LiftLibrary

3.0

Erzeugt von Doxygen 1.8.16

Kapitel 1

Datenstruktur-Verzeichnis

1.1 Datenstrukturen

Hier folgt die Aufzählung aller Datenstrukturen mit einer Kurzbeschreibung:

Kapitel 2

Datei-Verzeichnis

2.1 Auflistung der Dateien

Hier folgt die Aufzählung aller dokumentierten Dateien mit einer Kurzbeschreibung:

Kapitel 3

Datenstruktur-Dokumentation

3.1 Fsm Strukturreferenz

Repräsentiert eine Zustandsmaschine.

`#include <LiftLibrary.h>`

Datenfelder

- struct Fsm_tag ∗ Next

  nächste registrierte Zustanndsmaschine
- uint8_t RxMask

  Maske, welche angibt, welche EventSources bearbeitet werden sollen.
- StateHandler CurrentState

  Funktion, welche den aktuellen Zustand representiert.

3.1.1 Ausführliche Beschreibung

Repräsentiert eine Zustandsmaschine.

Eine oder mehrere Zustandsmaschinen können im Framework registriert werden. Beim Auftreten einer Meldung werden alle Zustandsmaschinen, welche auf diese *EventSource* registriert sind, notifiziert

Definiert in Zeile 283 der Datei LiftLibrary.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- LiftLibrary.h

3.2 LiftStatus Strukturreferenz

typedef um status information an den Terminal zu schicken

`#include <LiftLibrary.h>`

Datenfelder

- uint8_t SystemStatus
  Status information of the elevator model.
- uint8_t OpenDoors
  Bitflag to indicate which doors are open.

3.2.1 Ausführliche Beschreibung

typedef um status information an den Terminal zu schicken

Diese Information wird benötigt, um die Buttons auf dem Bildschirm zu enabled/disablen

Definiert in Zeile 252 der Datei LiftLibrary.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- LiftLibrary.h

3.3 Message Strukturreferenz

Item um Information zwischen verschiedenen Komponenten des Systems auszutauschen.

```
#include <LiftLibrary.h>
```

Datenfelder

- uint8_t Source
  Identification des Erzeugers (Sender) der Meldung (a SignalSource)
- uint8_t Id
  Id der Meldung.
- uint8_t MsgParamLow
  lower byte[0] des Meldungs-Parameters
- uint8_t MsgParamHigh
  upper byte[1] des Meldungs-Paramsters

3.3.1 Ausführliche Beschreibung

Item um Information zwischen verschiedenen Komponenten des Systems auszutauschen.

Definiert in Zeile 238 der Datei LiftLibrary.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- LiftLibrary.h

Kapitel 4

Datei-Dokumentation

4.1 LiftLibrary.h-Dateireferenz

Funktionen und Framework für die Steuerung des LiftSimulators.

```
#include <inttypes.h>
```

Datenstrukturen

- struct Message

  Item um Information zwischen verschiedenen Komponenten des Systems auszutauschen.
- struct LiftStatus

  typedef um status information an den Terminal zu schicken
- struct Fsm

  Repräsentiert eine Zustandsmaschine.

Makrodefinitionen

- #define IS_RESERVATION_BUTTON_PRESSED(button, floor) ((button)& (1<<(floor+4)))

  Abfrage, ob Reservations-Taste für Etage ∗floor gedruckt ist.
- #define IS_TARGET_BUTTON_PRESSED(button, floor) ((button) & (1<<floor))

  Abfrage, ob Zielwahl-Taste für Etage ∗floor gedruckt ist.
- #define IS_EMERGENCY_BUTTON_PRESSED(button) ((button)&EmergencyButton)

  Abfrage, ob Reservations-Taste für Etage ∗floor gedruckt ist.
- #define MAX_DOORS 4

  Anzahl der vorhanden Tueren.
- #define LIFT_MAX_POS 49

  hoechster Positionswert des Liftes
- #define POS_STEPS_PER_FLOOR 16

  Anzahl der Position-Schritte pro Etage.

Typdefinitionen

- typedef void(∗ StateHandler) (Message ∗msg)

  Prototyp einer Zustandsfunktion.
- typedef void(∗ PositionChangeSignal) (uint8_t currentPosition, uint8_t targetPosition)

  Callback-Funktion, welche über einen Positionswechsel der Kabine informiert.
- typedef void(∗ TestHandlerCallback) (uint8_t ∗payload, uint8_t len)

  EIne Funktion mit dieser Signatur kann registriert werden, um ∗Test-Meldungen zu verarbeiten.

Aufzählungen

- enum Boolean { false = 0, true = 1 }

  enumerate für die explizite Festlegung von false und true
- enum AvrPacketType {
  PacketType_Undefined = 0, **PacketType_TraceMessage** = 1, **PacketType_LiftSimulatorButton** = 2, **PacketType_TestCommand** = 3,
  PacketType_LiftStatus = 4 }

  Definition des Protokolles zwischen PC und uP.
- enum FloorType { **Floor0** = 0, **Floor1** = 1, **Floor2** = 2, **Floor3** = 3 }

  Beschreibt die eine Etage im System.
- enum ButtonType {
  LiftButton_F0 = 0x01, **LiftButton_F1** = 0x02, **LiftButton_F2** = 0x04, **LiftButton_F3** = 0x08,
  FloorButton_F0 = 0x10, **FloorButton_F1** = 0x20, **FloorButton_F2** = 0x40, **FloorButton_F3** = 0x80,
  EmergencyButton = 0x100 }

  Type mit Bitmaske für die verschiedenen Buttons.
- enum DoorPosType {
  Door100 = 0xF0, **Door75** = 0x70, **Door50** = 0x30, **Door25** = 0x10,
  Door00 = 0x00 }

  Bitmaske für den Zustand der Tür
- enum DoorStateType {
  DoorMooving = 0x01, DoorOpen = 0x10, DoorOpening = 0x11, DoorClosed = 0x20,
  DoorClosing = 0x21 }

  Beschreibt den Zustand der Tür.
- enum DisplayStateType { **On**, **Off** }

  Beschreibt den Zustand der Stockwerksanzeige.
- enum ButtonStateType { **Released**, **Pressed** }

  Beschreibt den Zustand eines Buttons, welches das Event ausgelöst hat.
- enum LiftStateType {
  LiftStateNone = 0, **LiftStateMoves** = 0x10, **LiftStateError** = 0x20, **LiftStateOverload** = 0x40,
  LiftStateUpperStop = 0x80 }

  Beschreibt den aktuellen Zustand der Fahrgastzelle.
- enum DirectionType { **Down** = -1, **DirectionNone** = 0, **Up** = 1 }

  Spezifizert die Fahrtrichtung des Liftes.
- enum SpeedType { **Slow** = 8, **Medium** = 6, **Fast** = 4, **VeryFast** = 2 }

  Spezifiziert die Geschwindigkeit des Lifts.
- enum SignalSourceType {
  SignalSourceEnvironment = 0x01, **SignalSourceElevator** = 0x02, **SignalSourceEtageButton** = 0x04,
  SignalSourceLiftButton = 0x08,
  SignalSourceDoor = 0x10, **SignalSourceApp** = 0x20 }

  Beschreibt die möglichen Ereignisquellen des Systems.
- enum WellKnownMessageIds {
  SystemMessage = 0xC0, **LiftStarted** = 0xC1, **LiftCalibrated** = 0xC2, **LiftDoorEvent** = 0xC5,
  ElevatorAtFloor = 0xC6, **ButtonEvent** = 0xC7, **TimerEvent** = 0xC8, **DoorEmergencyBreak** = 0xC9 }

  Beschreibt die Meldungs-Id's welche vom Framework generiert werden.

Funktionen

- void EnterAtomic (void)

  Markiert den Beginn einer atomaren Ausführung.
- void LeaveAtomic (void)

  Markiert das Ende einer atomaren Ausführung.
- void SetState (Fsm ∗fsm, StateHandler state)

  Funktion für das Ausführen eine Zustandübergangs
- void RegisterFsm (Fsm ∗fsm)

  Registriert eine Zustandsmaschine im Framework.
- void SendEvent (uint8_t source, uint8_t id, uint8_t msgLow, uint8_t msgHigh)

  Funktion zum Senden einer Meldung.
- void RegisterTestHandler (TestHandlerCallback testHandler)

  Registrierung einer TestHandler-Funktion, welche aufgerufen wird, um einen Test-Befehl vom PC zu bearbeiten.
- void InitializePorts (void)

  Initialisierug der notwendigen Ports für das Framework
- void InitializeStart (void)

  Initialisierung der LiftLibrary.
- void Usart_Init (void)

  Initialisierung der UART Schnittstelle.
- void Usart_PutChar (char ch)

  Schreibens eines Zeichens auf die serielle Schnittstelle.
- void CalibrateElevatorPosition (PositionChangeSignal notify)

  Kalibrieren der Fahrgastzelle auf die Position: Etage0.
- ButtonStateType **ReadKeyEvent** (ButtonType button)
- DoorStateType **ReadDoorState** (FloorType floor)
- void SetDoorState (DoorStateType state, FloorType floor)

  Setzen des Tuerenzustandes pro Etage; Public-Function.
- void SetElevatorSpeed (SpeedType speed)

  Setzen der Geschwindigkeit der Fahrgastzelle.
- void MoveElevator (uint8_t targetPos, PositionChangeSignal signal)

  Startet die Fahrt der Fahrgastzelle.
- LiftStateType ReadElevatorState ()

  Liefert den aktuellen Zustand des Lifts.
- void SetDisplay (FloorType displayValue)

  Setzen der numerischen Etagenanzeige im Lift.
- void SetIndicatorFloorState (FloorType floor)

  Setzen der Ruftastenanzeige pro Etage.
- void SetIndicatorElevatorState (FloorType floor)

  Setzen der Etagenauswahlanzeige im Lift.
- void ClrIndicatorFloorState (FloorType floor)

  Loeschen der Ruftastenanzeige pro Etage.
- void ClrIndicatorElevatorState (FloorType floor)

  Loeschen der Etagenauswahlanzeige im Lift.
- uint8_t StartTimer (uint16_t ms)

  Starten eines Timers, welcher nach einer bestimmten Zeit ein Event auslöst.
- void StopTimer (uint8_t timerId)

  Verferfen eines laufenden Timers.

Variablen

- Boolean EnableStatusUpdate

  Dieses Flag enabled/disabled eine Periodischen Status-Meldung and den Terminal.

4.1.1 Ausführliche Beschreibung

Funktionen und Framework für die Steuerung des LiftSimulators.

Autor

Werner Odermatt

Rolf Laich

Im Unterricht der Module M121 sowie M242 wird ein Liftsimulator verwendet, welcher auf dem ATMEL chipset AT↩ MEGA32 basiert. Dieser Simulator bildet einen Lift mit vier Etagen ab. Die Library liefert die folgenden Funktionen:

- Eine Infrastruktur für die Implementierung und Registrierung von Zustandsmaschinen
- Eine Infrastruktur für das Versenden von Messages
- Einbindung der seriellen Schnittstelle für Debugging, Test und Remote User Interface, welches auf dem PC implementiert ist.
- Hardware-Abstraktionen für die Liftsimulation

Datum

25.05.2015 Erste Implemenierung W.Odermatt

07.09.2016 Ueberarbeitung der Software, Code-Richtlinien C#

07.25.2019 Verwendung der UART für

- Debugging und test
- User-Interaktionen;
- Lift-Buttons/Etage-Buttons (PortD ist für UART gebraucht)

10.10.2019 Einführung eines einfachen State-Event frameworks

- Lift und Türen bewegen läuft auf Timer-Interrupt
- INT1 Trigger für Türstopp/Emergency
- Funktionen für Atomare Code Abschnitte
- Kapselung von avr/IO.h

22.10.2019 Formatierung der Kommentare und Dokumenation mit Doxygen

19.11.2019 Regelmässige Status Updates für Terminal-Applikation

Version

3.0

4.1.2 Dokumentation der benutzerdefinierten Typen

4.1.2.1 PositionChangeSignal

```
typedef void(* PositionChangeSignal) (uint8_t currentPosition, uint8_t targetPosition)
```

Callback-Funktion, welche über einen Positionswechsel der Kabine informiert.

Parameter

currentPosition	enthält die aktuelle Position der Fahrgastzelle
targetPosition	enthält die Zielposition der Fahrgastzelle

Dieser Callback informiert den Prozess in definierten Zeitabschnitten über die aktuelle Position. Damit kann z.B. die Positionsanzeige gesteuert werden. Die Funktion wird im Interrutp-Kontext des Timers aufgerufen. z.Z. gibt es keine Zeitüberwachung für ISR's.

Definiert in Zeile 275 der Datei LiftLibrary.h.

4.1.2.2 TestHandlerCallback

```
typedef void(* TestHandlerCallback) (uint8_t *payload, uint8_t len)
```

EIne Funktion mit dieser Signatur kann registriert werden, um ∗Test-Meldungen zu verarbeiten.

Definiert in Zeile 294 der Datei LiftLibrary.h.

4.1.3 Dokumentation der Aufzählungstypen

4.1.3.1 AvrPacketType

```
enum AvrPacketType
```

Definition des Protokolles zwischen PC und uP.

Die Datenpackete, welche zwischen dem uP und dem PC ausgetauscht werden haben einen header. Dieser Header besteht aus einem 'PacketType' und einer Länge. Dies ist notwendig um die verschiedenen Inhalte auseinander zu halten.

Definiert in Zeile 59 der Datei LiftLibrary.h.

4.1.3.2 Boolean

```
enum Boolean
```

enumerate für die explizite Festlegung von false und true

In neueren Versionen des C-Standards sind true und false bereits definiert.

Aufzählungswerte

false	0 == false
true	1 == true

Definiert in Zeile 45 der Datei LiftLibrary.h.

4.1.3.3 DoorStateType

`enum DoorStateType`

Beschreibt den Zustand der Tür.

Aufzählungswerte

DoorMooving	Tür öffnet oder schliesst.
DoorOpen	Tür ist nicht geschlossen.
DoorOpening	Tür bewegt sich und ist nicht gesclossen.
DoorClosed	Tür ist nicht offen.
DoorClosing	Tür bewegt sich und ist nicht offen.

Definiert in Zeile 129 der Datei LiftLibrary.h.

4.1.3.4 FloorType

`enum FloorType`

Beschreibt die eine Etage im System.

Der FloorType wird sowohl für die Anzeige als auch für die bestimmung der Position des Liftes verwendet.

Definiert in Zeile 74 der Datei LiftLibrary.h.

4.1.3.5 SignalSourceType

`enum SignalSourceType`

Beschreibt die möglichen Ereignisquellen des Systems.

Jede ereignisquelle wird durch ein separates Bit modelliert. Dadurch ist es möglich, sich auf einfache Art und Weise für eine oder mehrere Signalquellen zu registrieren.

Definiert in Zeile 203 der Datei LiftLibrary.h.

4.1.3.6 SpeedType

```
enum SpeedType
```

Spezifiziert die Geschwindigkeit des Lifts.

Der Speedtype erlaubt es, Geschwindigkeitsrampen zu implementieren, wenn der Lift über grössere Distanzen fährt

Definiert in Zeile 189 der Datei LiftLibrary.h.

4.1.3.7 WellKnownMessageIds

```
enum WellKnownMessageIds
```

Beschreibt die Meldungs-Id's welche vom Framework generiert werden.

Das Framework gibt einige Medlungen vor. Alle *Systemmeldungen* haben eine Id welche $>$ 192 ist. Meldungen, welche in der eigentlichen Applikation definiert werden, sollen eine kleinere Id haben.

Definiert in Zeile 220 der Datei LiftLibrary.h.

4.1.4 Dokumentation der Funktionen

4.1.4.1 ClrIndicatorElevatorState()

```
void ClrIndicatorElevatorState (
            FloorType floor )
```

Loeschen der Etagenauswahlanzeige im Lift.

Parameter

gewähltes	Etage

4.1.4.2 ClrIndicatorFloorState()

```
void ClrIndicatorFloorState (
            FloorType floor )
```

Loeschen der Ruftastenanzeige pro Etage.

Parameter

bestellte	Etage

4.1.4.3 InitializeStart()

```
void InitializeStart (
            void  )
```

Initialisierung der LiftLibrary.

Das Board wird initialisiert (I/O) und das Framework wird gestartet (der MainLoop und der Message-Dispatcher) In dieser Funktion wird die Meldung LiftStarted erzeugt. Die SignalSource ist 'SignalSourceEnvironment'

4.1.4.4 MoveElevator()

```
void MoveElevator (
            uint8_t targetPos,
            PositionChangeSignal signal )
```

Startet die Fahrt der Fahrgastzelle.

Parameter

targetPos	Zielposition der Fahrgastzelle
signal	Callback über welchen die aktuelle Position mitgegeben wird

4.1.4.5 ReadElevatorState()

```
LiftStateType ReadElevatorState ( )
```

Liefert den aktuellen Zustand des Lifts.

4.1.4.6 RegisterFsm()

```
void RegisterFsm (
            Fsm * fsm )
```

Registriert eine Zustandsmaschine im Framework.

Parameter

fsm	Zustandsmaschine, welche registriert werden soll

4.1.4.7 RegisterTestHandler()

```
void RegisterTestHandler (
            TestHandlerCallback testHandler )
```

Registrierung einer TestHandler-Funktion, welche aufgerufen wird, um einen Test-Befehl vom PC zu bearbeiten.

Parameter

testHandler	Befehls-Interpreter

4.1.4.8 SendEvent()

```
void SendEvent (
            uint8_t source,
            uint8_t id,
            uint8_t msgLow,
            uint8_t msgHigh )
```

Funktion zum Senden einer Meldung.

Parameter

source	Id des Senders der Meldung
id	Id der Meldung
msgLow	low byte des Meldungs-Parameters
msgHigh	upper byte des Meldungs-Parameters

4.1.4.9 SetDisplay()

```
void SetDisplay (
            FloorType displayValue )
```

Setzen der numerischen Etagenanzeige im Lift.

Die numerische Etagenanzeige ist die (7-Segment-Anzeige) auf dem Simulationsboard

4.1.4.10 SetIndicatorElevatorState()

```
void SetIndicatorElevatorState (
            FloorType floor )
```

Setzen der Etagenauswahlanzeige im Lift.

Wird benötigt, um anzuzeigen, wohin jemand fahren will!

4.1.4.11 SetIndicatorFloorState()

```
void SetIndicatorFloorState (
            FloorType floor )
```

Setzen der Ruftastenanzeige pro Etage.

Wird benötigt, um anzuzeigen, dass jemand auf einer bestimmten Etage am Warten ist!

4.1.4.12 SetState()

```
void SetState (
            Fsm * fsm,
            StateHandler state )
```

Funktion für das Ausführen eine *Zustandübergangs*

Parameter

fsm	Zuststandsmaschine, welche den Zustandsübergang ausführen soll
state	Funktion, welche den neuen Zustand implementiert

4.1.4.13 StartTimer()

```
uint8_t StartTimer (
            uint16_t ms )
```

Starten eines Timers, welcher nach einer bestimmten Zeit ein Event auslöst.

Parameter

ms	nrOfMilliseconds bis der Timer ausgelöst wird.

Rückgabe

die Id des Timer 0xFF wenn der Timer nicht aktiviert werden konnte;

Das Framework erlaubt es bis zu acht timer gleichzeitig aktiviert zu haben. Ein Timer ist wird immer nur einmal aufgerufen. Periodische Timer müssen vom Klienten wieder gestartet werden.

4.1.4.14 StopTimer()

```
void StopTimer (
            uint8_t timerId )
```

Verferfen eines laufenden Timers.

Wenn ein gestarteter Timer nicht mehr benötigt wird, kann er über diese Funktion wieder abgestellt werden.

4.1.4.15 Usart_Init()

```
void Usart_Init (
            void  )
```

Initialisierung der UART Schnittstelle.

Die Serielle Schnittstelle wird verwendet um mit dem PC zu kommunizieren. Die Schnittstelle wird auf 38400 bauds initialisiert mit 8 Daten-Bits, einem Stop-Bit, ohne Parity-Bit RX ist auf Port D0 gemapped, TX auf Port D1 Das entspricht den Pins 1 und 2 auf dem Simulationsboard

4.1.4.16 Usart_PutChar()

```
void Usart_PutChar (
            char ch )
```

Schreibens eines Zeichens auf die serielle Schnittstelle.

Parameter

ch	Zeichen, welches ausgegeben werden soll.

Index