Entwicklung eines Mikrocontroller-basierten Kalkulators

Im Zeitalter der digitalen Technologie streben Ingenieure und Entwickler ständig danach, Werkzeuge und Geräte zu verbessern, die unseren Alltag und unsere Arbeitsweise beeinflussen. Das hier vorgestellte Projekt konzentriert sich auf die Entwicklung eines Mikrocontroller-basierten Kalkulators, der die Lücke zwischen traditionellen Taschenrechnern und der Rechenleistung moderner PCs überbrückt. Durch die Verwendung eines Arduino Nano in Verbindung mit einer speziell entwickelten PC-Anwendung bietet dieser Kalkulator eine innovative Plattform für die Durchführung algebraischer Berechnungen über eine serielle Schnittstelle.

Projektziel

Das Hauptziel dieses Projekts war die Schaffung eines benutzerfreundlichen, effizienten und technologisch fortschrittlichen Kalkulators, der die Vorteile der Mikrocontroller-Technologie nutzt, um eine direkte Interaktion mit dem PC zu ermöglichen. Indem wir die intuitive Bedienbarkeit einer grafischen Benutzeroberfläche mit der präzisen Berechnungskapazität eines Mikrocontrollers kombinieren, zielen wir darauf ab, ein einzigartiges Werkzeug für Bildungszwecke, Ingenieurwesen und den täglichen Gebrauch zu schaffen.

Technische Umsetzung

Die technische Umsetzung des Projekts gliederte sich in zwei Hauptkomponenten: die Entwicklung einer PC-Anwendung mit Qt Creator und die Programmierung des Arduino Nano.

- **PC-Anwendung (Qt Creator):** Qt Creator bot die idealen Voraussetzungen, um eine ansprechende und intuitive Benutzeroberfläche zu gestalten. Diese Oberfläche ermöglicht es Benutzern, ihre Berechnungen visuell einzugeben und direkt Feedback zu erhalten. Die Anwendung unterstützt grundlegende arithmetische Operationen wie Addition, Subtraktion, Multiplikation und Division sowie erweiterte algebraische Funktionen.
- **Mikrocontroller-Programmierung (Arduino Nano):** Der Arduino Nano wurde aufgrund seiner Kompaktheit, Flexibilität und Leistungsfähigkeit ausgewählt. Die Programmierung in C ermöglichte eine effiziente Verarbeitung der Berechnungen und eine stabile Kommunikation über die serielle Schnittstelle. Besondere Aufmerksamkeit galt der Optimierung der Algorithmen für schnelle Antwortzeiten und der Implementierung einer robusten Fehlerbehandlung.

Herausforderungen und Lösungsstrategien

Die Realisierung des Projekts stellte uns vor mehrere Herausforderungen, insbesondere in Bezug auf die serielle Kommunikation, die Benutzerinteraktion und die Flexibilität der Port-Auswahl.

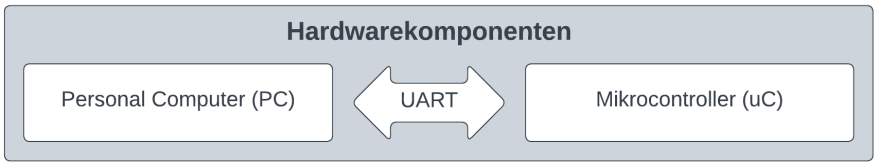
- **Serielle Kommunikation:** Die Gewährleistung einer stabilen und zuverlässigen Verbindung zwischen dem PC und dem Arduino Nano war eine der Hauptaufgaben. Die Herausforderung lag in der Entwicklung eines Mechanismus, der eine kontinuierliche Überprüfung und gegebenenfalls eine automatische Wiederherstellung der Verbindung ermöglicht. Ein spezieller Überwachungsalgorithmus wurde implementiert, um die Verbindungsstabilität laufend zu überprüfen und die Verbindung bei Bedarf zu reinitialisieren.
- **Benutzereingabevalidierung:** Eine weitere signifikante Herausforderung bestand in der Notwendigkeit, Benutzereingaben präzise zu validieren, um die korrekte und sichere Verarbeitung der Berechnungsaufträge zu gewährleisten. Durch den Einsatz von Filtern und Validierungslogiken konnten wir sicherstellen, dass nur valide algebraische Ausdrücke an den Mikrocontroller gesendet werden.
- **Port-Auswahl Flexibilität:** Anfangs war die Auswahl des Kommunikationsports zwischen dem PC und dem Mikrocontroller direkt im Code festgelegt. Diese starre Implementierung führte jedoch zu einem Mangel an Flexibilität, da der Port nicht dynamisch angepasst werden konnte, um unterschiedliche Hardware-Konfigurationen oder Benutzerpräferenzen zu berücksichtigen. Die Lösung dieses Problems erforderte die Entwicklung einer Benutzeroberfläche innerhalb der PC-Anwendung, die es dem Benutzer ermöglicht, den gewünschten Port aus einer Liste verfügbarer Ports auszuwählen. Diese Anpassung machte das System wesentlich benutzerfreundlicher und flexibler in der Anwendung, indem es eine einfache Anpassung an verschiedene Arbeitsumgebungen und Hardware-Setups ermöglichte.

Zusammenfassung und Ausblick

Das Projekt demonstriert erfolgreich die Entwicklung eines Mikrocontroller-basierten Kalkulators, der nicht nur technische Herausforderungen wie die serielle Kommunikation, die Benutzereingabevalidierung und die Flexibilität der Port-Auswahl überwindet, sondern auch eine intuitive und effiziente Plattform für die Durchführung algebraischer Berechnungen bietet. Die Fähigkeit, auf Benutzerfeedback und technische Anforderungen mit angepassten Lösungen zu reagieren, unterstreicht das Potenzial dieses Projekts, als nützliches Werkzeug in Bildung, Forschung und Alltag zu dienen. Zukünftige Erweiterungen könnten die Integration weiterführender mathematischer Funktionen und die Entwicklung einer mobilen Anwendung einschließen, um die Zugänglichkeit und Vielseitigkeit des Kalkulators zu erweitern.

Konzeptionsphase:
Implementierung eines einfachen Kalkulators auf einem Mikrocontroller

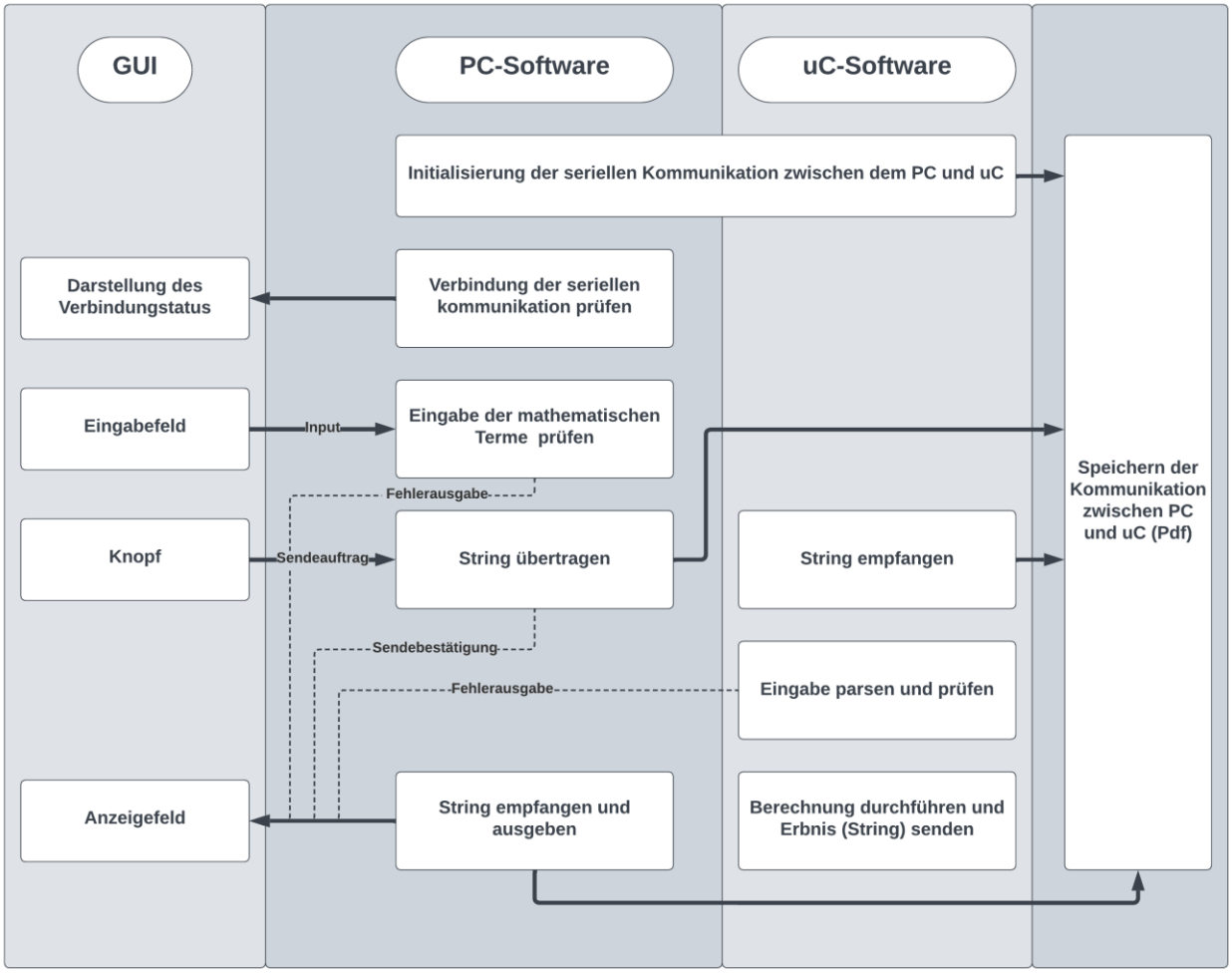
Ziel dieser Aufgabe ist es, einen Mikrocontroller zur Durchführung grundlegender algebraischer Berechnungen mit zwei Operanden zu verwenden. Dies geschieht über eine Schnittstelle (UART, RS232), die zwischen einem Personal Computer (PC) und einer Mikrocontroller-Platine (uC) eingesetzt wird. Der Mikrocontroller (Arduino) soll in der Lage sein, mathematische Ausdrücke zu interpretieren, zu berechnen und das Ergebnis zur Visualisierung an den PC zurückzusenden.



Softwarestruktur

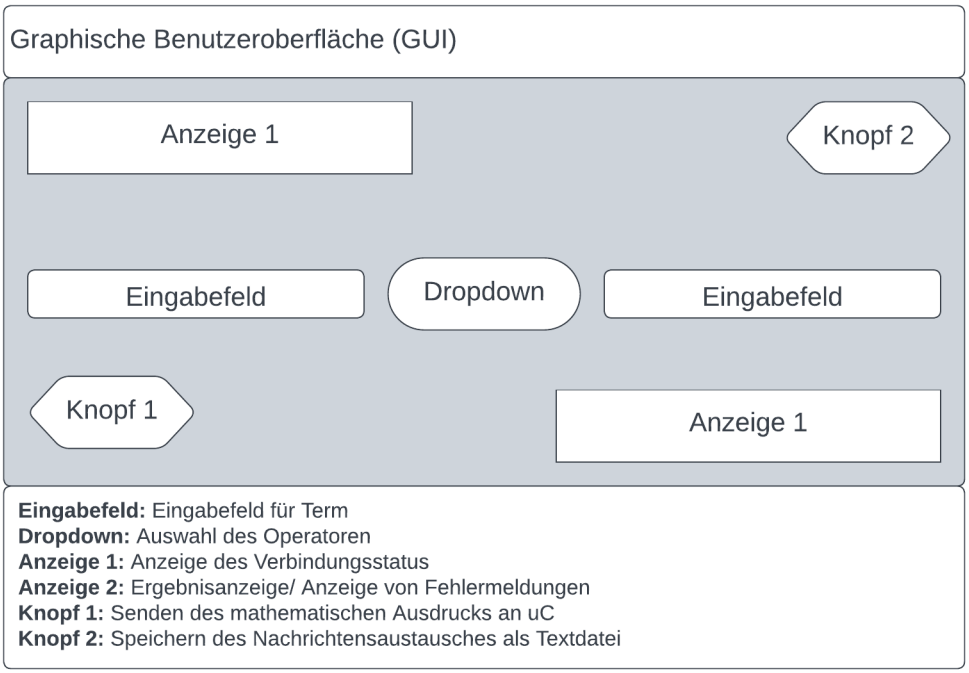
Die Softwarestruktur besteht aus zwei Hauptkomponenten: einer Anwendung für den PC und einer für den Mikrocontroller.

- PC-Anwendung: Diese ermöglicht es dem Benutzer, mathematische Ausdrücke einzugeben und an den Mikrocontroller zu senden. Vor dem senden des Ausdrucks wird die Eingabe geprüft. Anschließend wird das vom Mikrocontroller berechnete Ergebnis empfangen und auf dem PC dargestellt.
- Mikrocontroller-Anwendung: Diese empfängt die mathematischen Ausdrücke von der PC-Anwendung, prüft diesen, führt die Berechnungen durch und sendet die Ergebnisse zurück.



Aufbau der Benutzeroberfläche

Die graphische Benutzeroberfläche soll dem Nutzer ständig Informationen über den Verbindungsstatus zwischen PC und Mikrocontroller bereitstellen. Die Eingabe der algebraischen Berechnungen erfolgt über zwei textbasierte Eingabefelder für die Operanden und ein Dropdown-Menü für den Operator. Fehlermeldungen sowie die Ergebnisse der Berechnungen werden im selben Fenster angezeigt. Zusätzlich wird ein Knopf integriert, der den Nachrichtenaustausch zwischen PC und Mikrocontroller als Textdatei speichert.



Wahl der Programmiersprache und Entwicklungsumgebung

Für die PC-Software wurde C++ als Programmiersprache gewählt und wird in der integrierten Entwicklungsumgebung (IDE) Qt Creator verwendet. C++ bietet eine objektorientierte Herangehensweise, die die Strukturierung der Software und die Handhabung ihrer Komplexität vereinfacht. Die umfangreiche Standardbibliothek von C++ erleichtert die Implementierung der UART-Kommunikation (Bspw. QSerialPort) und der String-Verarbeitung. Die höhere Abstraktionsebene von C++ im Vergleich zu C erleichtert die Implementierung der Benutzerinteraktion und der Datenverarbeitung. Der Mikrocontroller, ein Arduino Micro, wird in der Arduino IDE mit C programmiert.

Erarbeitungsphase:

Implementierung eines einfachen Kalkulators auf einem Mikrocontroller

Das Dokument der Erarbeitungsphase ist in folgenden drei Teile gegliedert.

- **Die PC-Anwendung**
- **Der Arduino Code**
- **Weiterführende Informationen**

Die bereitgestellten Codes im Dokument sind auf GitHub verfügbar, wo sie eingesehen und heruntergeladen werden können. Darüber hinaus bietet die README.md-Datei auf GitHub zusätzliche essentielle Informationen zur Installation und Ausführung der Codes.

https://github.com/timoKlieIU/einfacher-Kalkulator-auf-Mikrocontroller.git

Die PC-Anwendung

Das PC-Programm für den Mikrocontroller-basierten Kalkulator lässt sich in mehrere Schlüsselsegmente oder Module unterteilen, die jeweils spezifische Aufgaben erfüllen. Eine Übersicht und Erklärung der Hauptsegmente des Codes werden folgend aufgezeigt

1. **Integration der Header-Dateien (Bibliotheken)**

Der Code benötigt die Einbindung verschiedener Header-Dateien in das C++ Projekt, das mit Qt, einem Framework für die plattformübergreifende Entwicklung von GUI-Anwendungen, entwickelt wurde. Jede **#include**-Direktive fügt die Deklarationen und Definitionen aus einer anderen Datei oder Bibliothek hinzu, damit sie im Code verwendet werden können. Hier ist eine kurze Erklärung, welche Header-Dateien benötigt werden und wofür diese verwendet werden:

- **#include "MainWindow.h"**: Bindet die Deklaration der **MainWindow**-Klasse ein, die die Haupt-Benutzeroberfläche der Anwendung definiert.
- **#include "ui_MainWindow.h"**: Bezieht sich auf eine durch das Qt User Interface Compiler (uic) generierte Datei, die das UI-Layout für **MainWindow** definiert, das in einer .ui-Datei erstellt wurde.
- **#include "QSerialPort"**: Ermöglicht die Kommunikation mit seriellen Ports, was für Anwendungen nützlich ist, die mit externen Geräten über Standards wie RS-232/UART kommunizieren.
- **#include "QSerialPortInfo"**: Stellt Hilfsfunktionen zur Verfügung, um Informationen über verfügbare serielle Ports zu erhalten, z.B. zur Identifizierung angeschlossener Geräte.
- **#include "QTimer"**: Wird verwendet, um Aktionen nach einem festgelegten Zeitintervall auszuführen, z.B. regelmäßige Updates in der Benutzeroberfläche oder zeitgesteuerte Aufgaben.
- **#include "QFile"**: Ermöglicht das Lesen von und das Schreiben in Dateien, was für viele Anwendungen essentiell ist, da sie mit Dateisystemen arbeiten, bei der erstellen des Nachrichtenverlaufs.
- **#include "QTextStream"**: Bietet eine bequeme Schnittstelle für das Lesen und Schreiben von Text in Dateien, wobei automatische Konvertierungen zwischen Textkodierungen unterstützt werden.
- **#include "QDesktopServices"**: Ermöglicht den Zugriff auf allgemeine Desktop-Funktionen, wie das Öffnen einer URL im Standard-Webbrowser oder das Anzeigen eines Dateiordners im Datei-Explorer. Hier wird es zum Öffnen der Textdatei verwendet.
- **#include "QUrl"**: Repräsentiert eine URL (Uniform Resource Locator) und bietet Funktionen zur einfachen Handhabung und Manipulation von URLs. Es wird verwendet, um die Log-Datei mit einem Desktop-Service zu öffnen.
- **#include "QDir"**: Bietet Funktionen zum Navigieren und Manipulieren des Dateisystems, z.B. zum Auflisten von Dateien in einem Verzeichnis oder zum Erstellen und Löschen von Verzeichnissen. Es wird im Code dafür verwendet, um den Pfad für die Log-Datei zu bestimmen

```cpp
#include "MainWindow.h"
#include "ui_MainWindow.h"
#include "QSerialPort"
#include "QSerialPortInfo"
#include "QTimer"
#include "QFile"
#include "QTextStream"
#include "QDesktopServices"
#include "QUrl"
#include "QDir"
```

2. Initialisierung und Setup

- **MainWindow Konstruktor und Destruktor:** Hier wird die Benutzeroberfläche und die serielle Kommunikation initialisiert. Die Verbindungseinstellungen für den seriellen Port werden festgelegt (z.B. Portname, Baudrate), und es wird ein Timer gestartet, der für die regelmäßig Überprüfung der Verbindung genutzt wird. Darüber hinaus werden die Widgets der Benutzeroberfläche initialisiert beispielsweise die Knöpfe und das Auswahlfenster des Operators.

```cpp
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    setupSerialPort();

    connect(&serialPort, &QSerialPort::readyRead, this, &MainWindow::readSerialData);

    timer = new QTimer(this);
    connect(timer, &QTimer::timeout, this, &MainWindow::checkConnection);
    timer->start(1000);

    connect(ui->saveLogButton, &QPushButton::clicked, this, &MainWindow::onSaveLogClicked);

    ui->comboBox->addItem("+");
    ui->comboBox->addItem("-");
    ui->comboBox->addItem("*");
    ui->comboBox->addItem("/");

    connect(ui->pushButton, &QPushButton::clicked, this, &MainWindow::performCalculation);
}

MainWindow::~MainWindow()
{
    if (serialPort.isOpen()) {
        serialPort.close();
    }
    delete ui;
}
```

- **setupSerialPort():** Diese Funktion konfiguriert den seriellen Port mit spezifischen Parametern wie Baudrate und Datenbits. Fehler beim Öffnen des Ports werden durch eine Statusmeldung angezeigt.

```cpp
void MainWindow::setupSerialPort() {
    serialPort.setPortName("COM9");
    serialPort.setBaudRate(QSerialPort::Baud9600);
    serialPort.setDataBits(QSerialPort::Data8);
    serialPort.setParity(QSerialPort::NoParity);
    serialPort.setStopBits(QSerialPort::OneStop);
    serialPort.setFlowControl(QSerialPort::NoFlowControl);

    if (!serialPort.open(QIODevice::ReadWrite)) {
        ui->statuslabel->setText("Kein Gerät angeschlossen oder Port nicht verfügbar.");
    } else {
        ui->statuslabel->setText("Port ist geöffnet.");
    }
}
```

3. Serielle Kommunikation und Verbindungstest

- **checkConnection():** Überprüft periodisch die Verbindung durch Senden einer Testnachricht. Bei Verbindungsabbruch wird der Port geschlossen und erneut versucht, die Verbindung herzustellen.

```
void MainWindow::checkConnection() {
    if (serialPort.isOpen()) {
        if (serialPort.write("Test\n") == -1) {
            ui->statuslabel->setText("Verbindung unterbrochen.");
            serialPort.close();
            setupSerialPort();
        }
    }
}
```

- **readSerialData():** Liest eingehende Daten vom seriellen Port und aktualisiert die Benutzeroberfläche mit den empfangenen Daten.

```
void MainWindow::readSerialData() {
    QByteArray data = serialPort.readAll();
    QString receivedData = QString::fromUtf8(data);
    ui->inputlabel->setText(receivedData);
    messageLog.append("Empfangen: " + receivedData);
}
```

4. Benutzerinteraktion und Logik

- **performCalculation():** Diese Funktion wird ausgelöst, wenn der Benutzer eine Berechnung anfordert. Sie liest die Eingaben der Benutzeroberfläche, führt die gewählte Operation aus, und sendet das Ergebnis über den seriellen Port.

```
void MainWindow::performCalculation() {
    QString num1 = ui->lineEdit->text();
    QString num2 = ui->lineEdit_2->text();
    QString operation = ui->comboBox->currentText();

    bool ok1, ok2;
    double val1 = num1.toDouble(&ok1);
    double val2 = num2.toDouble(&ok2);

    if (!ok1 || !ok2) {
        ui->inputlabel->setText("Error: Ungültige Eingabe");
        return;
    }

    QString toSend = num1 + " " + operation + " " + num2 + "\n";
    if (serialPort.isOpen() && serialPort.isWritable()) {
        serialPort.write(toSend.toUtf8());
        ui->statuslabel->setText("gesendet.");
        messageLog.append("Gesendet: " + toSend);
    } else {
        ui->statuslabel->setText("Kann nicht senden: keine Verbindung.");
    }
}
```

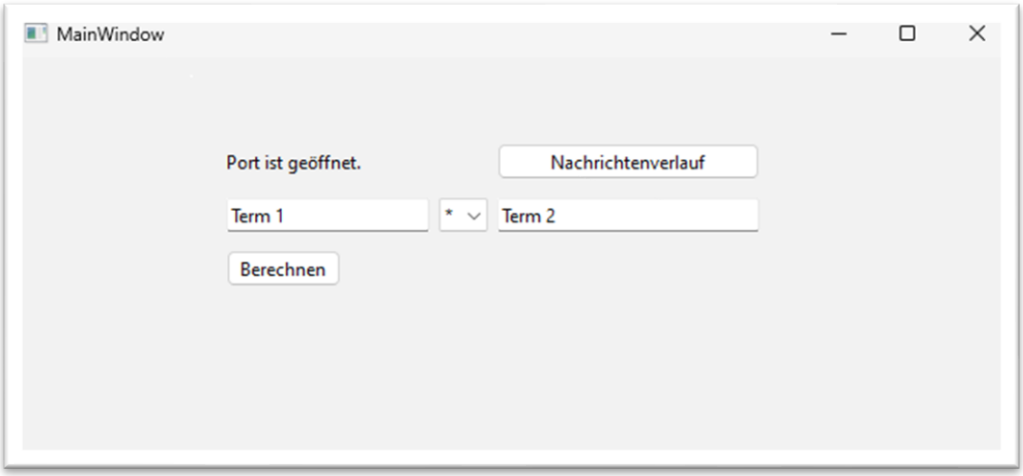
- **onSaveLogClicked():** Speichert die Protokolldaten der Nachrichtenkommunikation in einer Datei und öffnet diese nach dem Speichern.

```
void MainWindow::onSaveLogClicked() {
    QString filePath = QDir::currentPath() + "/nachrichtenaustausch_log.txt";
    QFile file(filePath);
    if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QTextStream out(&file);
        for (const QString &message : messageLog) {
            out << message << "\n";
        }
        file.close();S
        ui->statuslabel->setText("Nachrichtenaustausch gespeichert und wird geöffnet.");
        QDesktopServices::openUrl(QUrl::fromLocalFile(filePath));
    } else {
        ui->statuslabel->setText("Fehler beim Speichern der Datei.");
    }
}
```

5. Benutzeroberfläche

Der Code initialisiert die Benutzeroberfläche mit verschiedenen Elementen wie Buttons, Textfeldern und einem Dropdown-Menü für die Auswahl der Operationen. Die Benutzeroberfläche dient als Schnittstelle

für die Eingabe von Daten und die Anzeige von Ergebnissen oder Statusmeldungen. Folgend wir der „einfache“ Aufbau der generierten Benutzeroberfläche.



Header-Datei

Im Folgenden wird die Header-Datei wiedergegeben in dem die einzelnen Klassen definiert werden.

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QSerialPort>
#include <QTimer>


QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void performCalculation();
    void readSerialData();
    void onSaveLogClicked();
    void checkConnection();

private:
    Ui::MainWindow *ui;
    QSerialPort serialPort;
    QTimer *timer;
    QStringList messageLog;
    void setupSerialPort();
};

#endif
```

Jedes Modul des Programms arbeitet zusammen, um eine funktionierende Anwendung zu erstellen, die Berechnungen durchführt und diese über einen Mikrocontroller mit angeschlossenem seriellem Port verarbeitet. Die Benutzeroberfläche ermöglicht die Interaktion des Benutzers mit dem Programm, während die Hintergrundlogik und die serielle Kommunikation die Berechnungen und den Datenaustausch mit einem externen Gerät handhaben. Die Struktur des Codes folgt einem klaren Aufbau, der es ermöglicht, einzelne Teile unabhängig voneinander zu testen und zu modifizieren, ohne die Funktionalität des Gesamtprogramms zu beeinträchtigen.

Der Arduino-Code

Der Arduino-Code ermöglicht es, einfache mathematische Berechnungen durchzuführen, die über die serielle Schnittstelle empfangen werden. Die Struktur des Codes gliedert sich in drei Hauptteile: setup(), loop(), und processMessage().

```
// Initialisiere die serielle Kommunikation im setup()-Teil des Programms.
void setup() {
  // Starte die serielle Kommunikation mit einer Baudrate von 9600.
  Serial.begin(9600);
}

// Die loop()-Funktion wird kontinuierlich ausgeführt, solange das Arduino läuft.
void loop() {
  // Überprüfe, ob Daten auf der seriellen Schnittstelle verfügbar sind.
  if (Serial.available() > 0) {
    // Lese die eingehende Nachricht bis zum Zeilenende ('\n').
    String message = Serial.readStringUntil('\n');

    // Überprüfe, ob die Nachricht ein Testbefehl ist.
    if (message == "Test") {
      // Ignoriere den Testbefehl und kehre sofort zur loop()-Funktion zurück.
      return;
    }

    // Verarbeite die Nachricht, wenn sie nicht der Testbefehl ist.
    processMessage(message);
  }
}

// Die processMessage()-Funktion verarbeitet die empfangene Nachricht.
void processMessage(String message) {
  // Suche die Position des ersten und letzten Leerzeichens in der Nachricht.
  int firstSpaceIndex = message.indexOf(' ');
  int lastSpaceIndex = message.lastIndexOf(' ');

  // Überprüfe, ob das Format der Nachricht gültig ist.
  if (firstSpaceIndex == -1 || lastSpaceIndex == -1 || firstSpaceIndex == lastSpaceIndex) {
    // Sende eine Fehlermeldung zurück, wenn das Format ungültig ist.
    Serial.println("Error: Ungültiges Format");
    return;
  }

  // Extrahiere die Zahlen und den Operator aus der Nachricht.
  String num1Str = message.substring(0, firstSpaceIndex);
  String operatorStr = message.substring(firstSpaceIndex + 1, lastSpaceIndex);
  String num2Str = message.substring(lastSpaceIndex + 1);

  // Konvertiere die Zahlenstrings in Fließkommazahlen.
  float num1 = num1Str.toFloat();
  float num2 = num2Str.toFloat();
  float result;

  // Führe die entsprechende mathematische Operation aus.
  if (operatorStr == "+") {
    result = num1 + num2;
  } else if (operatorStr == "-") {
    result = num1 - num2;
  } else if (operatorStr == "*") {
    result = num1 * num2;
  } else if (operatorStr == "/") {
    // Überprüfe, ob eine Division durch Null versucht wird.
    if (num2 == 0) {
      Serial.println("Error: Division durch Null");
      return;
    }
    result = num1 / num2;
  } else {
    // Sende eine Fehlermeldung zurück, wenn der Operator unbekannt ist.
    Serial.println("Error: Unbekannter Operator");
    return;
  }

  // Sende das Ergebnis der Operation zurück über die serielle Schnittstelle.
  Serial.println(result);
}
```

1. setup() Funktion

- Initialisiert die serielle Kommunikation.
- Setzt die Baudrate auf 9600 Baud, um die Kommunikation zwischen dem Arduino und einem angeschlossenen Computer oder einem anderen seriellen Gerät zu starten.

2. loop() Funktion

- Überprüft kontinuierlich, ob Daten über die serielle Schnittstelle verfügbar sind, und verarbeitet eingehende Nachrichten.
- Vorgehen:
    - Überprüft, ob Daten verfügbar sind.
    - Liest eine eingehende Nachricht bis zum Zeilenumbruch (**'\n'**).
    - Ignoriert Nachrichten mit dem Inhalt "Test".
    - Ruft **processMessage()** mit der empfangenen Nachricht auf, wenn diese nicht "Test" ist.

3. processMessage() Funktion

- Verarbeitet die empfangene Nachricht, extrahiert die Operanden und den Operator, führt die angegebene Operation aus und sendet das Ergebnis zurück.
- Vorgehen:
    - Trennt die Nachricht in die beiden Operanden und den Operator.
    - Überprüft das Format der Nachricht und gibt bei Fehlern eine entsprechende Meldung zurück.
    - Wandelt die Operanden von **String** in **float** um.
    - Führt die durch den Operator spezifizierte mathematische Operation aus (Addition, Subtraktion, Multiplikation, Division).
    - Sendet das Ergebnis oder eine Fehlermeldung (bei ungültigem Operator oder Division durch Null) zurück über die serielle Schnittstelle.

Der Code ist logisch so aufgebaut, dass er in einer Endlosschleife (**loop()**) auf Eingaben wartet, diese bei Eintreffen sofort verarbeitet und das Ergebnis zurückgibt. Die Trennung der Verarbeitungslogik in **processMessage()** ermöglicht eine klare Abgrenzung der Funktionalität und macht den Code modular und leichter wartbar. Die direkte Rückmeldung von Fehlern (ungültiges Format, unbekannter Operator, Division durch Null) über die serielle Schnittstelle ermöglicht eine interaktive Nutzung und einfache Fehlerbehebung.
Diese Struktur ermöglicht es dem Arduino, als einfache Recheneinheit für mathematische Operationen zu fungieren, die über eine serielle Verbindung gesteuert wird, wodurch eine Grundlage für komplexere Anwendungen geschaffen wird.

Weiterführende Informationen

Die bereitgestellten Codes bieten eine solide Basis für die in der Aufgabe geforderten Anwendungen. Es gibt jedoch Möglichkeiten, die Anwendungen über das Mindestmaß hinaus zu erweitern. Im Folgenden finden Sie einige Beispiele, wie die Anwendungen weiterentwickelt werden können.

Verbesserungen für die PC-Anwendung:

1. **Flexibilität bei der Portauswahl:** Anstatt den Portnamen fest im Code zu verankern (z.B. "COM9"), könnte eine Dropdown-Liste in der Benutzeroberfläche integriert werden, die alle verfügbaren seriellen Ports anzeigt. Dies ermöglicht es dem Benutzer, den gewünschten Port auszuwählen, ohne den Code ändern zu müssen.

2. **Fehlerbehandlung und Benutzerfeedback:** Die Fehlerbehandlung könnte durch Bereitstellung spezifischerer Fehlermeldungen verbessert werden. Unterschiedliche Nachrichten könnten beispielsweise angezeigt werden, je nachdem, ob der Port nicht geöffnet werden kann, weil er nicht existiert oder weil er bereits in Gebrauch ist.

3. **Modularisierung des Codes:** Bestimmte Teile des Codes könnten in separate Funktionen oder Klassen ausgelagert werden, um die Wartbarkeit und Lesbarkeit zu verbessern. Funktionen zur Konfiguration des seriellen Ports und zur Verarbeitung der Daten könnten beispielsweise in separate Klassen extrahiert werden.

4. **Validierung von Benutzereingaben:** Es sollte sichergestellt werden, dass die Eingaben für die Berechnungen gültig sind, bevor sie verarbeitet werden. Obwohl grundlegende Überprüfungen bereits implementiert sind, könnten diese um zusätzliche Überprüfungen erweitert werden, um sicherzustellen, dass die Zahlenwerte innerhalb sinnvoller Grenzen liegen.

Verbesserungen für das Arduino-Programm:

1. **Optimierung der String-Verarbeitung:** Die Verwendung von String-Objekten ist zwar bequem, kann jedoch in einem ressourcenbeschränkten Umfeld wie dem Arduino zu Speicherfragmentierung führen. Es könnte erwogen werden, stattdessen Char-Arrays und C-String-Funktionen zu verwenden.

2. **Effizientere Nachrichtenverarbeitung:** Anstatt jedes Mal die gesamte Nachricht zu durchsuchen, könnte die Nachricht schrittweise verarbeitet werden, sobald neue Zeichen eintreffen. Dies könnte die Effizienz steigern und den Speicherverbrauch reduzieren.

3. **Verbesserung der Fehlerbehandlung:** Ähnlich wie bei der PC-Anwendung könnten detailliertere Fehlermeldungen bereitgestellt werden, um dem Benutzer konkrete Hinweise zu geben, was falsch gelaufen ist.

Über diese beispielhaften Punkte hinaus könnte auch das Design der Benutzeroberfläche weiterentwickelt werden.

Finalisierungsphase:

Implementierung eines einfachen Kalkulators auf einem Mikrocontroller

In der Finalisierungsphase wurde der Code weiter überarbeitet und verbessert der geänderte Code ist im Dokument auf GitHub verfügbar, wo es eingesehen und heruntergeladen werden kann. Darüber hinaus bietet die README.md-Datei auf GitHub zusätzliche essentielle Informationen zur Installation und Ausführung der Codes.

https://github.com/timoKlieIU/einfacher-Kalkulator-auf-Mikrocontroller.git

Qt-Creator Anwendung:

In der neuen Version des PC-Codes im Vergleich zur ersten wurden mehrere Änderungen und Erweiterungen vorgenommen, um die Funktionalität und Benutzerfreundlichkeit der Anwendung zu verbessern:

1. **Hintergrundbild hinzugefügt:** In der MainWindow-Klasse wurde ein Hintergrundbild für das Hauptfenster eingefügt. Dies wurde durch Laden eines Bildes mit QPixmap und das Setzen des Fensterhintergrunds mit einem entsprechenden Stil erreicht. Dies verbessert das visuelle Erscheinungsbild der Anwendung.

```
16
17      QPixmap background(":/new/prefix1/Backround.png");
18      this->setFixedSize(background.size());
19      this->setObjectName("mainWindow");
20      this->setStyleSheet("#mainWindow { "
21                          "background-image: url(:/new/prefix1/Backround.png); "
22                          "background-repeat: no-repeat; "
23                          "background-position: center; "
24                          "}"
25                          "QPushButton { "
26                          "background-color: white; "
27                          "}");
28
```

2. **Dynamische Aktualisierung der verfügbaren seriellen Ports:** Ein neuer Timer portsUpdateTimer wurde hinzugefügt, der periodisch die Liste der verfügbaren seriellen Ports aktualisiert. Dies ist besonders nützlich, um auf Änderungen in den verfügbaren Geräten zu reagieren, ohne die Anwendung neu starten zu müssen.

```
35
36      QTimer *portsUpdateTimer = new QTimer(this);
37      connect(portsUpdateTimer, &QTimer::timeout, this, &MainWindow::listAvailablePorts);
38      portsUpdateTimer->start(1000);
39
```

```
89
90  void MainWindow::listAvailablePorts() {
91      ui->portsComboBox->clear();
92      const auto infos = QSerialPortInfo::availablePorts();
93      for (const QSerialPortInfo &info : infos) {
94          ui->portsComboBox->addItem(info.portName(), info.portName());
95      }
96  }
97
```

3. **Verbesserte Fehlerbehandlung bei der Portkonfiguration:** Die Methode setupSerialPort wurde erweitert, um spezifischere Fehlermeldungen zu liefern, je nach Art des Fehlers, der beim Versuch, den Port zu öffnen, aufgetreten ist. Dies erleichtert die Fehlersuche und Benutzerinteraktion bei Verbindungsproblemen.

```
58  void MainWindow::setupSerialPort() {
59      serialPort.setBaudRate(QSerialPort::Baud9600);
60      serialPort.setDataBits(QSerialPort::Data8);
61      serialPort.setParity(QSerialPort::NoParity);
62      serialPort.setStopBits(QSerialPort::OneStop);
63      serialPort.setFlowControl(QSerialPort::NoFlowControl);
64
65      if (!serialPort.open(QIODevice::ReadWrite)) {
66          QString errorString = "Fehler: Port kann nicht geöffnet werden. \n ";
67          switch (serialPort.error()) {
68          case QSerialPort::PermissionError:
69              errorString += "Keine Berechtigung.";
70              break;
71          case QSerialPort::DeviceNotFoundError:
72              errorString += "Gerät nicht gefunden.";
73              break;
74          case QSerialPort::OpenError:
75              errorString += "Kann nicht geöffnet werden.";
76              break;
77          case QSerialPort::WriteError:
78              errorString += "Schreibfehler.";
79              break;
80
81          default:
82              errorString += "Unbekannter Fehler.";
83          }
84          ui->statuslabel->setText(errorString);
85      } else {
86          ui->statuslabel->setText("Port ist geöffnet.");
87      }
88  }
```

4. **Änderungen in der Benutzeroberfläche zur Auswahl des Ports:** Eine ComboBox (ui->portsComboBox) wurde hinzugefügt, um die verfügbaren seriellen Ports anzuzeigen und die Auswahl eines Ports zur Nutzung zu ermöglichen. Dies verbessert die Flexibilität und Benutzerfreundlichkeit, indem es dem Benutzer ermöglicht wird, dynamisch zwischen den verfügbaren Ports zu wählen, ohne Änderungen am Code vornehmen zu müssen.

```
48      connect(ui->portsComboBox, SIGNAL(currentIndexChanged(const QString &)), this, SLOT(on_portsComboBox_
49  }
```

```
99  void MainWindow::on_portsComboBox_currentIndexChanged(const QString &portName) {
100     if (serialPort.isOpen()) {
101         serialPort.close();
102     }
103     serialPort.setPortName(portName);
104     setupSerialPort();
105 }
```

5. **Erweiterte Fehlermeldungen bei der Berechnungsausführung:** Die Methode performCalculation wurde aktualisiert, um detailliertere Fehlermeldungen zu liefern, falls die Eingabewerte nicht gültig sind. Außerdem wird nun überprüft, ob beim Senden der Daten über den seriellen Port ein Schreibfehler auftritt, und eine entsprechende Fehlermeldung ausgegeben.

```
void MainWindow::performCalculation() {
    QString num1 = ui->lineEdit->text();
    QString num2 = ui->lineEdit_2->text();
    QString operation = ui->comboBox->currentText();

    bool ok1, ok2;
    double val1 = num1.toDouble(&ok1);
    double val2 = num2.toDouble(&ok2);

    if (!ok1 || !ok2) {
        QString errorMessage = "Error: Ungültige Eingabe bei ";
        if (!ok1) errorMessage += "erstem Wert";
        if (!ok1 && !ok2) errorMessage += " und \n";
        if (!ok2) errorMessage += "zweitem Wert";
        ui->inputlabel->setText(errorMessage);
        return;
    }

    QString toSend = num1 + " " + operation + " " + num2 + "\n";
    if (serialPort.isOpen() && serialPort.isWritable()) {
        if (serialPort.write(toSend.toUtf8()) == -1) {
            ui->statuslabel->setText("Fehler beim Senden: Schreibfehler.");
        } else {
            ui->statuslabel->setText("Daten erfolgreich gesendet.");
            messageLog.append("Gesendet: " + toSend);
        }
    } else {
        ui->statuslabel->setText("Kann nicht senden: keine Verbindung.");
    }
```

6. **Verbesserung der Dateispeicherung:** Die Methode onSaveLogClicked wurde angepasst, um potenzielle Fehler beim Öffnen der Datei zum Speichern zu berücksichtigen. Dies erhöht die Robustheit der Funktion zum Speichern des Nachrichtenverlaufs.

```
void MainWindow::onSaveLogClicked() {
    QString filePath = QDir::currentPath() + "/nachrichtenaustausch_log.txt";
    QFile file(filePath);
    if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QTextStream out(&file);
        for (const QString &message : messageLog) {
            out << message << "\n";
        }
        file.close();
        ui->statuslabel->setText("Nachrichtenaustausch gespeichert und wird geöffnet.");
        QDesktopServices::openUrl(QUrl::fromLocalFile(filePath));
    } else {
        ui->statuslabel->setText("Fehler beim Öffnen der Datei zum Speichern. Bitte Berechtigungen überp
    }
}
```

Zusammenfassend beinhalten die Änderungen und Erweiterungen in der zweiten Version eine deutliche Verbesserung der Benutzeroberfläche durch das Hintergrundbild, eine erhöhte Benutzerfreundlichkeit durch die dynamische Aktualisierung und Auswahl der seriellen Ports, eine verbesserte Fehlerbehandlung und Feedback für den Benutzer sowie eine robustere Implementierung der Log-Speicherfunktion.

Arduino-Anwendung:

Der Arduino-Code ist so konzipiert, dass er eine flexible und robuste Kommunikation über die serielle Schnittstelle ermöglicht, um mathematische Berechnungen basierend auf den empfangenen Befehlen durchzuführen. Es gibt mehrere Gründe, warum der Arduino-Code nicht angepasst werden muss:

1. Universalität und Einfachheit des Kommunikationsprotokolls:

Der Arduino-Code verwendet ein einfaches und effektives Protokoll für die Kommunikation über die serielle Schnittstelle. Er wartet auf Befehle in einem vorhersehbaren Format ("Zahl Operator Zahl") und führt die angeforderte Operation durch, wenn die Eingabe gültig ist. Dieses Protokoll ist unabhängig von der spezifischen Implementierung auf der Client-Seite (in diesem Fall der Qt-Anwendung) und erfordert daher keine Anpassung, solange die Client-Anwendung die Daten im erwarteten Format sendet.

2. Fehlerbehandlung und Validierung:

Der Arduino-Code enthält bereits grundlegende Fehlerbehandlungen, wie die Überprüfung auf gültige mathematische Operatoren und die Vermeidung von Divisionen durch Null. Darüber hinaus sendet er Fehlermeldungen über die serielle Schnittstelle zurück, wenn die Eingaben das falsche Format haben oder andere Probleme auftreten. Diese Art der Fehlerbehandlung ermöglicht es der Qt-Anwendung, entsprechend auf Probleme zu reagieren, ohne dass der Arduino-Code geändert werden muss.

3. Modularität und Erweiterbarkeit:

Der Arduino-Code ist modular aufgebaut und behandelt spezifische Aufgaben (wie das Lesen von Eingaben, das Verarbeiten von Befehlen und das Senden von Antworten) in getrennten Blöcken oder Funktionen. Diese Struktur erleichtert die Erweiterung oder Anpassung des Codes für zusätzliche Funktionalitäten, ohne die bestehende Logik zu stören. Wenn die Qt-Anwendung erweitert wird oder zusätzliche Funktionen benötigt, können diese in der Regel durch Anpassungen auf der Client-Seite implementiert werden, ohne dass Änderungen am Arduino-Skript erforderlich sind.

4. Konsistenz und Zuverlässigkeit:

Der bestehende Arduino-Code wurde bereits getestet und als zuverlässig für die vorgesehene Aufgabe befunden. Änderungen am Code könnten unerwartete Nebeneffekte haben und würden zusätzliche Tests erfordern. Wenn die Kommunikation zwischen der Qt-Anwendung und dem Arduino wie erwartet funktioniert, gibt es keinen Grund, ein funktionierendes System zu ändern, was das Prinzip "Wenn es nicht kaputt ist, repariere es nicht" (englisch: "If it ain't broke, don't fix it") widerspiegelt.

5. Kompatibilität mit anderen Client-Anwendungen:

Der Arduino-Code ist so gestaltet, dass er nicht nur mit der spezifischen Qt-Anwendung funktioniert, sondern auch kompatibel mit anderen Anwendungen sein kann, die Nachrichten im gleichen Format senden. Dies erhöht die Wiederverwendbarkeit des Codes und ermöglicht die Interaktion mit einer Vielzahl von Client-Anwendungen, ohne dass für jede neue Anwendung Anpassungen erforderlich sind.

Zusammenfassend lässt sich sagen, dass der Arduino-Code aufgrund seines flexiblen Kommunikationsprotokolls, seiner robusten Fehlerbehandlung, seiner Modularität und seiner bewährten Zuverlässigkeit keine Anpassung benötigt, um mit dem Qt Creator-Code zu funktionieren. Stattdessen könnten erforderliche Anpassungen oder Erweiterungen auf der Seite der Qt-Anwendung implementiert werden, um die Kommunikation und Interaktion mit dem Arduino-Code zu optimieren.