with all kinds of facilities, environment, advices and continuous supports which help me to finish my bachelor thesis.
In addition, I would like to express my deep gratefulness to M.Sc. Hanesov á for giving various suggestions during my bachelor thesis, and also for her efforts to support me during my study in FHL.
I am also very grateful to my whole family, especially my parents, who keep supporting me in my life and encouraging me to improve myself.
At last, I want to say thank you to my friends, my classmates and anyone who helped me in my life and during my study, which enables me to overcome the difficulties and keep going.

7 Reference

[1] Raspberrypi.org. “Raspberry Pi - Teach, Learn, and Make with Raspberry Pi”, *Raspberry Pi*, 2018. [Online]. Available at: https://www.raspberrypi.org/about. [Accessed 31 May 2018].

[2] G. Singhpannu, A. M. Dawud and P. Gupta. “Design and Implementation of Autonomous Car using Raspberry Pi”. *International Journal of Computer Applications*, 2015, 113(9):22-29.

[3] Ferdoush, S. and Xinrong Li. “Wireless Sensor Network System Design Using Raspberry Pi and Arduino for Environmental Monitoring Applications”, *Procedia Computer Science*, 2014, 34.3-4:103-110.

[4] D. Bowater. “Mini Raspberry Pi computer goes on sale for £22”, *Telegraph.co.uk*, 2012. [Online]. Available at: https://www.telegraph.co.uk/technology/news/9112841/Mini-Raspberry-Pi-computer-goes-on-sale-for-22.html. [Accessed 30 May 2018].

[5] N. Friedman. “Announcing Xamarin 3”, *Xamarin Blog*, 2014. [Online]. Available at: https://blog.xamarin.com/announcing-xamarin-3. [Accessed 31 May 2018].

[6] A. Hern. “Raspberry Pi 3: the credit card-sized 1.2GHz PC that costs $35”, *The Guardian*, 20 16. [Online]. Available at: https://www.theguardian.com/technology/2016/feb/29/raspberry-pi-3-l aunch-computer-uk-bestselling. [Accessed 31 May 2018].

[7] Raspberrypi.org. “Raspbian - Raspberry Pi Documentation”, *Raspberry Pi*, 2018. [Online]. Available at: https://www.raspberrypi.org/documentation/raspbian. [Accessed 31 May 2018].

[8] Raspberrypi.org. “GPIO - Raspberry Pi Documentation”, *Raspberry Pi*, 2018. [Online]. Available at: https://www.raspberrypi.org/documentation/usage/gpio. [Accessed 31 May 2018].

[9] I2C Bus. “I2C - What's That?”, *I2C Bus*, 2018. [Online]. Available at: https://www.i2c-bus.org. [Accessed 31 May 2018]

[10] Microsoft. “I2C transport”, *Microsoft Docs*, 2017. [Online]. Available at: https://docs.microsoft.com/en-us/windows-hardware/drivers/sensors/the-i2c-transport. [Accessed 31 May 2018].

[11] User:Cburnett. “File:I2C.svg”, *Wikimedia Commons*, 2012. [Online]. Available at: https://commons.wikimedia.org/wiki/File:I2C.svg. [Accessed 5 Jun. 2018].

[12] Team Rudra. “Board to Board Communication”, *Medium*, 2013. [Online]. Available at: https://medium.com/@srmmarsroverteam/board-to-board-communication-a6ebe7423a4. [Accessed 31 May 2018].

[13] Microsoft. “Windows.Devices.I2c Namespace - UWP app developer”, *Microsoft Docs*, 2018. [Online]. Available at: https://docs.microsoft.com/en-us/uwp/api/windows.devices.i2c. [Accesse d 31 May 2018].

[14] Microsoft. “Introduction to the C# Language and the .NET Framework”, *Microsoft Docs*, 2015. [Online]. Available at: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework. [Accessed 31 May 2018].

[15] C. Petzold, *Creating Mobile Apps with Xamarin.Forms*, Redmond, USA: Microsoft Press, 2016, p. 6-8.

[16] C. Petzold, *Creating Mobile Apps with Xamarin.Forms*, Redmond, USA: Microsoft Press, 2016, p. 21.

[17] J. Bayerlein, “Toolprogramme Bayerlein\WindfCsharp\SolWINDF 7.7.10.zip”, *Lernraum der Fachhochschule Lübeck*, 2018. [Online]. Available at: https://lernraum.fh-luebeck.de/course/view.php?id=1727. [Accessed: 06-Jun-2018].

[18] H. Liu, “Mobile Phone Cross Platform App development using C# and Xamarin Forms”, *Lernraum der Fachhochschule Lübeck*, 2017. [Online]. Available at: https://lernraum.fh-luebeck.de/mod/folder/view.php?id=85523. [Accessed: 02-Jun-2018].

[19] Microsoft. “.NET Standard”, *Microsoft Docs*, 2017. [Online]. Available at: https://docs.microsoft.com/en-us/dotnet/standard/net-standard. [Accessed 31 May 2018].

[20] Microsoft. “Visual Studio IDE overview”, *Microsoft Docs*, 2018. [Online]. Available at: https://docs.microsoft.com/en-us/visualstudio/ide/visual-studio-ide. [Accessed 31 May 2018].

[21] Microsoft. “Windows 10 IoT Core Dashboard”, *Microsoft Docs*, 2017. [Online]. Available at: https://docs.microsoft.com/en-us/windows/iot-core/connect-your-device/iotdashboard. [Accessed 1 Jun. 2018].

[22] Microsoft. “Use your PC as a mobile hotspot”, *Microsoft Support*, 2017. [Online]. Available at: https://support.microsoft.com/en-us/help/4027762/windows-use-your-pc-as-a-mobile-hotspot. [Accessed 1 Jun. 2018].

[23] Raspberrypi.org. “Raspberry Pi Downloads - Software for the Raspberry Pi”, Raspberry Pi, 2018. [Online]. Available at: https://www.raspberrypi.org/downloads. [Accessed 2 Jun. 2018].

[24] Thonny.org. “Thonny, Python IDE for beginners”, *Thonny*, 2018. [Online]. Available at: http://thonny.org. [Accessed 2 Jun. 2018].

[25] Tutorialspoint. “Python Strings”, *Tutorialspoint*, 2018. [Online]. Available at: https://www.tutorialspoint.com/python/python_strings.htm. [Accessed 2 Jun. 2018].

[26] Microsoft. “Strings (C# Programming Guide)”, *Microsoft Docs*, 2015. [Online]. Available at: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/strings. [Accessed 1 Jun. 2018].

[27] Microsoft. "Using the StringBuilder Class in .NET", *Microsoft Docs*, 2017. [Online]. Available at: https://docs.microsoft.com/en-us/dotnet/standard/base-types/stringbuilder. [Accessed 1 Jun. 2018].

[28] AndiDog. "Are strings pooled in Python", *Stack Overflow*, 2010. [Online]. Available at: http s://stackoverflow.com/questions/2519580/are-strings-pooled-in-python?utm_medium=organic&u tm_source=google_rich_qa&utm_campaign=google_rich_qa. [Accessed 1 Jun. 2018].

[29] J. Bayerlein, "Example I2C Raspi - Xamarin.Forms for Mr. Gu", *Lernraum der Fachhochsc hule Lübeck*, 2018. [Online]. Available at: https://lernraum.fh-luebeck.de/mod/folder/view.php?i d=87019. [Accessed: 03-Jun-2018].

[30] Tutorialsteacher.com. "StringBuilder in C#", *Tutorialsteacher.com*, 2015. [Online]. Available at: http://www.tutorialsteacher.com/csharp/csharp-stringbuilder. [Accessed 2 Jun. 2018].

[31] Microsoft. "Timer Class", *Microsoft Developer Network*, 2018. [Online]. Available at: https://msdn.microsoft.com/en-us/library/system.timers.timer(v=vs.110).aspx. [Accessed 2 Jun. 2018].

[32] Microsoft. "Timer Class", *Microsoft Developer Network*, 2018. [Online]. Available at: https://msdn.microsoft.com/en-us/library/system.threading.timer(v=vs.110).aspx. [Accessed 2 Jun. 2018].

[33] Ken.loveday. "Microsecond and Millisecond C# Timer", *CodeProject*, 2013. [Online]. Avail able at: https://www.codeproject.com/Articles/98346/Microsecond-and-Millisecond-NET-Timer?fid=1581582&df=90&mpp=25&sort=Position&view=Normal&spc=Relaxed&fr=21&prof=True. [Accessed 2 Jun. 2018].

[34] Microsoft. "Introduction to DependencyService", *Microsoft Docs*, 2017. [Online]. Available at: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/dependency-servi ce/introduction. [Accessed 2 Jun. 2018].

[35] C. Jensen and L. Anderson, *Harvard graphics 3: the complete reference*. Berkeley, CA: Osborne McGraw-Hill, 1992, p. 413.

[36] R. Marinho. "XLabs/Xamarin-Forms-Labs", *Github*, 2017. [Online]. Available at: https://github.com/XLabs/Xamarin-Forms-Labs. [Accessed 2 Jun. 2018].

[37] OxyPlot.org. "OxyPlot", *OxyPlot*, 2015. [Online]. Available at: http://www.oxyplot.org. [Ac cessed 2 Jun. 2018].

[38] Al. Deniel. "Aloisdeniel/Microcharts", *Github*, 2017. [Online]. Available at: https://github.com/aloisdeniel/Microcharts. [Accessed 2 Jun. 2018].

[39] M. P. Groover, *Fundamentals of modern manufacturing: materials, processes, and systems*, 4th ed. Hoboken, NJ: J. Wiley & Sons, 2010, p.887.

[40] J. Bayerlein, “Basics of digital PID control”, *Lernraum der Fachhochschule Lübeck*, 2018. [Online]. Available at: https://lernraum.fh-luebeck.de/mod/folder/view.php?id=88841. [Accessed: 02-Jun-2018].

[41] C. Terevinto, “Xamarin Forms - The application called an interface that was marshalled for a different thread”, *Stack Overflow*, 2018. [Online]. Available at: https://stackoverflow.com/questions/50136433/xamarin-forms-the-application-called-an-interface-that-was-marshalled-for-a-di. [Accessed: 05-Jun-2018].

[42] J. Bayerlein, “Example program Bayerlein at meeting 6.2.2018”, *Lernraum der Fachhochschule Lübeck*, 2018. [Online]. Available at: https://lernraum.fh-luebeck.de/mod/folder/view.php?id=85520. [Accessed: 07-Jun-2018].

[43] Microsoft, “Stopwatch Class”, *Microsoft Developer Network*, 2018. [Online]. Available at: https://msdn.microsoft.com/en-us/library/system.diagnostics.stopwatch(v=vs.110).aspx. [Accessed: 12-Jun-2018].

8 List of Figures/Tables/Listings

8.1 List of Figures

8.2 List of Tables

8.3 List of Listings

9 Abbreviation

• **IDE**	Integrated development environment, software used for software development
• **UWP**	Universal Windows Platform, a portable operating system introduced by Microsoft.
• **SDA**	Serial data-line, an I^2C line for data transmission
• **SCL**	Serial clock-line, an I^2C line for clock signal transmission
• **I^2C**	Inter-Integrated Circuit, a technology used for electronic device interconnection
• **PCL**	Portable Class Library, a project share strategy supported by Xamarin.Forms
• **SAP**	Shared Asset Project, a project share strategy supported by Xamarin.Forms
• **DLL**	Dynamic-Link Library, a file storing frequently used code
• **ADC**	Analog to digital converter
• **DAC**	Digital to analog converter
• **PID**	Proportional–integral–derivative controller, an algorithm used in automation control
• **IoT**	Internet of Things, electronic devices which are connected to the Internet and communicate with each other

10 Appendix

The Appendix provides some additional information about the thesis, including the mockup of the application, the methods description about program "ADC-DAC", visual element names in each applications and reference information about the old Windows Forms applications.

10.1 Mockup of the applications

Figure 10-1 shows the mockup design of the program "ADC-DAC" in Section 4.4. The left-hand side displays the master page as the navigation bar and the left-hand side shows the main page of "ADC-DAC".

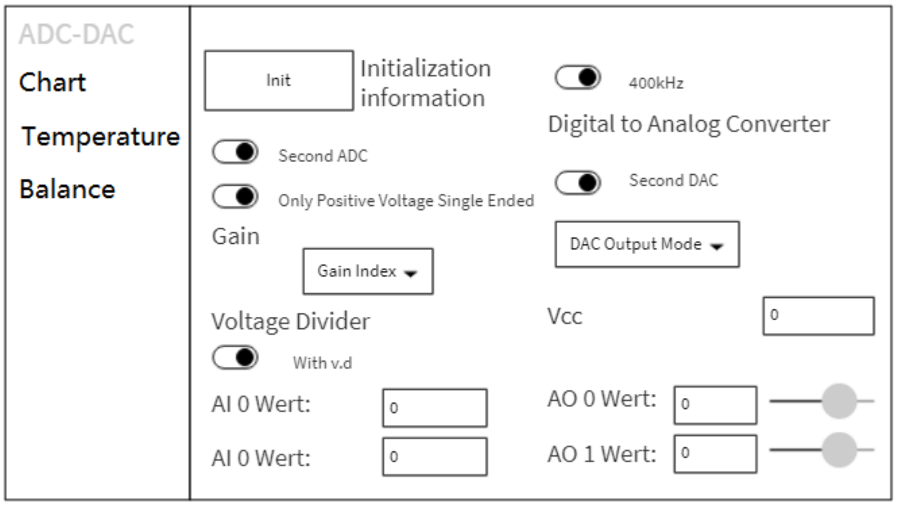


Figure 10-1: Mockup of Page "ADC-DAC"

Figure 10-2 shows the mockup design of the program "MicroChartsView" in Section 4.5. The detailed page shows the "Microcharts" display.

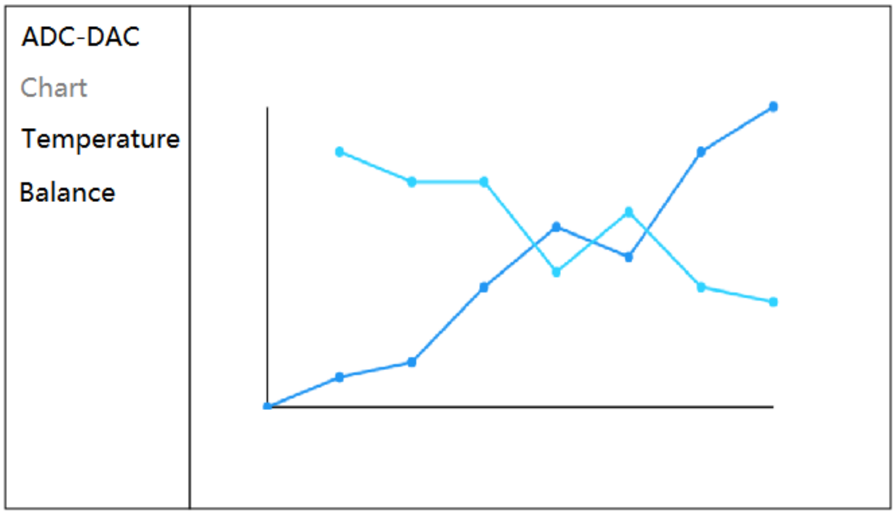


Figure 10-2: Mockup of Page "Chart"

Figure 10-3 shows the mockup design of the program “TempControl” in Section 4.6. The user can click the start button to use Raspberry Pi and control the temperature on “Hand Dryer”.

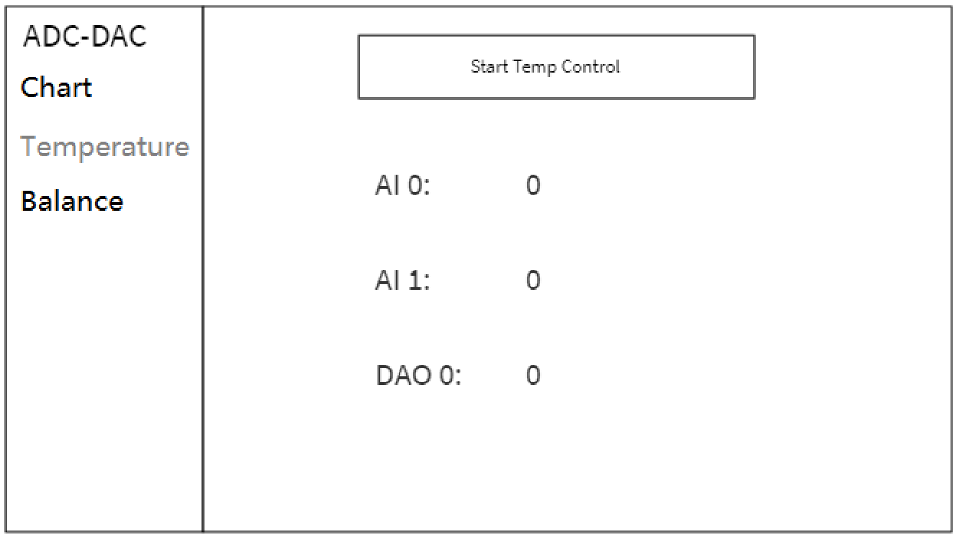


Figure 10-3: Mockup of Page "Temperature"

Figure 10-4 shows the mockup design of the program “BalCon” in Section 4.5. The user can use Raspberry Pi and control the balance on the balance control device.

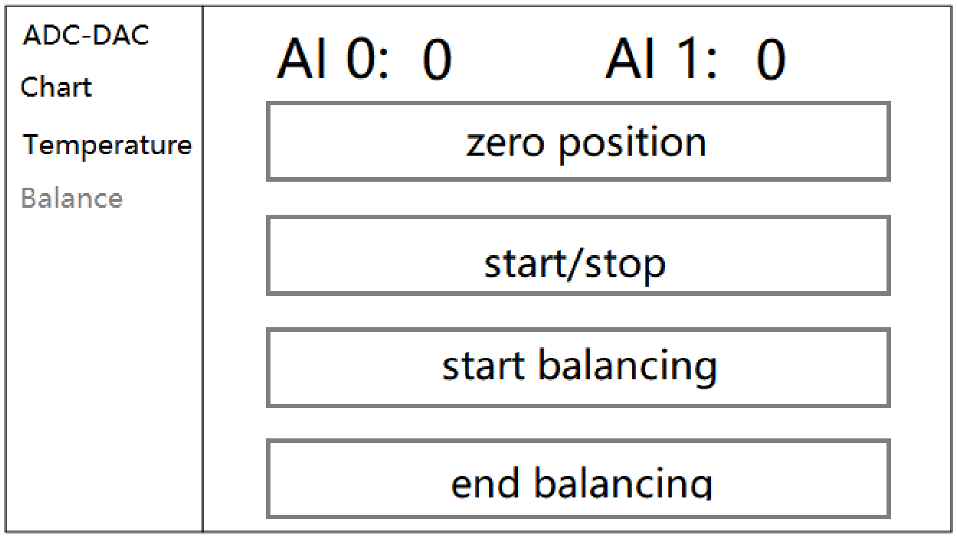


Figure 10-4: Mockup of Page "Balance"

Figure 10-5 shows the mockup design of the application “TempCon” in Section 4.5. The user can use Android to control the Raspberry Pi remotely and control the temperature on “Hand Dryer”. The user can also see the temperature curves on “Chart” page.

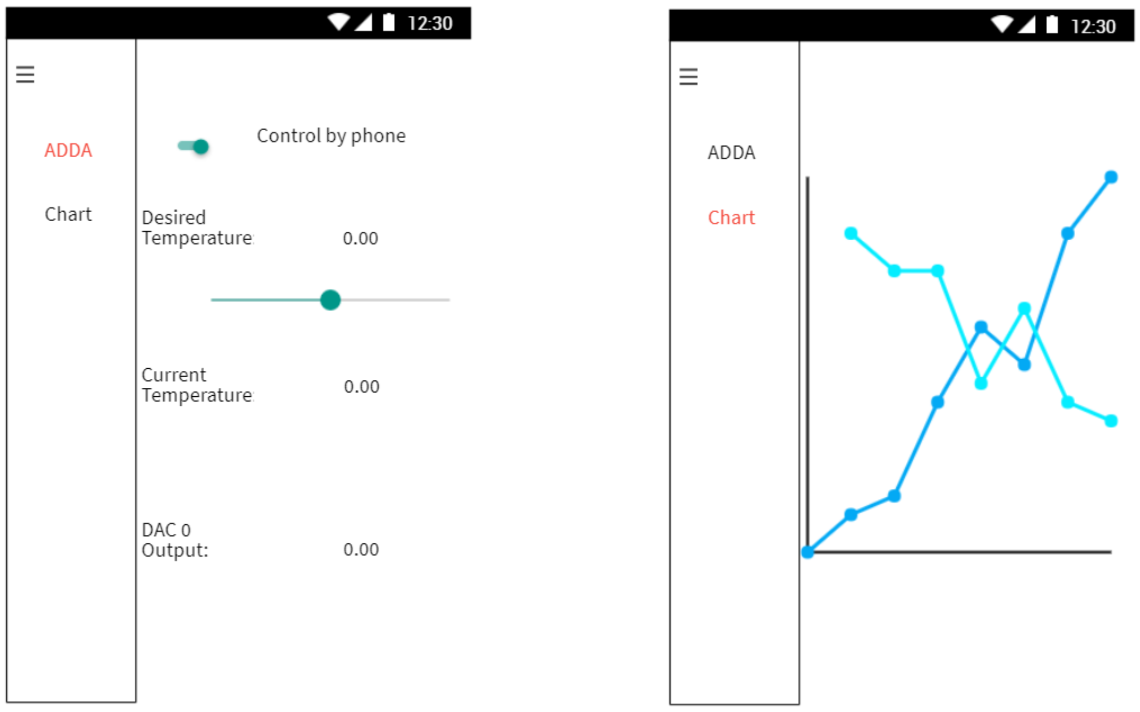


Figure 10-5: Mockup of Android remote control application

10.2 Method descriptions in “ADC-DAC” program

The following tables are the description about the main methods in program “ADC-DAC” which are responsible for human interface interaction, I ℃ device management and ADC/DAC communication.

10.2.1 The human interface interaction methods

No.	Method Name	Description
1	void SliAo0_ValueChanged(object sender, ValueChangedEventArgs e)	Get the change value of the Slider “sliAo0” and show the output value in the “Ao0”, output the corresponding analog voltage at AO 0
2	void SliAo1_ValueChanged(object sender, ValueChangedEventArgs e)	Get the change value of the Slider “sliAo1” and show the output value in the “Ao1”, output the corresponding analog voltage at AO 1
3	void Ao0_TextChanged(object sender, TextChangedEventArgs e)	Process the input value of the Entry “Ao0” and set the corresponding value of Slider “sliAo0”

4	void Ao1_TextChanged(object sender, TextChangedEventArgs e)	Process the input value of the Entry “Ao1” and set the corresponding value of Slider “sliAo1”
5	bool CheckInput(string text)	Get the string of Entry “Ao0”/ “Ao1” and check whether it is a float
6	int ParseString(string text)	Get the string of Entry “Ao0”/ “Ao1” and parse the string to float
7	void DACPicker_SelectedIndexChanged(object sender, EventArgs e)	Get the selected index of the DAC output mode and set the maximum and minimum value of the Slider “sliAo0” and “sliAo1”, reset the timer and clear the ADC input voltage record list “entries0” and “entries1”
8	void VD_Toggled(object sender, ToggledEventArgs e)	Clear the “Gain” picker content and add new items into it, set the corresponding voltage divider gain factor
9	void Volt_SelectedIndexChanged(object sender, EventArgs e)	Get the corresponding gain index
10	void OldDAC_Toggled(object sender, ToggledEventArgs e)	Set the Corresponding DAC 0 address to “0x60” or “0x62”
11	void SecDAC_Toggled(object sender, ToggledEventArgs e)	Set the second DAC view element visible or invisible

Table 10-1: List of human interface interaction methods

10.2.2 The I C device management methods

No.	Method Name	Description
1	void Init_Clicked(object sender, EventArgs e)	Initialize the ADCs and DACs, configure them according to the selected gain, input voltage type and other parameters, initialize and start the application
2	void ConfigADCs()	Configure the ADCs after changing the gain or input voltage type.
3	void ReActive_Clicked(object sender, EventArgs e)	Reconfigure the ADCs and reset the system status (data list for chart display and system timer)

4	void Hz_SelectedIndexChanged(object sender, EventArgs e)	Change the bus speed of the I^2C devices and reset the system status (data list for chart display and system timer)
5	void PositivVol_Toggled(object sender, ToggledEventArgs e)	Change the input voltage type for the old ADDA hardware, reconfigure the ADCs and reset the system status (data list for chart display and system timer)
6	void SecADC_Toggled(object sender, ToggledEventArgs e)	Set the second ADC view element visible or invisible

Table 10-2: List of I2C device management methods

10.2.3 The ADC/DAC communication methods

No.	Method Name	Description
1	void InputI2C(int chan, ref double val)	Read the input voltage from the target ADC and return the value
2	void OutputI2C(int chan, double val)	Write the out voltage to the target DAC
3	void SaveData(string time, double value, int dev)	Save time and input voltage to the Entry list
4	void ShiftLeft(int dev, Entry newElement)	Shift the data in the list to left if the volume of the list exceed to maximum limitation
5	void InitSecondTimer(int interval)	Initialize and start the timer to read data from ADCs and display it in the human interface

Table 10-3: List of ADC/DAC communication methods

10.3 Element names of the applications

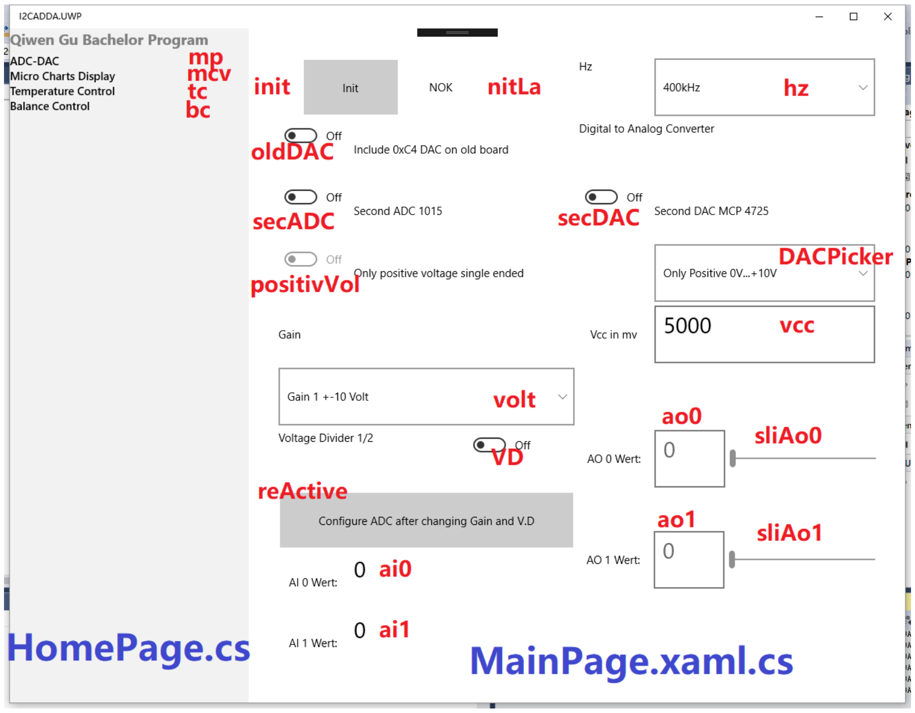


Figure 10-6: Element names in "ADC-DAC"

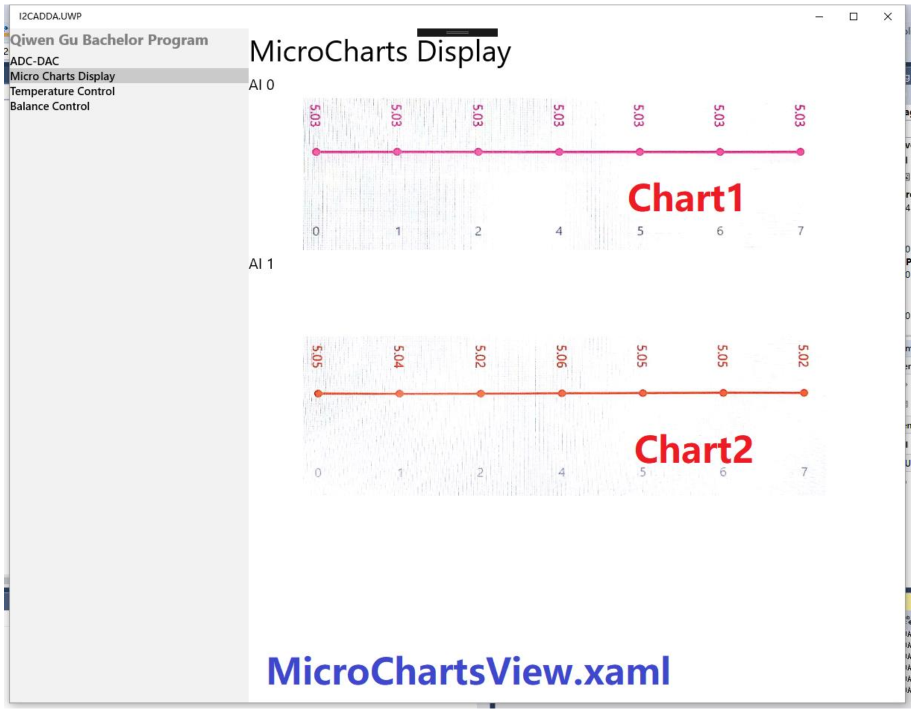


Figure 10-7: Element names in "Micro Charts Display"

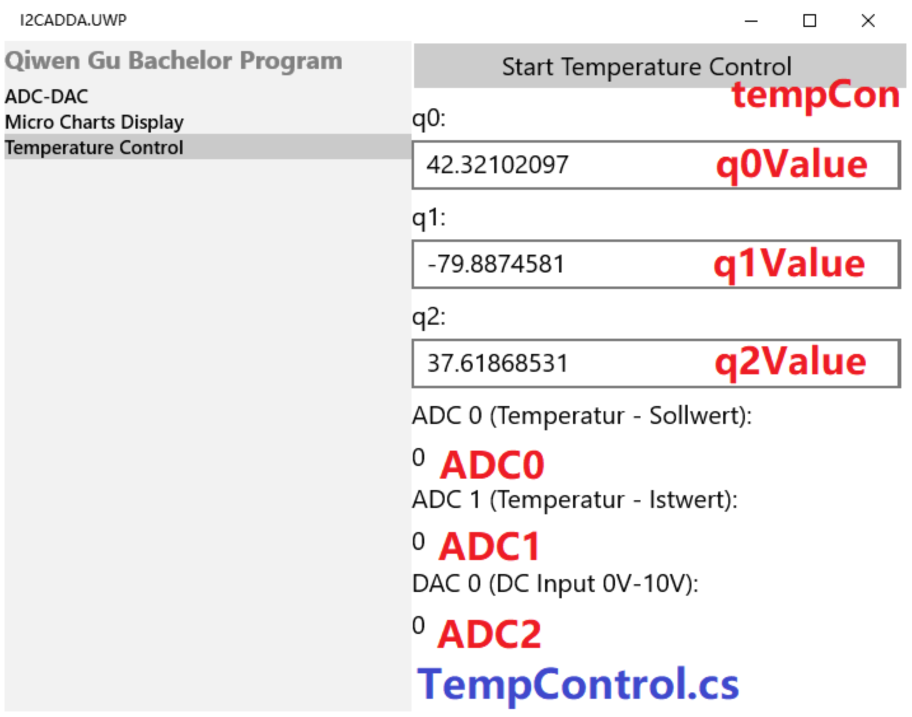


Figure 10-8: Element names in "Temperature Control"

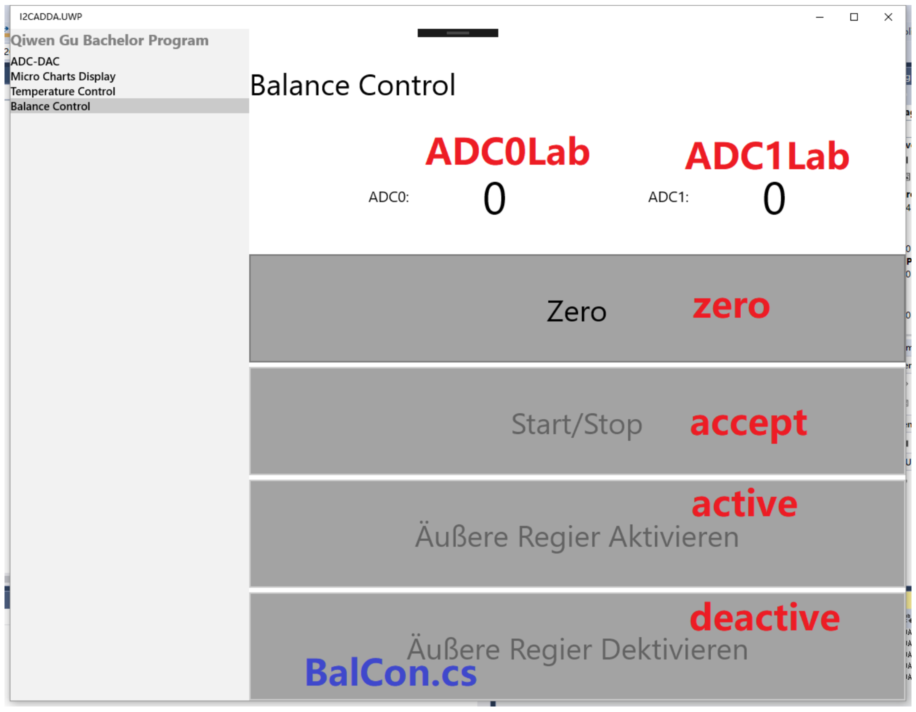


Figure 10-9: Element names in "Balance Control"

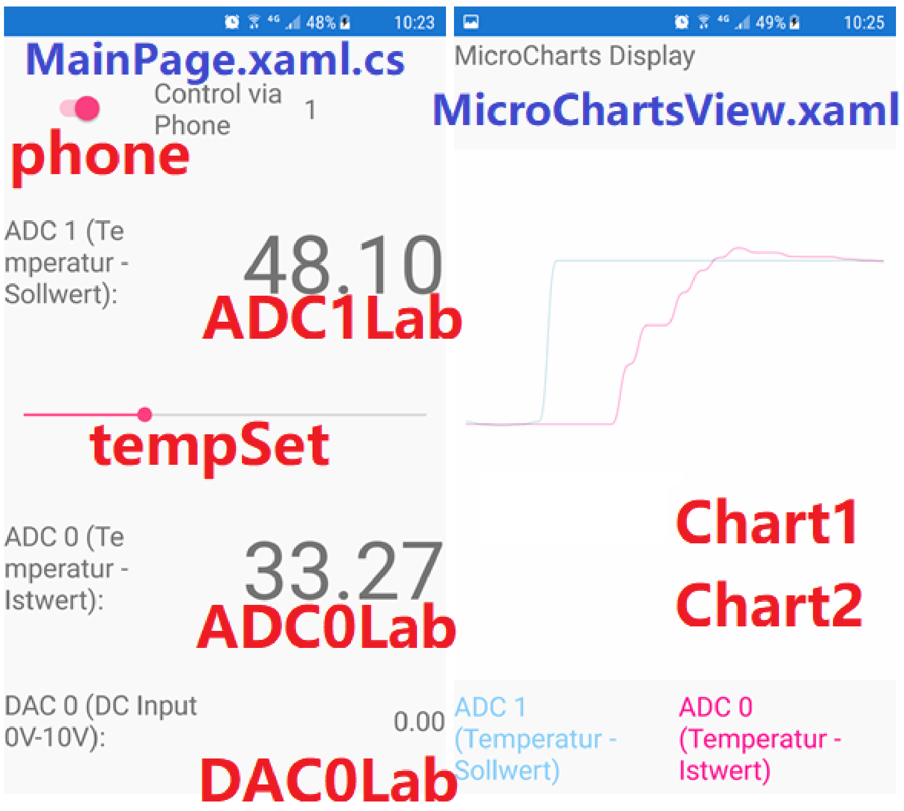


Figure 10-10: Element names in Android remote control "TempCon"

10.4 Information about original Windows Forms application

For checking the reference code in Windows Forms application "SolWINDF 7.7.8", the path of these programs are given as follows.

- "ADC-DAC": "SolWINDF 7.7.8\projWindf\Hardwaretools\I2CCH341.cs"
- "BalCon": "SolWINDF 7.7.8\projWindf\Hardwaretools\Formxypendel.cs"

For checking the reference code in Xamarin.Forms application "AppBayFinal2017 V3 kompakt", the path of these programs are given as follows.

- I C initialization and configuration: "AppBayFinal2017 V3 kompakt\AppBayFinal2017\AppBayFinal2017.UWP\MainPage.xaml.cs", "AppBayFinal2017 V3 kompakt\AppBayFinal2017\AppBayFinal2017\ADC-DAC.cs"
- ADC/DAC input and output methods: "AppBayFinal2017 V3 kompakt\AppBayFinal2017\AppBayFinal2017\ADC-DAC.cs"
- MasterDetailPage: "AppBayFinal2017 V3 kompakt\AppBayFinal2017\AppBayFinal2017\PageMain.cs", "AppBayFinal2017 V3 kompakt\AppBayFinal2017\AppBayFinal2017\PageFetchData.cs"