AUTOSAR

SoSe 2015

12. Juni 2015

Beitragende:
Daniel Tatzel (DT)
Florian Laufenböck (FL)
Markus Wildgruber (MW)
Philipp Eidenschink (PE)
Tim Schmiedl (TimS)
Tobias Schwindl (TobiS)

VersionsNr	Datum	Auslöser	Beschreibung
1.0	21.04.2015	DT	Erster Entwurf
1.1	7.06.2015		Überarbeitung/Funktionsapi

1 Projekt Beschreibung

1.1 Vernetzte Ballschussanlage

- 1-2 Bricks
- Ausgabe(durch Display,LEDs etc.)
- Stop-Trigger
- Variable Aufteilung unter den Bricks: Stopp-Taste, Auslösung Taste(auch über Ultraschall), Ausgabe

1.2 Benötigte VFB-Komponenten und Schnittstellen (DT)

- Komponenten
  - Application Software Component
  - Sensor-Actuator Software Component
  - ECU Abstraction Software Component
- Schnittstellen
  - Client/Server
  - Events
  - Sender/Receiver (auch mit synchronisierung)

1.3 Namenskonventionen und Standardrückgabtyp (Alle)

Für RTE-Funktionen:	RTE_<Komponentenname>_<Funktionsname>_<Portname>_<Direction>
Für den Rest:	<Komponente>_<Funktionsname>
Standardrückgabtyp:	uint32_t ≡ std_return

1.4 Komponenten Funktionsapi

Die Funktionsprototypen für die RTE-Funktionen sind in der Codedokumentation zu finden(unter *YASA_RTEAPI.h*).

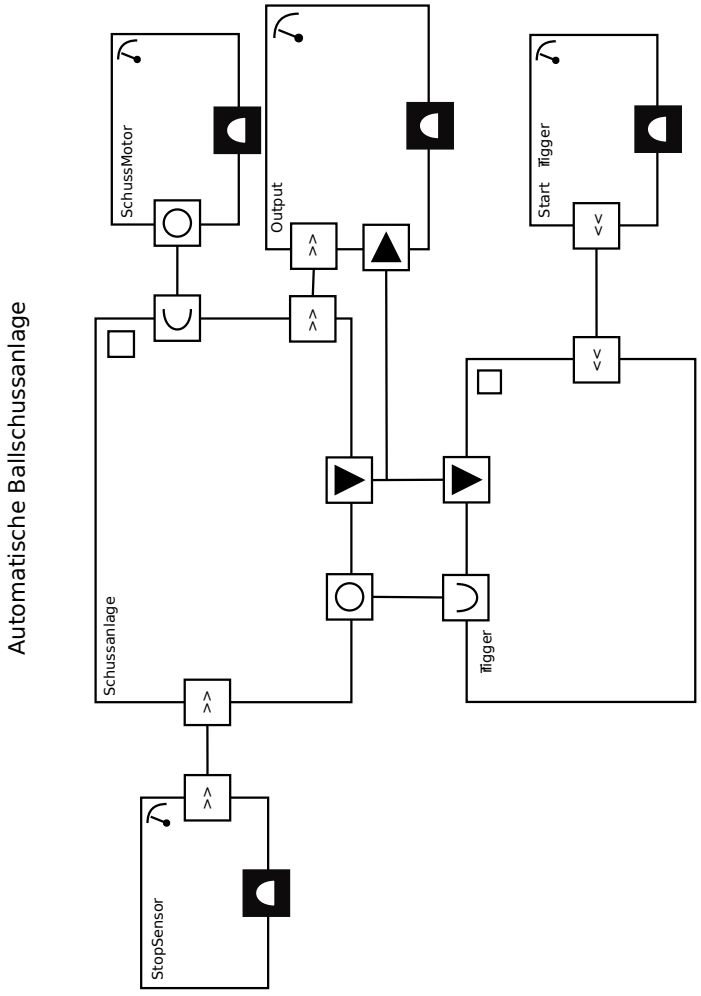


Abbildung 1.1: Komponentendiagramm der Ballschussanlage (DT)

2 Komponenten-Beschreibung

2.1 Lose Beschreibung

Schussanlage (FL)

- Besteht aus einer Task mit zwei Runnables
- erste Runnable prüft periodische die Abbruchbedinung(hier: Taster)
- zweite Runnable managt den Schussmotor
- Kein Autostart des Tasks, wird über den Trigger gestartet
- Ports siehe Komponentendiagramm

Benötigt: Task und Event

Trigger (PE)

- Ein Task
- Wird zu beginn gestartet (Autostart)
- Wartet auf Event vom Input

Benötigt: Task und Event

Output (MW)

- Autostart
- Wird durch Event von Schussanlage getriggert
- Prüft nach Event die empfangene Nachricht
- Zeigt Nachricht in Abhängigkeit der empfangen Nachricht an

Benötigt: Task und Event

SchussMotor (TimS)

- Kein Autostart
- Servertask wird durch Schussanlage (client) gestartet
- Steuert Motor zum schießen an

StopSensor (TobiS)

- Autostart
- Prüft Taster
- Setzt Event für Schussanlage

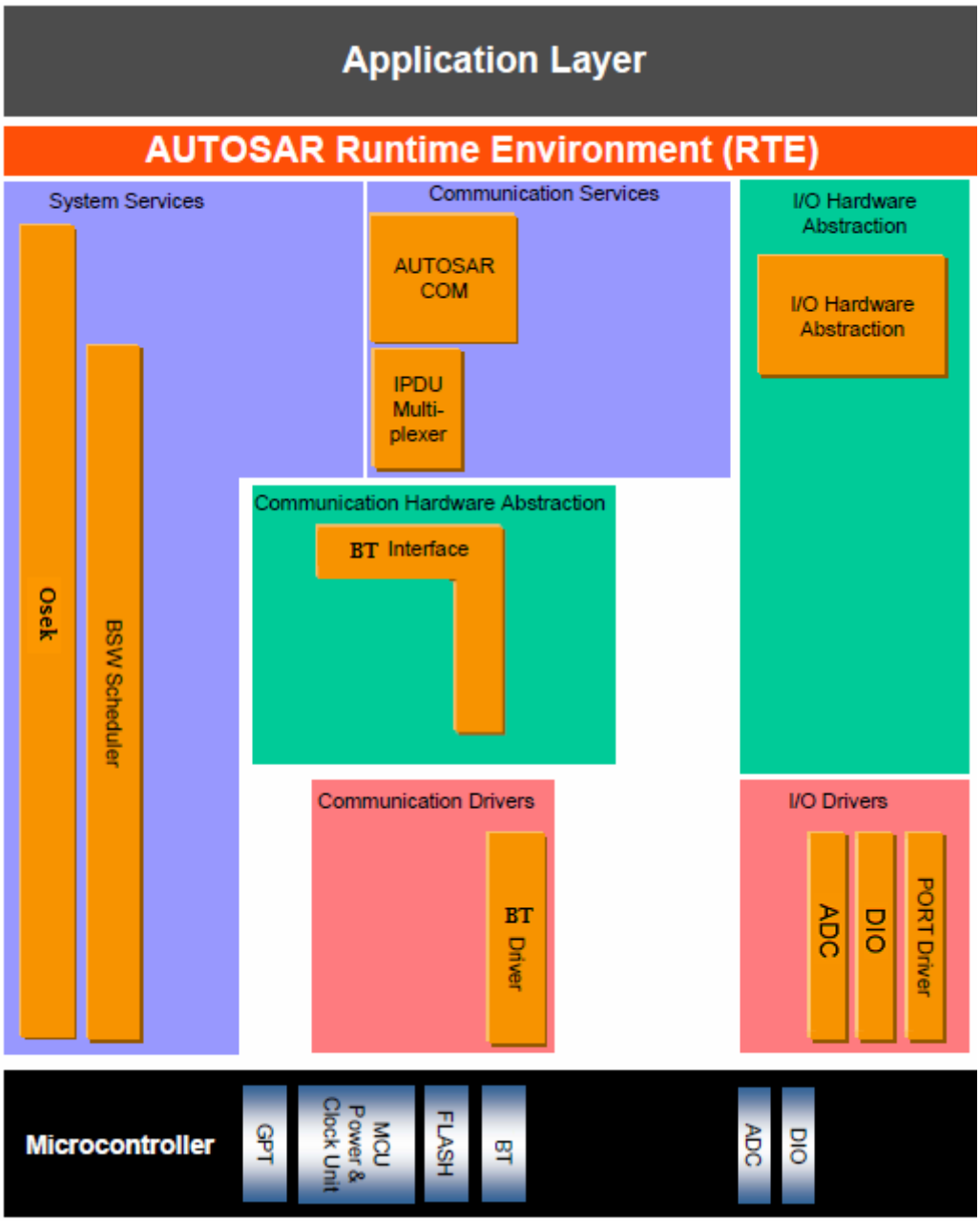
Benötigt: Task, Timer und Event

StartTrigger (TobiS)

- Task zum Erkennen von Zielen
- Autostart
- Sendet Event an Trigger
- Erkennung durch periodische Abfrage

Benötigt: Task und Timer

2.2 Architekturschicht und Funktionsapi



2.2.1 Funktionapi

1.) System Services
keine Funktionen

2.) Communication Services

- Abstraktionsebene um Nachrichten zu verschicken
- `StdReturnType TransmitMessage(char* message)`
- `StdReturnType ReceiveMessage(char* message)`

3.) I/O Hardware Abstraction

- `StdReturnType ReadDigitalInput(PortName)`
- `StdReturnType ReadAnalogInput(PortName)`
- `StdReturnType DriveMotor(Port_Name, Direction, speed, angle)`

4.) Communication Hardware Abstraction
- Für unser Projekt eigentlich unnötig, da wir nur eine Kommunikationsebene haben(theoretisch mehr durch I2C, aber hier uninteressant)
- `StdReturnType SendMessageBT(char* message )`
- `StdReturnType GetMessageBT(char* message)`

5.) Communication Drivers
- es wird nur ein Treiber für das Hardware BT gebraucht:
- `StdReturnType BT_Write(char* message)`
- `StdReturnType BT_Read(char* message)`

6.) I/O Drivers
- benötigt für den zusätzlichen I2C expander
- `StdReturnType ReadI2C(PortName)`
- `StdReturnType WriteI2C(PortName)`

2.3 Versendete Nachrichten

Jede versendete Nachricht besteht aus

- 1 Byte Header
- bis zu 127 Byte Nachricht(allerdings mit terminierender 0, das heißt nur 126 Byte Nutzdaten)

Diese Größe ist Applikationsspezifisch. Außerdem gibt es noch für jede Nachricht eine eindeutige *ID*, diese wird allerdings nicht von der Applikation, sondern von den *Communication Services* vergeben und versendet, bleibt also für die Applikation unsichtbar. (TODO:) 2 Möglichkeiten: 1 Möglichkeit die der Nachrichtenheader wird auch im ComService aufgelöst und dann kann man nur noch die eigentlichen Daten an den entsprechenden Port weiterleiten oder aber der Header wird in der Applikation aufgelöst. Ich beschreibe hier Möglichkeit 2, dann sollte das System flexibler sein, das auflösen wird dann in der SW-Komponente über einen Funktionsaufruf gemacht (End TODO)

2.3.1 Nachricht

Es gibt mehrere verschiedene Modi für Nachrichten:

1.) Nachricht: eine String-Nachricht soll übertragen:
Nachrichtenheader $\&0x80 == 1 \Rightarrow$ zeigt das die folgende Nachricht einem String entspricht:
Die restlichen Bits des Header zeigen an wie viele der folgenden Nachrichtenbytes gültig sind.
Der Nachrichtenheader $\&0x7F$ zeigt an wie viele Bytes als Nachricht gültig sind.

2.) Nachricht wird als Integer übertragen: Nachrichtenheader $\&0x80 == 0 \Rightarrow$ zeigt an das folgende Nachricht ein Integer sein muss. Dieser ist IMMER ein *unsignend integer*. Es gilt weiterhin: Alle restlichen Bits des Nachrichtenheader **müssen** 0 sein. Die folgende Nachricht muss ausserdem auch nur aus 0en bestehen.

3.) Nachricht wird als Event übertragen: Nachrichtenheader $\&0x80 == 0 \Rightarrow$ zeigt an das folgende Nachricht ein Event übertragen wird, wenn der Nachrichtenheader nicht aus lauter 0en besteht. Dabei ist die Eventnummer die Zahl nach dem Nachrichtenheader $\&0x7F$. Es gibt als maximal 127 verschiedene Events. Die eigentliche Nachricht besteht nur aus 0en.

2.3.2 Nachrichten-IDs

Jede mögliche Nachricht hat eine eindeutige ID, diese wird von dem *Com-Service* vergeben. Die Anzahl der IDs ist abhängig von der Applikation.
OS-spezifische Zugriffe können nicht über BT versendet werden und brauchen deswegen keine ID. Folgende IDs sind für unsere Applikation vergeben:

Verbindung	**Porttyp**	**ID**
SAK StopSensor - SWK Schussanlage	Event	1
SWK Schussanlage - SAK Schussmotor	SenderReceiver	2
SAK Start_Trigger - SWK Trigger	Event	4
SWK Schussanlage - SAK Output	Event	8
SWK Schussanlage - (SWK Trigger, SAK Output)	ServerClient	16
SWK Trigger - SWK Schussanlage	SenderReceiver	32

3 Konventionen/Definitionen

3.1 Tasks

3.1.1 impliziter Task

Pro Brick gibt es einen impliziten Task, der immer bei der Codegenerierung mithineingeneriert wird. Dieser Task ist für die Nachrichtenabholung, die per Bluetooth eintreffen, zuständig. Dieser Task hat nur eine Runnable. In der Runnable ist nichts zu tun, ausser auf Nachrichten zu warten und wenn eine Nachricht eintritt den Communicationservice aufzurufen/zu informieren.