Labor Embedded Systems 2

Versuch	Vorbereitung	Durchführung	Fertigstellung	Summe
1	2 h	3 h	2 h	7 h
2	2 h	3 h	8 h	13 h
3	2 h	3 h	2 h	7 h
4	2 h	3 h	4 h	9 h
5	2 h	3 h	4 h	9 h
6	2 h	3 h	10 h	15 h
Summe	12 h	18 h	30 h	56 h

Versuch 6a:

(Prozess-)Automatisierung, Automatisierungspryramide

Raspberry Pi 4 mit RPI SENSE HAT Shield

Embedded Linux mit Qt

Version 0.8

Vorbereitung

- Verschaffen Sie sich einen Überblick über die Programmierung mit Qt und der IDE QtCreator. Insbesondere das Slot-und Signalprinzip ist wichtig. Hierzu finden Sie genügend Videos unter Youtube (https://www.youtube.com/watch?v=S6ScnZKR3nM).

Überblick (Prozess-)Automatisierung

Als Einführung in die (Prozess-)Automatisierung sollen vorab die folgenden Begriffsdefinitionen dienen. Der Bezug zum Gebiet der Embedded Systems wird noch erfolgen.

Definition 1

*Ein **Prozess** ist die Umformung, der Transport und/oder die Speicherung von Materie, Energie und/oder Informationen. (nach DIN 19 222).*

Ein Prozess (lat. Procedere = voranschreiten) beschreibt grundsätzlich einen Arbeitsablauf. Es lassen sich dabei technische (z.B. die Erhitzung von Wasser oder die Steuerung eines Kraftfahrzeuges) und nicht technische Prozesse (z.B. Denkprozesse im Gehirn oder klimatische Prozesse) unterscheiden.

Definition 2a

*Ein **technischer Prozess** ist ein Vorgang, durch den Materie, Energie oder Informationen in ihrem Zustand verändert wird. Diese Zustandsänderung kann beinhalten, dass ein Anfangszustand in einen Endzustand überführt wird.*

Definition 2b

*Ein **technischer Prozess** ist ein Prozess, dessen physikalische Größen mit technischen Mitteln erfasst und beeinflusst werden. (nach DIN 66 201)*

Physikalische Prozessgrößen werden dabei durch Sensoren erfasst, während Aktoren zur Beeinflussung des Prozesses dienen. Prozessgrößen beschreiben den Zustand eines technischen Prozesses unabhängig von der Beeinflussung durch den Prozessrechner. Prozessdaten hingegen werden zwischen dem Prozessrechner und dem technischen Prozess ausgetauscht. Sie dienen zur Zustandsbestimmung (Sensoren) sowie zur Steuerung (Aktoren) des technischen Prozesses.

Definition 3

*Unter **Prozessautomatisierung** versteht man das Fachgebiet der Automatisierung beliebiger technischer Prozesse.*

Definition 4

Unter **Prozessleittechnik** versteht man die Aufgaben des Menschen, einen technischen Prozess unter Zuhilfenahme von EDV-Anlagen zu leiten und zu bedienen. Die Visualisierung der Prozessergebnisse sowie die Prozessbeeinflussung stehen hierbei im Vordergrund.

Definition 5

*Die Informationsverarbeitung im Umfeld technischer Prozesse wird als **Prozessdatenverarbeitung** bezeichnet.*

(Prozess-)Automatisierung stellt seit Jahrzehnten ein wichtiges Teilgebiet in der Elektrotechnik dar. Komplette Studiengänge oder Vertiefungsrichtungen werden hier von Hochschulen angeboten. In der Oberrheinebene dominiert die Automatisierungstechnik weit vor der Kommunikationstechnik.

Im Umfeld der (Prozess-)Automatisierung wird heutzutage das Fünf-Ebenen-Modell zur hierarchischen Darstellung verwendet (Automatisierungspyramide).

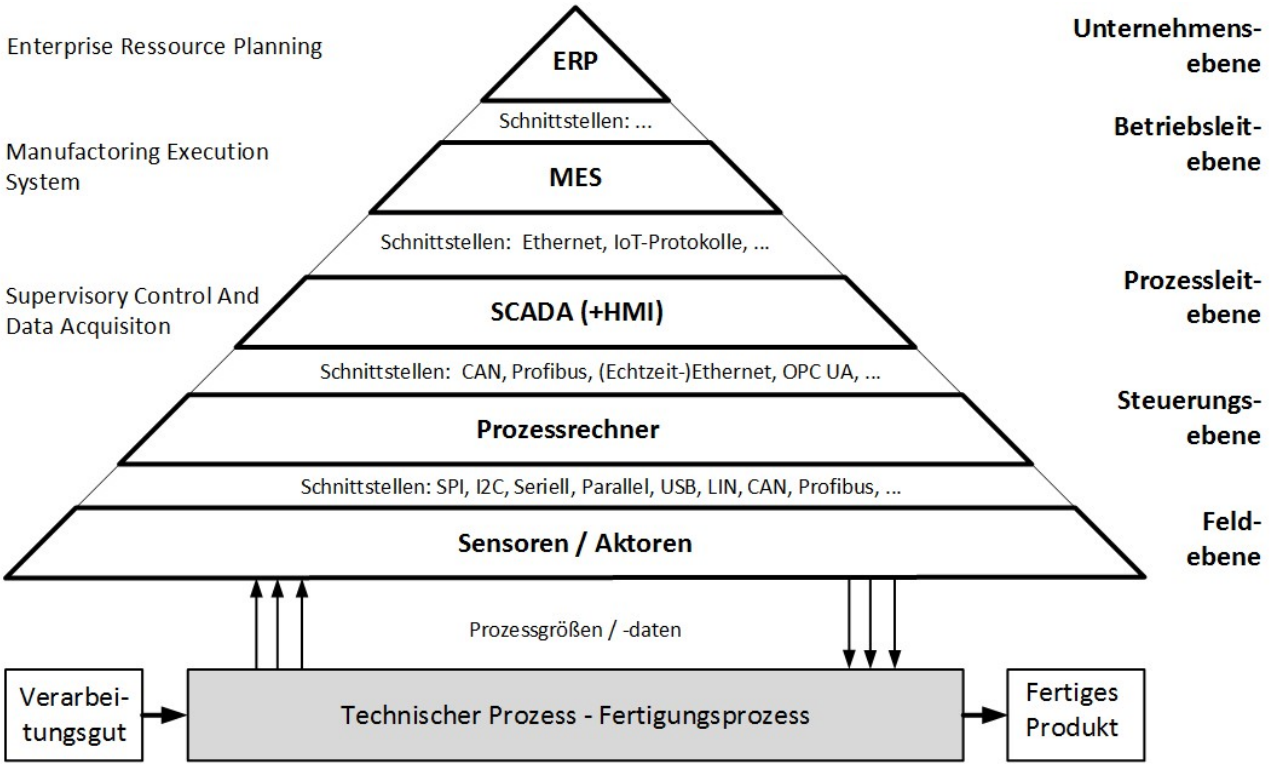

Abb. 1: Fünf-Ebenen-Modell aus der (Prozess-)Automatisierung (Automatisierungspyramide)

ERP-Systeme (z.B. von SAP oder auch von lokalen Anbietern hier aus der Region) sind dabei auf der **Unternehmensebene** angesiedelt. Auf **Betriebsleitebene** finden sich sogenannte Manufactoring Execution Systeme (MES), die die gesamte Produktion steuern und überwachen. Auf der **Prozessleitebene** finden sich SCADA-Systeme, welche die Prozessrechner steuern und überwachen und als Gateway in die Cloud oder zur MES-Welt dienen. Zunehmend gewinnt hier auch ein modernes HMI (Human Machine Interface) an Bedeutung. Auf **Steuerungsebene** finden sich Prozessrechner, die mit den Sensoren und Aktoren kommunizieren (**Feldebene**). Oft findet auch schon in den Sensoren und Aktoren eine Daten(vor-)verarbeitung mit Mikrocontrollern statt. In diesem Fall handelt es sich dann um intelligente Sensoren und Aktoren (Intelligenz = „Mikrocontroller inside“).

Eine harte **Echtzeitverarbeitung** im Milli- oder Mikrosekundenbereich ist oft auf der Feld- und der Steuerungsebene gefordert.

Zwischen den einzelnen Ebenen haben sich unterschiedliche Schnittstellen und Protokolle etabliert.

Historisch und technologisch bedingt war die Lücke zwischen Prozessleitebene und Betriebsleitebene immer sehr groß. Schon seit den 80er Jahren wurde versucht diese Lücke durchgängiger zu gestalten. Oberhalb der Prozessleitebene werden Ethernet-Protokolle gefahren, während unterhalb der Prozessleitebene spezielle Protokolle und Kommunikationstechnologien zum Einsatz kamen. Heutzutage verschwindet diese klare Trennung durch echtzeitfähige Ethernetprotokolle wie z.B. EtherCAT, Profinet und VARAN, immer mehr, da diese Ethernetprotokolle jetzt auch im „Echtzeitbereich“ unterhalb der Prozessleitebene zu finden sind.

Seit Jahren ist der Begriff Industrie 4.0 in aller Munde. In Wikipedia (Abruf 3.6.2018, https://de.wikipedia.org/wiki/Industrie_4.0) findet sich die folgende Beschreibung hierzu:

Mit der Bezeichnung „Industrie 4.0“ soll das Ziel zum Ausdruck gebracht werden, eine vierte industrielle Revolution einzuleiten. Die erste industrielle Revolution bestand in der Mechanisierung mit Wasser- und

Dampfkraft, darauf folgte die zweite industrielle Revolution: Massenfertigung mit Hilfe von Fließbändern und elektrischer Energie, daran anschließend die dritte industrielle Revolution oder digitale Revolution mit Einsatz von Elektronik und IT (v. a. die speicherprogrammierbare Steuerung) zur Automatisierung der Produktion.

Kritik an der Begriffsbildung ist letztendlich angebracht und nachvollziehbar, wenn die Historie der (Prozess-)Automatisierung betrachtet wird Der Experte Prof. Dr. Dr. Wolfgang Halang (Wolfgang A. Halang, Herwig Unger: *Industrie 4.0 und Echtzeit.* 2014, Springer Vieweg) bezeichnet diese Begriffsbildung als „sicher vermessen und unseriös". Letztendlich stellt Industrie 4.0 nur die konsequente Weiterführung der „IT-Durchgängigkeit" dar, welche seit den 80er-Jahren vorangetrieben wurde (Industrie 3.0). Die Fortschritte und die Standardisierung in der Kommunikation zwischen den Schichten haben diese Entwicklung in den letzten Jahren deutlich beschleunigt.

Ein weiterer Begriff, der derzeitig in Politik, Forschung und Wirtschaft verwendet wird, ist das Internet der Dinge (engl. IoT Internet of Things). In Wikipedia (Abruf 3.6.2018, https://de.wikipedia.org/wiki/Internet_der_Dinge) findet sich die folgende Beschreibung hierzu:

Das Internet der Dinge (IdD) (auch: „Allesnetz"; englisch Internet of Things, Kurzform: IoT) ist ein Sammelbegriff für Technologien einer globalen Infrastruktur der Informationsgesellschaften, die es ermöglicht, physische und virtuelle Gegenstände miteinander zu vernetzen und sie durch Informations- und Kommunikationstechniken zusammenarbeiten zu lassen.

Die damit einhergehende und umfangreiche Datenerfassung ergibt die folgenden oft unterschätzten Herausforderungen:

- Security (Datensicherheit)
- Auswertung und Interpretation der riesigen Datenmengen (Big Data, Data Analytics, Business Intelligence, …). Hier kommt zukünftig immer mehr Machine Learning zum Einsatz.

Während bei Industrie 4.0 eher ein hierarchischer Ansatz zu finden ist, steht das Peer-to-Peer Computing bei IoT im Vordergrund (Peer-to-Peer (P2P): Kommunikation zwischen gleichberechtigten Systemen).

Der Focus dieses Versuches liegt im Bereich der Prozessleit- und Steuerungsebene (Automatisierungspyramide, Industrie 4.0). Ebenso soll zukünftig noch eine IoT-Komponente (MQTT) eingebaut werden.

Wie sah früher (70er/80er Jahre) die Schnittstelle zwischen der Betriebsleitebene und der Prozessleitebene aus?

Ganz einfach: Papier und „Turnschuh-Netzwerk"

Das Bedienpersonal der Maschinen füllte „Maschinenzettel" aus. Diese enthielten Eckdaten wie Auftragsnummer, produzierte Teile und Zeiten. Diese Maschinenzettel wurden dann abgeholt und manuell in einem EDV-System (programmiert in Cobol) eingegeben und dann weiterverarbeitet.

Prozessleitebene, Steuerungsebene und Feldebene

Auf der **Steuerungsebene** finden sich unterschiedliche Arten von Prozessrechnern:

- Speicherprogrammierbare Steuerungen (SPS, engl. PLC Programable Logic Controller)
- Mikrocontroller (Atmel AVR, Cortex-M3 wie LPC1768)
- Digitale Signalprozessoren und Signalcontroller (TMS320C6713BPYP200 von TI, Cortex-M4)
- Einplatinen-Rechner (Raspberry Pi oder „industrietauglichere“ ähnliche Produkte)
- Personal Computer (Windows oder Linux) und IPC (Industrial Personal Computer)
- Speziallösungen (z.B. VME Vesa Modul Eurocard)

Auf der **Prozessleitebene** kommen meist Personal Computer oder Einplatinen-Rechner (wie z.B. Rasperry Pi oder „industrietauglichere“ ähnliche Produkte) zur Anwendung. Bezüglich HMI ist der Einsatz von gängigen grafischen Betriebssystemen wie Windows und Linux zu finden.

In den bisherigen Laborübungen in Embedded Systems 1 und Embedded Systems 2 wurden Automatisierungssysteme entwickelt, die Komponenten der Prozessleitebene, der Steuerungsebene und der Feldebene enthielten. Dabei lag der Schwerpunkt **nicht** auf den Kommunikationsschichten. Auf Prozessleitebene wurde bereits ein Grafikdisplay eingesetzt und auf einem zusätzlichen Applikationsboard fanden sich weitere Tasten. In den Versuchen 3 und 4, Labor Embedded Systems 1, stand die Data Acquisition im Vordergrund ohne eine Persistenz (dauerhafte Datenspeicherung) zu realisieren. Auf Steuerungsebene wurden Sensoren (simuliert) und Tasten abgefragt und Aktoren (PWM) angesteuert. Dabei kam ein Cortex-M3 als Prozessrechner zum Einsatz. Die Software für Steuerungsebene und Prozessleitebene war eine monolithische SW-Architektur (Architektur 1, Abbildung 2), was sich für die gegebenen Problemstellungen als ausreichend erwies. Auf eine Anbindung zur Betriebsleitebene wurde bisher im Rahmen der Laborübungen verzichtet.

Bei komplexeren Systemen ist dieser Ansatz nicht unbedingt zielführend (Architektur 1, Abbildung 2). Zuerst stellt sich die Frage, welche unterschiedlichen Rechner auf der Prozessleitebene und der Steuerungsebene einzusetzen sind. Selbstredend bieten hier die SPS-Hersteller (Siemens, Allen-Bradley, Beckhoff) Gesamtlösungen inklusive IDEs an. Aus verschiedenen Gründen (Platzbedarf, Kostengründen, etc.) kann es aber sinnvoller sein, eine eigene Lösung basierend auf Mikrocontrollern, (I-)PC, Einplatinen-Rechnern oder Speziallösungen zu entwickeln.

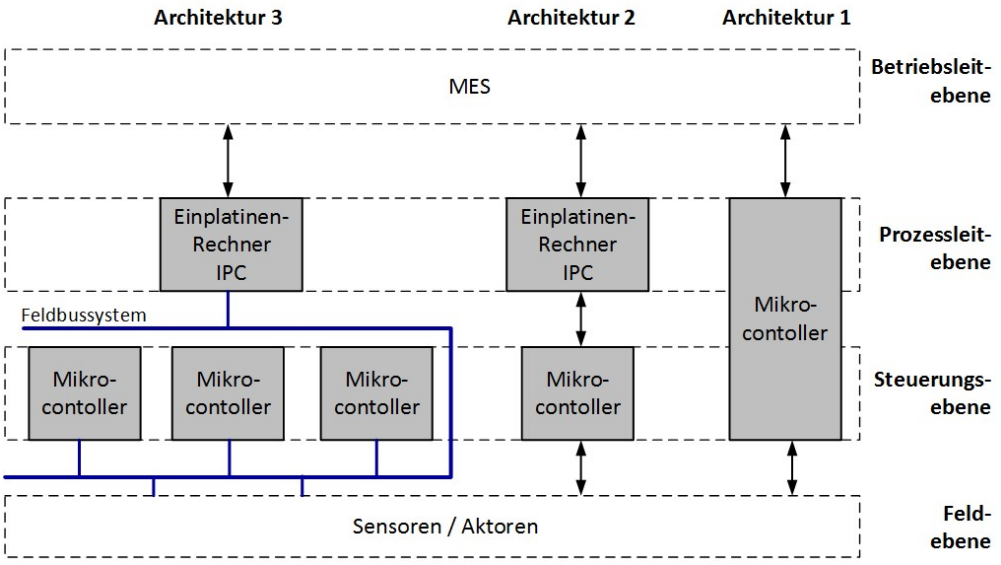

Abb. 2: *Architekturen von Automatisierungssystemen basierend auf Mikrocontrollern*

In intelligenten Sensoren und Aktoren finden sich wiederum auf Mikrocontroller basierte Embedded Systeme. Auch ist die Grenze zwischen einem Einplatinen-Rechner und einem Mikrocontroller eher fließend.

Ausgehend von der Architektur 1 zur Architektur 3 lassen sich folgende verallgemeinerte Aussagen treffen:

- Die Anbindung an die Betriebsleitebene nimmt zu.
- Falls eine Anbindung an die Betriebsleitebene besteht, dann wird diese meist durch Ethernet-basierte Protokolle realisiert. Hier setzen sich auch vermehrt standardisierte Protokolle wie OPC UA durch.
- Die Anforderungen an die GUI steigen („Wischtechnik“ und bunte Icons wie bei Smartphones). Unterschätzen Sie hier nicht den Messeeffekt: Anbieter mit moderner GUI (Eye-Catcher) lenken das Interesse der Messebesucher schnell auf Ihre Produkte.
- Persistenz (dauerhafte Speicherung von Daten) ist notwendig (Embedded Datenbanksysteme wie SQLite sind in industriellen und kommerziellen Anwendungen als gesetzt zu sehen).
- Anbindung mobiler Geräte über die Cloud - zumindest für einen Teil der Prozessgrößen und zumindest nur lesend.
- Der Einsatz von Feldbussystemen kommt immer häufiger vor und diese Feldbussysteme lösen andere, teilweise proprietäre, Schnittstellen, ab. Hier dominiert auch weiterhin CAN.

Prozessleitebene mit Raspberry Pi 4 B

Im folgenden Versuch 6a soll eine Prozessleitebene für Architektur 2 und 3 implementiert werden. Hierzu wird ein **Raspberry Pi 4 B** (Einplatinen-Rechner) verwendet (Non-Deeply Embedded). Die Steuerungsebene entfällt aus Vereinfachungsgründen in diesem Versuch. Die Anbindung an einen Mikrocontroller über ein Ethernet-Protokoll oder beispielsweise CAN wäre ebenso, z.B. mit einem PICAN 2 Shield, denkbar.

Der hier verwendete Raspberry Pi 4 B hat das Raspbian (Linuxderivat) als Betriebssystem auf der SD-Karte.

• 1,5 GHz ARM Cortex-A72 Quad-Core-CPU • 1 GB, 2 GB oder 4 GB LPDDR4 SDRAM • Gigabit LAN RJ45 (bis zu 1000 Mbit) • Bluetooth 5.0 • 2x USB 2.0 / 2x USB 3.0 • 2x microHDMI (1x 4k @60fps oder 2x 4k @30fps) • 5V/3A @ USB Typ- C Der hier im Versuch verwendete Raspberry Pi 4 B verfügt über 2 GB SDRAM. Die SD-Karte hat einen Speicher von 16GB. Auf der SD-Karte befindet sich das Image *Raspbian Buster with desktop*. .	 ***Abb. 3:*** *Raspberry Pi 4 B*

In diesem Versuch 6a wird das **RPI SENSE HAT Shield** verwendet. Dieses Shield wird/ist auf dem Raspberry Pi 4 B aufgesteckt (Sandwich). Als Einstieg soll nur eine wenig komplexe Anwendung erstellt werden, daher kommen nur das 8x8 LED-Matrix-Display zum Einsatz.

Technische Spezifikation • Gyroskop/Winkelgeschwindigkeitssensor • Beschleunigungsmesser/linearer Beschleunigungssensor • Magnetometer/magnetischer Sensor • Barometer mit einer absoluten Skala von 260–1260 hPa • Temperatursensor • relativer Feuchtigkeitsmesser • 8x8 LED-Matrix-Display • kleiner Joystick mit 5 Knöpfen Die Bibliothek zum Ansprechen des RPI SENSE HAT Shields (BSP) ist schon im Raspbian automatisch mit dabei.	 ***Abb. 4:*** *RPI SENSE HAT Shield*

Auf dem Raspberry Pi 4 B ist auf der SD-Karte das gesamte Betriebssystem und die Programme sowie Daten gespeichert, da der Raspberry Pi 4 nicht über eine Festplatte (HD) verfügt. Das gesamte Image kann z.B. unter

https://www.raspberrypi.org/downloads/raspbian/

in drei unterschiedlich großen Images heruntergeladen werden.

Grundsätzlich gibt es zwei Möglichkeiten bei der Entwicklung von Embedded Software:

- **Cross-Compilierung auf dem Host**: Hier bei wird auf einem Hostrechner (ideal wäre unter Linux) entwickelt und das generierte Binary wird dann auf das Target übertragen und dort ausgeführt. Diese Vorgehensweise ist am Anfang aufwändiger (Mehraufwand für Tool-Chain – Target-Bibliotheken müssen auf dem Host verfügbar sein und beim Liken hinzugefügt werden), allerdings ist dies die typische Vorgehensweise bei größeren Projekten. Eine Cross-Compilierung wurde bereits in den bisherigen Laborversuchen in ES1 und ES2 praktiziert.
- **Compilierung auf dem Target**: Dies ist deutlich einfacher und eignet sich für kleinere Projekte. Die Entwicklungsumgebung (IDE) muss dabei auf dem Target installiert sein. Die Beschränkungen des Targets hinsichtlich Performanz, Speicherzugriffszeiten (SD-Karte) und Bildschirmauflösung (nur ein Bildschirm?) müssen dabei akzeptiert werden.

Für den folgenden Versuch erfolgt die Compilierung auf dem Target. Als Programmiersprache/Framework kommt dabei Qt mit der Entwicklungsumgebung Qt Creator zur Anwendung. Dies ist bei einem Raspberry Pi 4 B nachträglich zu installieren. Die Installation erfolgte schon und steht auf der SD-Karte bereit.

Folgende Abbildung zeigt den grundsätzlichen Aufbau des Laborversuchs.

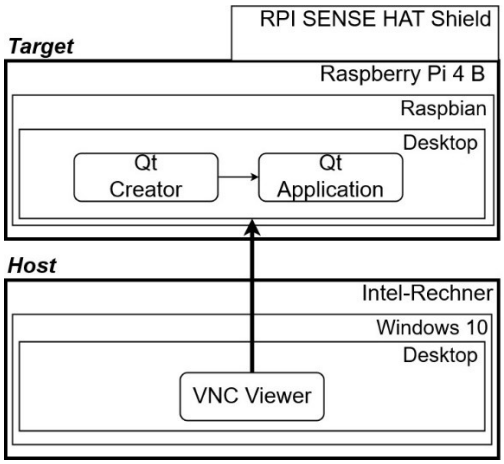

Abb. 5: *Grundsätzlicher Aufbau des Laborversuchs*

Inbetriebnahme

Die Inbetriebnahme erfolgt nach den folgenden Schritten:

- Starten Sie die VM in bwLehrpool.
- Schließen Sie das Netzteil an das Raspberry Pi 4 B Board an. Beim Hochfahren leuchtet das 8x8 LED-Matrix-Display. Die rote LED am Raspberry zeigt an, ob eine Spannungsversorgung anliegt. Die kleinere grüne LED zeigt an, ob der Bootvorgang noch läuft.
- Verbinden Sie danach Raspberry Pi 4 B mit dem Hostrechner (VM Fischerman) mit dem ausgeteilten USB-Netzwerk-Kabel.
- Starten Sie auf dem Hostrechner „Netzwerkverbindungen anzeigen“
- Es müsste jetzt „Ethernet 2“ (ASIX …) angezeigt werden. Rechtsklick auf das Symbol und Eigenschaften anklicken. In der VM müssen Sie nun ein Passwort eingeben. Dieses wird Ihnen am Labortag mitgeteilt.
- Wählen Sie in „Eigenschaften von Ethernet 2“ „Internetprotokoll, Version 4 (TCP/IPv4)“ an und betätigen Sie die Schaltfläche „Eigenschaften“.
- Geben Sie im erscheinenden Fenster als IP-Adresse 10.10.1.20 und als Subnetzmaske 255.255.255.0 ein. Schaltfläche „Okay“ drücken und mit „Schließen“ das Fenster „Eigenschaften von Ethernet 2“ schließen.
- Starten Sie nun den VNC Viewer und geben Sie als VNC Server-Adresse 10.10.1.200 ein.
- Im Authentifizierungsfenster ist als Benutzername „pi“ und als Kennwort „raspberry“ einzugeben. Die Warnung können wegklicken.

Jetzt müsste der Desktop vom Raspberry im VNC Viewer erscheinen. Sie können nun auf den Raspberry zugreifen und können jetzt nach Anwahl der Raspberry Symbols den Menüpunkt „Entwicklung“ und „Qt Creator“ anwählen.

Aufgabenstellung

Es soll eine Qt-Anwendung mit einem Fenster erstellt werden. Dieses Fenster enthält ein PushButton „Clear Matrix“ sowie 64 (8x8) PushButtons, welche die LED-Matrix des RPI SENSE HAT Shield spiegeln soll.

Die Namen der 8x8 Buttons sollten möglichst selbstsprechend sein.

Schritt 1: GUI entwerfen

Legen Sie sich ein neues Projekt unter /home/pi/QtProjects/LED88 Matrix an. Wählen Sie als Projekttyp Qt-Widget aus. Wählen Sie unter Formulardateien „mainwindow.ui“ aus und passen Sie das Fenster erst einmal an (Name des Fensters, Text, Größe (es sollte kein Resize möglich sein)).

Fügen Sie den Button „Clear Matrix“ ein. Fügen Sie 64 kleinere Buttons ohne Text in Matrixform 8x8 ein. Wählen Sie Name wie PBMatrix00, PBMatrix01, …

Compilieren und verifizieren Sie das Verhalten des Fensters.

Überprüfen nochmals die Objektnamen (objectName) Ihrer PushButtons. Dies ist für den nächsten Schritt sehr wichtig.

Schritt 2: Slots für die Buttons hinzufügen

Fügen Sie nun beginnend mit dem PushButton „Clear Matrix“ Slots für die PushButtons ein. Dies erscheinen dann als Methoden im Hauptfenster, die automatisch aufgerufen werden.

Rechter Mouseclick auf den PushButton, „Slot anzeigen…“ auswählen und dann als Signal „clicked()“ auswählen und mit der Schaltfläche „OK“ bestätigen.

Alle on_X_clicked-Signale der PushButtons der Matrixform sollten an eine private Methode

```
void MainWindow::ClickMatrixEventHandler(unsigned int uix, unsigned int uiy, QPushButton* pPB)
```

weitergeleitet werden.

Dabei stellen uix und uiy die Position in der Matrix dar (0-basiert). Ein Zeiger auf den jeweiligen Button ist mittels des folgenden Aufrufs vorher zu generieren.

```
QPushButton* pPB = qobject_cast<QPushButton*>(sender());
```

In der Methode `MainWindow::ClickMatrixEventhandler` wird ein QColorDialog geöffnet, der die RGB-Farbe zurück liefert und dann die Farbe des jeweiligen Buttons setzt. Um den QColorDialog zu verwenden, ist oben im Code der cpp-Datei noch `<QColorDialog>` zu includieren.

```
QColor color = QColorDialog::getRgba();
QString qss = QString("background-color:
rgb(%1,%2,%3)").arg(color.red()).arg(color.green()).arg(color.blue());
pPB->setStyleSheet(qss);
```

Jetzt muss nur noch im `MainWindow::on_PBClearMatrix_clicked()` Event alle anderen Buttons wieder auf die ursprüngliche Farbe zurückgesetzt werden.

Schritt 3: Implementierung der Klasse LED_Matrix

Implementieren Sie die Klasse LED_Matrix. Wichtig dabei ist, dass hier ein RGB565 Format zur Anwendung kommt. Für die drei Farben RGB werden nicht drei Bytes, sondern zwei Bytes verwendet. Daher hat das Array `pixel` auch nur den Datentyp uint16_t. Sie müssen daher im Code mit Bitoperatoren dies umrechnen.

```
class LED_Matrix
{
public:
    // Initialize the array pixel with 0x0000
    LED_Matrix();

    // Set the array pixel with given values pointed by pixelColorArray
    void SetPixelArrayRGB(uint16_t * pixelColorArray);

    // Set an Array element with QColor value
    // Attention: QColor has to be transfer to RGB565
    void SetPixelRGB(uint8_t x, uint8_t y, QColor color);

private:
    // pixel matrix in RGB565 format
    uint16_t pixel[8][8];

    // Sends the whole data in the array pixel
    void update(void);

};
```

Der Zugriff auf Geräte und Devices geschieht unter Linux dateibasiert.

```
void LED_Matrix::update()
{
   FILE * pF;

   pF = fopen("/dev/fb1", "wb");

   if (pF != nullptr)
   {
       fwrite(pixel, sizeof(uint16_t), 8*8, pF);
       fclose(pF);
   }
}
```

Schritt 4: Objekt der Klasse LED_Matrix in Mainwindow instanziieren und Methoden aufrufen.

Instanziieren Sie nun ein Objekt dieser Klasse im Mainwindow als Assoziation und rufen Sie die Methoden von LED_Matrix an der richtigen Stelle auf.

Schritt 5: Smilie

Nach dem Start der Anwendung soll ein Smilie auf dem RPI SENSE HAT Shield erscheinen.