Informationen zur Steuerung der Murmelbahn

Inhalt

- Aufgliederung der Quelltextdateien
- Regulär aufgerufene Funktionen
- Weitere Funktionskonzepte
- FSM

Aufgliederung der Quelltextdateien

Main

Beinhaltet die Funktion main().

config.h

Hier befinden sich Konfigurationsoptionen: Demo-Modus, Timer-Einstellungen zum Demo-Modus der Software, zu Timer-Verzögerungen und zu der einstellung, ob ein angeschlossenes Murmelbahnboard nur simuliert werden soll.

Hier kann auch der Simulationsmodus aktiviert werden. Wenn das Relax Kit nicht an die Murmelbahn angeschlossen ist, kann das Sensoren-Verhalten der Murmelbahn auch simuliert werden. *Wenn das Relax-Kit an die Murmelbahn angeschlossen ist, sollte der Simulationsmodus (*Murmelbahn_Sim *in config.h) ausgeschaltet werden.*

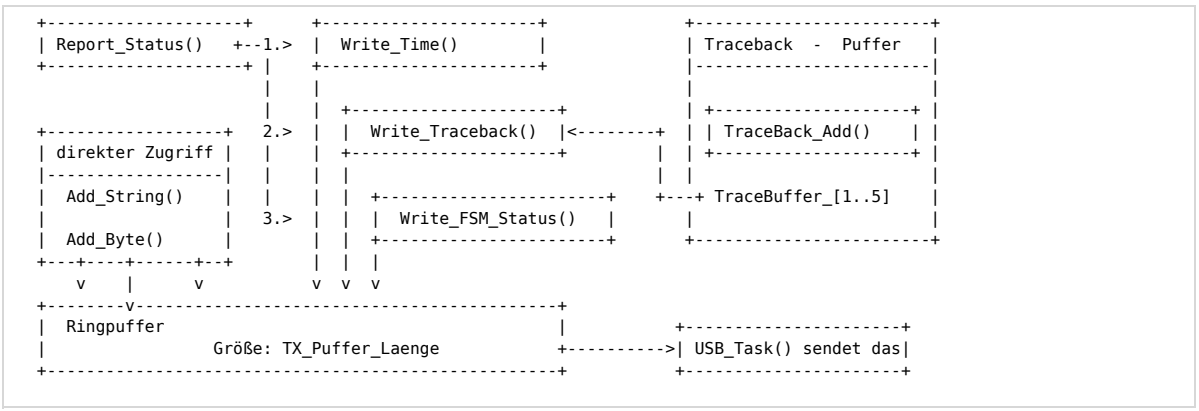
Debug

Leider konnte nicht auf die DBG002 - App von DAVE zurückgegriffen werden, weil die vorliegende Version des Relax-Boards diese nicht unterstützt :(. (Es gibt Gerüchte im Internet, dass die DBG002-App in der Version 2 des Relax Kits unterstützt wird.) Deswegen musste auf die USB-Schnittstelle zurückgegriffen werden, um eine detaillierte Debug-Ausgabe zu ermöglichen. Im Rahmen des Projekts wurden hierzu Funktionen geschrieben, die es ermöglichen, zuletzt aufgerufene Funktionen aufzulisten und dies mit Statusinformationen über die USB-Schnittstelle auszugeben.

Dazu ist es notwendig, das USB-Kabel mit der Seite des Relax-Kits zu verbinden, an der auch die Ethernetbuchse zu finden ist und in Windows die .INF Treiberdatei als USB-Gerätetreiber zu installieren. Diese Treiberdatei liegt in dem Verzeichnis, in dem DAVE die für die USB-App generierten Dateien ablegt.

Dazu die Funktionen:

- **USB_Init()** initialisiert den Ringpuffer und die USB-Schnittstelle
- **USB_Task()** muss regelmäßig aufgerufen werden.
- **Report Status(status)** gibt eine komplette Rückmeldung incl. der Systemzeit, den vorher aufgerufenen Funktionen und dem Status, der ein Zustand der FSM sein kann, oder ein anderer Wert.



Strg

Diese Quelltextdatei wurde zur Steuerung des Murmelbahnboards geschrieben. In der Strg.h sind die entsprechenden Steuercodes aufgelistet. Bei einer Änderung der Anschlüsse muss man nur diese Datei ändern. ' Zusätzlich dazu enthält sie die Funktion **Strg_Clear_ERU_Flags()**, die die ERU-Flags zurücksetzt. Das ist sehr wichtig, weil diese Bits in der ERU auf "sticky" eingestellt sind und nur bei erneutem gesetzt werden ein Interrupt auslösen. In der echten Welt bedeutet das: Ob der Mikrocontroller auf eine auslösende Lichtschranke reagiert oder nicht!

Zu einer Überprüfung, ob die Sensoren nicht fälschlicherweise ausgelöst wurden und um zu überprüfen, in welchem Zustand sie sich befinden, ist die Funktion **Strg_Sensoren_Status()**. Sollten beide Sensoren gleichzeitig ausgelöst werden, so gibt diese Funktion *Strg_Sensor_Exception* zurück. Bei Verwendung sollte das berücksichtigt werden. Dazu wird *Strg_Aufzug_irgendwo* zurückgegeben, falls keine der Schranken ausgelöst wurde.

Strg_MB_check_connection_to_board() gibt zurück, ob das Infineon-Board an das Murmelbahn-board angesteckt ist.
Strg_MB_not_connected=0x00, wenn nicht verbunden. Wenn nicht, soll es simuliert werden.

Argumente für die restlichen Funktionen:

- Strg_Aufzug(); *Strg_Aufzug_fahre_hoch* oder *Strg_Aufzug_bleibe_stehen* oder *Strg_Aufzug_fahre_runter*
- Strg_Abkuerzung(); *Strg_Abkuerzung_AN* oder *Strg_Abkuerzung_AUS* (wurde noch nicht ausgiebig getestet, Funktionieren nicht gewährleistet)
- Strg_Drehkreuz(); *Strg_Drehkreuz_AN* oder *Strg_Drehkreuz_AN*
- Strg_Murmelbahn(); *Strg_Murmelbahn_AN* oder *Strg_Murmelbahn_AUS* (regelt die ganze Spannungsversorgung für die Motoren)

FSM

Hier gibt es die für zukünftige Entwickler wichtigen Funktionen:

- **MB_Init()** initialisiert die FSM
- **MB_Reset();** setzt alles zurück und schaltet den Strom der Motoren ab. (FSM_AUS)

- **MB_CMD(Murmelbahn_Befehl);** gibt einen Befehl. abhängig davon, ob dieser ausgeführt werden kann, gibt die Funktion zurück: *new state if successful* oder *0x00 if not successful* Es filtert ungültige Eingaben einfach aus.
- **MB_Get_FSM_State();** liest den aktuellen Status der Murmelbahn FSM aus. (im Rückgabewert)
- **MB_Get_Abk_State();** liest den FSM Status der Abkürzung aus

Wobei sich die Zustands-Konstanten der FSMs in der FSM.h befinden. Die Funktion

- **MB_Set_FSM_State()** beinhaltet die FSM und regelt Übergange / Aktionen zwischen Zuständen.

... und dann gibt es noch Zeitschalter, die Funktionen der FSM regeln. Diese funktionierten über einen Countdown, der alle 100 ms zählt. MB_Timer_Event() startet dementsprechend eine Aktion, wenn der Zähler 1 erreicht. Ablaufzeiten sind in der config.h konfigurierbar.

- **MB_ResetTimer();** Setzt alle Timer auf 0.
- **MB_Abk_SetTimer();** Setzt den Timer auf die Zeit, die in der FSM Abkürzung (siehe oben) festgelegt wurde.
- **MB_Watchdog_SetCounter();** Wenn der Watchdog-Counter abläuft, so schaltet sich die FSM Murmelbahn aus. Dieser wird jedes mal zurückgesetzt, wenn z.B. MB_Set_FSM_State() eine Aktion ausführt.
- **MB_FSM_SetTimer(HowLong);** Kommt die FSM in einen Zustand, der nach einer bestimmten Zeit automatisch in einen anderen übergeht, wird das hier gesetzt.

Events

```
How the Demo works:
 1. -> Start the Demo with Murmelbahn_DEMO_Start
 2. The Demo Timer waits the in config.h specified amount of seconds
 3. The next action is performed
(Aktionen wie: Drehkreuz aktivieren, rauf- & runterfahren)
 4. Back to step 2, unless the number of actions specified is performed. The number of actions performed is defined in co
 5. The Murmelbahn will automatically timeout, if no new actions are performed (Watchdog-Counter)
 5. Stop the Demo anytime with Murmelbahn_DEMO_Stop.
```

- **MB_Events_Init();** initialisiert Timer und wird von MB_Init() aufgerufen.
- **Murmelbahn_next_state();** frägt den aktuellen FSM Status ab und versucht über MB_CMD() den entsprechenden nächsten Status aufzurufen. Rückgabewert, wenn's klappte ist der neue Status, sonst 0.
- **Murmelbahn_DEMO_handle();** wird alle 100 ms durch den Timer aufgerufen und bei Ablauf des Zähler *Murmelbahn_next_state()* auf; zählt bei Erfolg die verbleibenden Aktionen(-saufrufe) ab.
- **Murmelbahn_DEMO_Start();** setzt die Anzahl der durchzuführenden Aktionen. Startet Demo.
- **Murmelbahn_DEMO_Stop();**
- **Murmelbahn_SIM_SensorCD();** In der Simulation ersetzt diese Funktion die Sensoren -> ruft entsprechende Interrupt-Handler auf.

```
30      void Interrupt_Sensor_Oben();
31      void Interrupt_Sensor_Unten();
32      void Interrupt_Button_2();
33      void Polling_Button_1();
```

Regulär aufgerufene Funktionen

Es gibt mehrere Funktionen, die alle 100 ms durch Timer aufgerufen werden.

- immer: **MB_Timer_Event();**

Das hat die Funktion a) eines Watch Dog Timers für die FSM, der bei Ausführung von zurückgesetzt wird und nach Ablauf auf FSM_AUS schaltet. b) Der Timer für die FSM der Murmelbahn c) der Timer für die FSM der Abkürzung

- #ifdef Murmelbahn_DEMO: **Murmelbahn_DEMO_handle();**

Wird eine Demo gestartet, gibt es eine Anzahl von auszuführenden Aktionen. Die vorliegende Funktion führt in bestimmten Zeitabständen eine Aktion durch, indem sie Murmelbahn_next_state() aufruft. War dieser Funktionsaufruf erfolgreich => Verbleibende_Aktionen--;

- #ifdef Murmelbahn_Sim:**Murmelbahn_SIM_SensorCD ();**

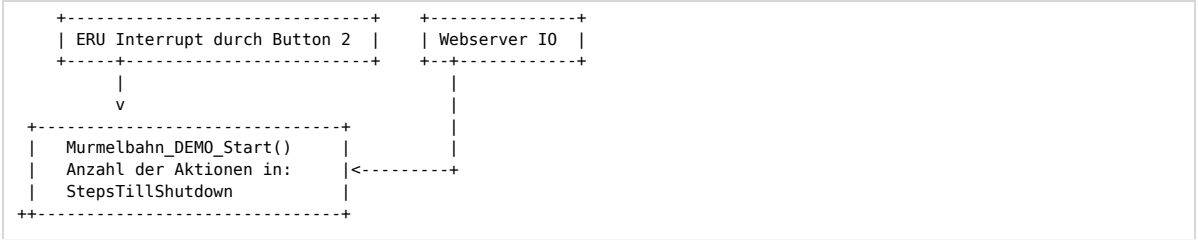
Im wesentlichen überprüft diese Funktion, ob gerade der Aufzug nach unten/oben fährt; wenn ja, dann wird ein Countdown gestartet. Bei Ablauf (SIM_Sensor_Countdown == 1) wird der Interrupt handler aufgerufen, als ob der Aufzug beim entsprechenden Sensor angekommen wäre. (Ich hätte das auch direkt in die FSM - d.h. MB_Timer_Event() - einbauen können.)

In der while-idle-loop in der main findet man noch:

- **USB_Task();** muss aufgerufen werden, um die USB-Funktion zu gewährleisten
- (veraltet) **EnlightLEDs();** aktualisiert darüber hinaus noch die LEDs (gut für Debug - sollte für eigene Bedürfnisse gut anzupassen sein) Es gibt ein paar Vorlagen zu dieser Fkt im Quelltext.

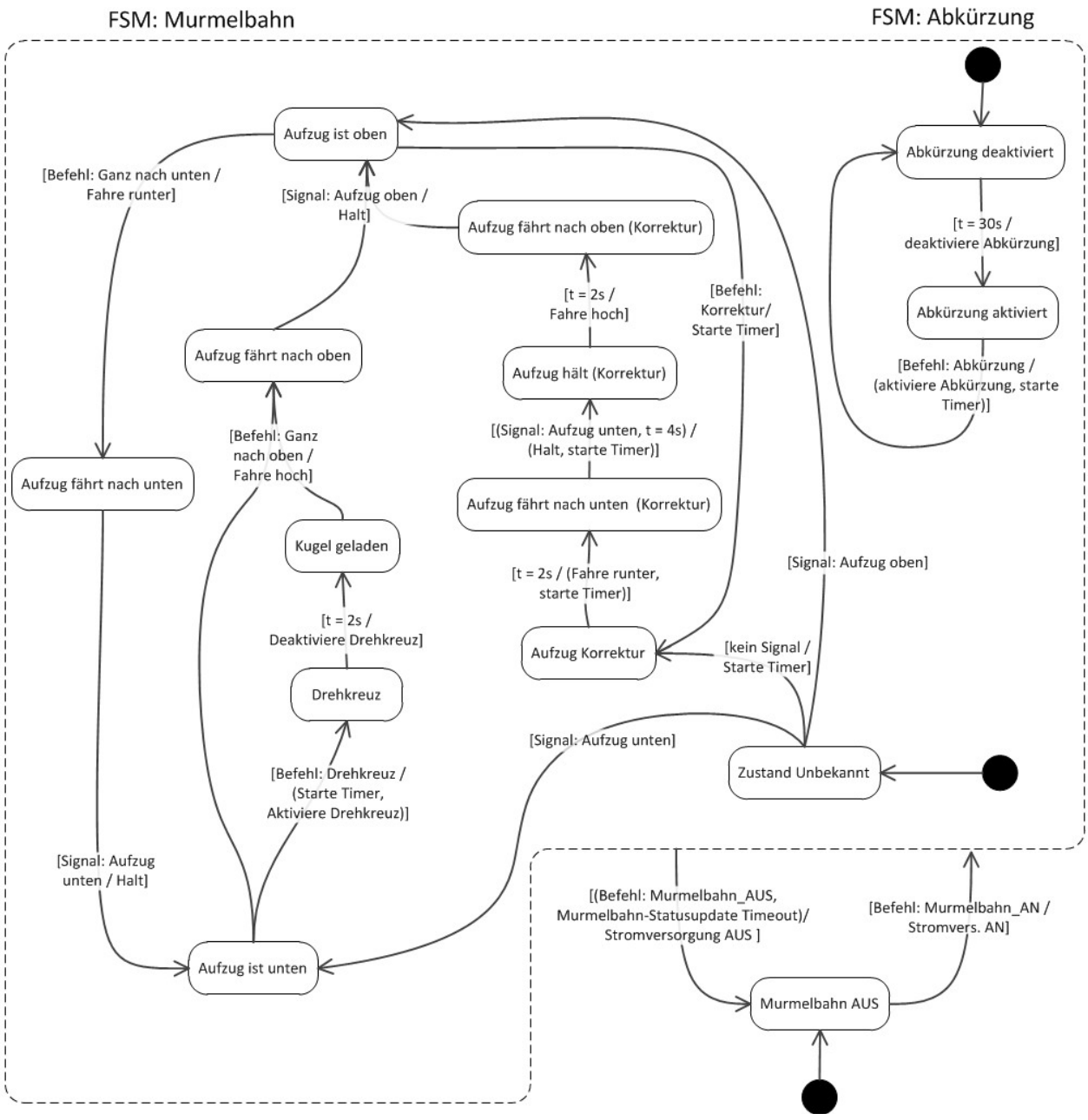
Weitere Funktionskonzepte

Murmelbahn Demo: Auf verschiedenen Ebenen werden hier nicht ausführbare bzw. ungültige Befehle ausgefiltert und Wartezeiten realisiert. Damit liess sich die Murmelbahn gut automatisieren und ist robust gegen Störungen.



```
 |
 |Alle 100 ms:  +---------------------------+
 +>------------>|  Murmelbahn_DEMO_handle()  | <+
               ++---------------------------+  |gab es einen möglichen neuen Zustandsübergang?
(nach Countdown&&| falls Aktionen übrig)        |erfolgreich nächster Zustand?
                 v                       +------+  => StepsTillShutdown--
     +-----------------------------------+-+
     |  Murmelbahn_next_state()            |<+
     +-----------+-------------------------+ |
                 |                           |Konnte der Befehl ausgeführt werden?
                 v                   +-------+
     +-------------------------------++
     | MB_CMD(ensprechende Aktion)    |<-------+
     +--------------------------------+        |
                 +                             |
                 |                             | Fand der entsprechende Statuswechsel statt?
                 v                             |
     +-----------------------------------+     |
     |  MB_Set_FSM_State()               +----+
     +-----------------------------------+
```

FSM



Attachments

- FSM_Murmelbahn.jpg (156.1 KB) - added by *no72bak* 9 months ago. “Zustandsdiagramm der Murmelbahnsteuerung, rev 4”