OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG
FAKULTÄT INFORMATIK MATHEMATIK

LEGOSAR

Dokumentation des Legosar Models

Juni 2018, Regensburg

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

ACRONYM

1

BASISSOFTWARE

1.1 SCHICHTENARCHITEKTUR

Die in AUTOSAR definierte Schichtenarchitektur soll eine einfachere Portierung von Software auf unterschiedliche Hardware ermöglichen. Mussten bislang bei ungünstig konzipierten Software-Architekturen verschiedene Stellen bis hin zur Anwendungsschicht umfangreich angepasst werden, müssen mit AUTOSAR lediglich alle mikrocontroller-spezifischen Treiber im MCAL ersetzt werden. Dadurch reduziert sich der Implementierungs- und Testaufwand sowie das damit verbundene Risiko deutlich. Softwarekomponenten können so durch die hardwareunabhängige Schnittstellen leicht auf unterschiedliche Steuergeräte übertragen werden. In AUTOSAR unterscheidet man grundsätzlich die folgende Schichten:

- Anwendungsschicht
- Runtime Environment (RTE)
- Service Layer
- ECU Abstraction Layer
- Microscontroller Abstraction Layer (MCAL)

Die Umsetzung des Projekts erforderte konkrete Implementierungen sowohl im Bereich der Service Layer als auch in der ECU Hardware Abstraction.

1.1.1 *Service Layer*

Die Service Layer enthält im Allgemeinen die Betriebsystem-Funktionen und stellt verschiedene Arten von Hintergrunddiensten wie Netzwerkdienste, Speicherverwaltung und Buskommunikationsdienste für die Anwendungsschicht bereit. Sie ist die höchste Schicht der Basissoftware und ist damit essentiell für die Anwendungsschicht. Die Implementierung ist in den meisten Fällen hardwareunabhängig und damit leicht austauschbar.

1.1.1.1 *Sound Handler*

Der Sound Handler stellt der RTE spezifische Funktionen zur Audiowiedergabe zur Verfügung. Als Funktionalität implementiert der Sound Handler dabei sowohl die einfache Wiedergabe eines Tons oder einer WAV-Datei als auch die Mehrfachwiedergabe mit optionaler Pause. Umgesetzt wird dies durch die folgenden Methoden:

```
void play_single_wav(U32 freq, U32 ms, U32 vol);
void play_single_wav(U32 file, U32 length, U32 freq, U32 vol);

void play_multiple_tones(U32 freq, U32 ms, U32 vol, U32 rep, U32 pause);
void play_multiple_wavs(const CHAR *file, U32 length, S32 freq, U32 vol, U32 rep, U32
    pause);
```

1.1.1.2 *Motor Handler*

Der Motor Handler erlaubt die Ansteuerung der Motoren.

```
void motor_set_speed(S8 speed_left, S8 speed_right);

S8 motor_read_speed();
```

1.1.1.3 *Communication Handler*

Der Communication Handler ist für die Bluetooth Verbindung zwischwen zwei Geräten zuständig. Da eine Bluetooth-Verbindung eine Master-Slave Verbindung ist, muss die physische Slave-Adresse im Code definiert werden. Diese wurde mithilfe eines simplen konstangten Byte-Array gelöst. Für die RTE wird eine initialize, send, recv und terminate Funktion zur Verfügung gestellt, wobei die send und recv Funktion mit einem Ringbuffer arbeiten und somit asynchron sind. Um Threadsafety zu gewährleisten, werden in diesen beiden Funktionen für das Lesen und Schreiben des Ringbuffers alle Interrupts mithilfe der Funktion *DisableAllInterrupts* deaktiviert und bei Verlassen der Funktion mithilfe von *EnableAllInterrupts* wieder aktiviert. Es gibt 2 Tasks, einen für send und einen für recv, welche dann die Ringbuffer befüllen oder auslesesen. Diese Tasks sind in der OIL-Datei mit *SCHEDULE = NON* deklariert, damit diese nicht unterbrochen werden können und weiterhin Threadsafety ohne Implementierung von einem Mutex oder Semaphor gewährleistet ist. Damit Pakete nicht auseinander gerissen werden, muss ein einzelnes receive-Runnable im Application Layer definiert werden, welches dann die Zusammensetzung für empfangene Daten zu einem ganzen Paket übernimmt und die anderen Runnables benachrichtigt.

```
//Returns 1 on success, 0 on failure
U8 com_init(U8 is_master);
//Returns number of sent bytes, 0 on error [Async]
U32 com_send(U8 *buff, U32 len);
//Returns number of received bytes, 0 on error [Async]
U32 com_recv(U8 *buff, U32 len);
void com_terminate();
```

1.1.1.4 *Display Handler*

Der Display Handler bietet Funkionalität zum Darstellen von Textausgaben auf dem Display des Bricks an. Ganze Strings und einzelne Zeichen können entweder fortlaufend oder an einer bestimmten Stelle des Monitors angezeigt werden. Mit der Funktion *display_clear_line* können ganze Zeilen gelöscht werden. Folgende Funktionen stehen zur Verfügung:

```
void display_write_xy(int x, int y, const char *str);
void display_clear_line(int y);
void display_write(const char *str);
```

1.1.2 *ECU Hardware Abstraction*

Die ECU Hardware Abstraction Layer versteckt den konkreten Aufbau des Steuergeräts und bietet einen einheitlichen Zugriff auf alle Funktionalitäten eines Steuergeräts wie Kommunikation, Speicher oder IO - unabhängig davon, ob diese Funktionalitäten Bestandteil des Mikrocontrollers sind oder durch Peripheriekomponenten realisiert werden.

1.1.2.1 *Ultraschall Hardware Abstraction*

1.1.2.2 *ADC Hardware Abstraction*

1.1.2.3 *DIO Hardware Abstraction*

Die DIO Hardware Abstraction stellt der RTE ein Interface zur Ansteuerung der internen und externen digitalen Ein- und Ausgänge zur Verfügung.

```
#define DIO_Read_Data(DIOIndex, Port, Adresse) \
        DioIfReadFctPtr[DIOIndex](Port, Adresse)

#define DIO_Write_Data(DIOIndex, Port, Adresse, Data) \
        DioIfWriteFctPtr[DIOIndex](Port, Adresse, Data)
```

Listing 1.1: DIO Interface

Generell ist das Interface für die Ansteuerung aller digitalen IO-Ports konzipiert worden. Der Funktionen *Read* und *Write* wird ein Index übergeben. Dieser verweist in einem Array auf die dazugehörige interne oder externe Funktion.

Um die Taster des Roboters auszulesen, muss somit ein Funktionsaufruf der *Read*-Funktion mit dem Index 0 getätigt werden.

```
U8 dio_read_int(U8 port_id, U8 i2c address){
        return ecrobot_get_touch_sensor(port_id);
}
```

Listing 1.2: Aufruf der internen DIO-Read Funktion

Das Ansteuern der LEDs geschieht über den Funktionsaufruf *Write*. Hier muss der Index 1 übergeben werden, da sich die LEDs an einem externen IO-Port befinden. Da auch die ADC Hardware Abstraction mit dem I^2C Expander arbeitet, wird an dieser Stelle der Zugriff über die I^2C Hardware Abstraction erfolgen. Diese ist in Kapitel 1.1.2.4 beschrieben.

```
void dio_write_ext(U8 port_id, U8 i2c address, U8 data){
        i2c_write(port_id, i2c_address, data);
}
```

Listing 1.3: Aufruf der I²C Hardware Abstraction

1.1.2.4 *I²C Hardware Abstraction*

2

HARDWAREAUFBAU

Im Rahmen des Projektes, wurde der Roboter durch einen externen Hardwareaufbau erweitert. Die prinzipielle Schnittstelle zwischen Roboter und Steckbrett wurde unter Verwendung eines I^2C-Expanders realisiert.

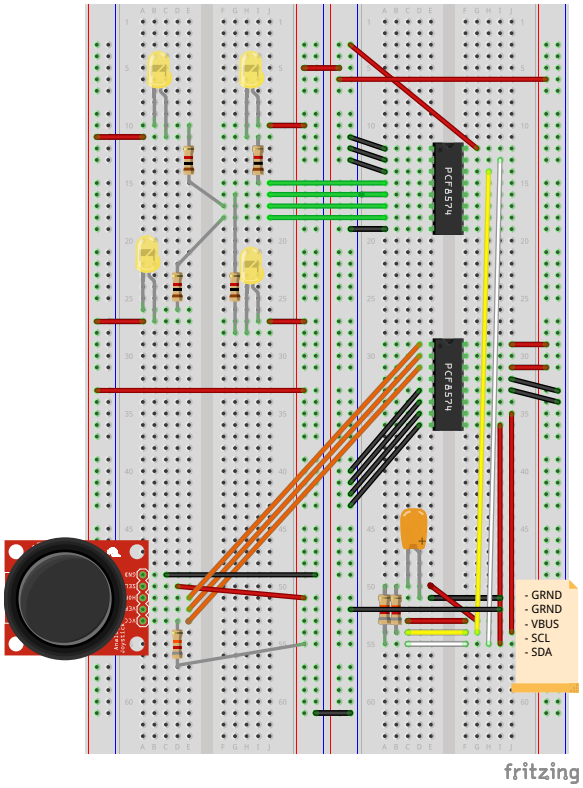


Abbildung 2.1: Hardwareaufbau auf Steckbrett

In Abbildung 2.1 ist die finale Erweiterung dargestellt. Folgende Bauteile wurden hierfür verwendet:

- 2 PCF8574 Schnittstellen-IC
- 4 LEDs
- Joystick

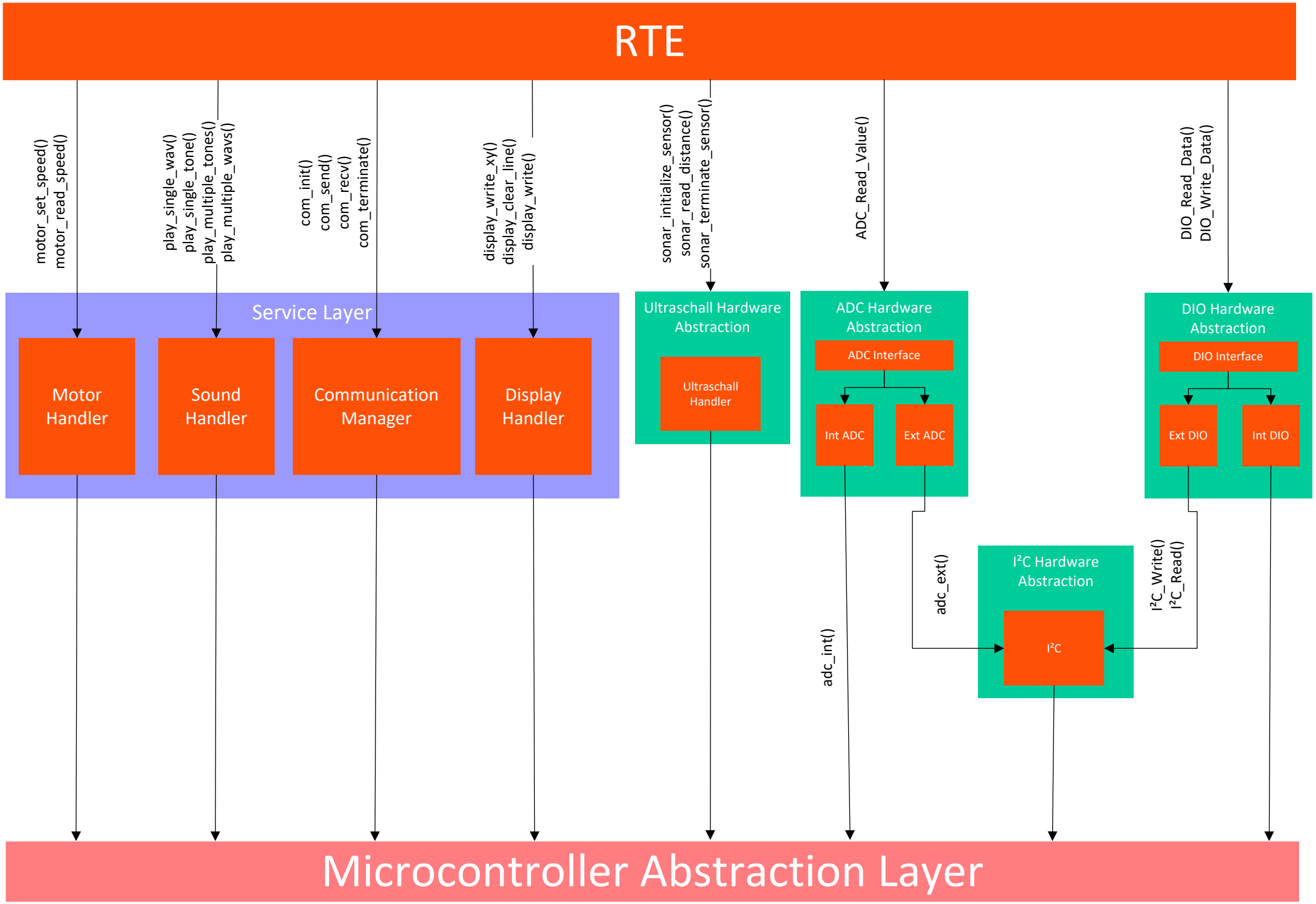
Der Roboter kann nun über ein Verbindungskabel an das Steckbrett angebunden werden. Hierfür teilen sich die zwei ICs einen I^2C-Bus. Aus Gründen der besseren Übersicht, wurden die LEDs sowie der Joystick an jeweils eigene ICs angeschlossen.

Die LEDs sind, wie in Abbildung 2.1 gezeigt, an den oberen IC angeschlossen. Es werden vier der sieben Digital-IO Pins verwendet.

Der Joystick ist, wie in Abbildung 2.1 gezeigt, an den unteren IC angeschlossen. Es werden alle der drei Analogen-IO Pins verwendet.

A

ANHANG


RTE
motor_set_speed()
motor_read_speed()
play_single_wav()
play_single_tone()
play_multiple_tones()
play_multiple_wavs()
com_init()
com_send()
com_recv()
com_terminate()
display_write_xy()
display_clear_line()
display_write()
sonar_initialize_sensor()
sonar_read_distance()
sonar_terminate_sensor()
ADC_Read_Value()
DIO_Read_Data()
DIO_Write_Data()
Service Layer
Motor Handler
Sound Handler
Communication Manager
Display Handler
Ultraschall Hardware Abstraction
Ultraschall Handler
ADC Hardware Abstraction
ADC Interface
Int ADC
Ext ADC
DIO Hardware Abstraction
DIO Interface
Ext DIO
Int DIO
adc_ext()
adc_int()
I²C Hardware Abstraction
I²C
I²C_Write()
I²C_Read()
Microcontroller Abstraction Layer

Test. Ende im Gelände

LITERATURVERZEICHNIS

[1] ARM: *AMR7TDMI Technical Reference Manual*, 14.06.2018, http://infocenter.arm.com/help/topic/com.arm.doc.ddi0210c/DDI0210B.pdf