Karlsruher Institut für Technologie
Institut für Technik der Informationsverarbeitung

Dokumentation zum Projektpraktikum Informationstechnik

Gruppe:	Freitag 2, Gruppe 6
Gruppenmitglieder:	Nicholas-Philip Brandt, Marcel Vogel, Selina Eckel
Tutor:	Fabian Marc Lesniak
Abgabetermin:	23.01.2015
Semester:	WS2014/2015

Institutsleitung
Prof. Dr.-Ing. Dr. h. c. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

KIT - Universität des Landes Baden Württemberg und nationales Forschungszentrum in der Helmholtz-Gemeinschaft

Inhaltsverzeichnis

Kapitel 1
Einleitung

Die praktisch orientierte Aufgabe im dritten Semester war das Projektpraktikum. Die Aufgabenstellen lautetet, die Hardwaresteuerung eines TivSeg, ähnlich einem Segway, zu programmieren. Hierbei wurde man in zufälligen Dreiergruppen eingeteilt.

1.1 Arbeitsumfeld

Als Arbeitsumfeld wählte man den webbasierten Hosting Dienst Github. Github ermöglicht eine unabhängige Arbeitsweise der Teammitglieder. Durch die Versionskontrolle wird sichergestellt, dass der Code stets konsistent bleibt. Durch die History ist der gesamte Verlauf des Projekts gut zu verfolgen und Backups sind jederzeit verfügbar.

1.2 Zielbestimmung

Laut Aufgabenstellung sollte bei dem Projektpraktikum gelernt werden, wie komplexe C++ Codeabschnitte geschrieben werden und wie eine hardwarenahe Programmierung realisiert. Des Weiteren sollte das Arbeiten mit einer integrierten Entwicklungsumgebung geübt werden. Das Herz des TivSegs, mit welchem die Sensorik und Aktorik gesteurt wird, besteht aus dem EI-Mikrocontroller Board, das bereits aus den Workshops der ersten beiden Semester bekannt ist. Darüber hinaus sollte auch das Testen der unabhängigen Module auf dem Board geübt werden.
Als Simulation eines Industrieprojekts gehörten die Komponenten Zeitplanung und Teamarbeit genauso zum Praktikum, wie das eigentliche Umsetzen des Quellcodes.

1.3 Einsatz

Das TivSeg soll in den Bereichen touristische Stadtführungen, Patrouillenfahrten bei Polizeibehörden sowie die Erleichterung für Lehrpersonals auf dem Weg zu Lehrveranstaltungen eingesetzt werden.

1.4 Zeitplanung

Beim ersten Treffen wurden die einzelnen Module auf die Gruppenmitglieder aufgeteilt. Marcel übernahm die Module ADC und ADC Sensor, Nicholas die Module PWM, Motor und GPIO und Selina den Timer sowie die Dokumentation. Das Testen der einzelnen Module auf dem EI-Board wurde von den jeweils Zuständigen übernommen.
Der Gesamttest wurde mit Hilfe der Simulation zusammen durchgeführt. Für die Programmierung der einzelnen Module wurden 4 Wochen eingeplant. Die Woche darauf wurden Probleme der einzelnen Gruppenmitglieder besprochen und mit Hilfe des Tutors besprochen. In dieser Woche wurden auch die Module den Einzeltests unterzogen. In Woche 6 wurde dann der erste Gesamttest gestartet und nach ein paar Anläufen funktionierte die Simulation. Abschließend wurde in Woche 7 die Dokumentation geschrieben und die einzelnen Module wurden von den jeweils Verantwortlichen nochmal den anderen Gruppenmitgliedern erläutert.

Kapitel 2
Konzeption

2.1 Regelalgorithmus

Der Regelalgorithmus wurde den Gruppen bereit gestellt.
Dieser Regelalgorithmus übernimmt die Aufgabe der Auswertung der Sensorwerte zur Berechnung der Motorgeschwindigkeit.

2.2 Benötigte Module und ihre Funktion

Die folgenden Module mussten programmiert werden:

- Timer
  Die Aufgabe des Timers besteht darin, periodisch nach einem festen Zeitintervall eine Interrruptroutine auszulösen.
  Intern zählt dieser bis zu einem bestimmten Grenzwert, bei dem der Interrupt ausgelöst wird. Das feste Zeitintervall ist nötig, da der Regelalgorithmus Messwerte zu bestimmten Zeiten erwartet.
  Als Zählmodus wurde eine Sägezahnkurve gewählt, der im Register "Wavesalemode"gesetzt wurde. Das Intervall des Timers wurde auf 10 ms gesetzt mittels variablen RC-Wert , der bei der Initialisierung berechnet wird.

- GPIO Sensor
  Der GPIO Sensor steuert einzelne Pins des Mikrocontroller Boards an oder liest deren binäre Werte aus.

- ADC
  Die ADC Klasse ist hauptsächlich dazu da, den ADC des Mikrokontrollers zu initialisieren, die Ausgangspins zu aktivieren, konkret Signale zu konvertieren und auszulesen.
  Durch setzten des ersten Bits im Status Register des ADC kann eine solche Konvertierung gestartet werden. Dies geschieht an einem Kanal, der zuvor über das Enable Register aktiviert wurde. Nach der Konvertierung kann der konvertierte Wert im Last Converted Data Register ausgelesen werden. Des weiteren kann in diesem Modul eingestellt werden, ob ein Pin durch den GPIO oder durch eine Peripheral Function kontrolliert wird.

- ADC Sensor
  Die ADCSensor Klasse bildet einen Wrapper für einen bestimmten ADC.
  Sie initialisiert einen ADC und leitet get -Aufrufe anschließend an den ADC weiter. Bei der Initialisierung werden Slope Factor und Offset des ADC gesetzt.

- PWM
  Das PWM Modul steuert die Geschwindigkeit der Motoren.
  PWM ist eine wichtige Steuerungsweise, bei der Leistung über das Ein/Aus-Verhältnis einer binären Spannung auf einem Zeitintervall geregelt wird. Je großer der Ein-Anteil, desto großer die Leistung und damit die Geschwindigkeit.
  Die PWM wird mit Hilfe eines Zählers umgesetzt. Liegt der Zähler über dem Duty Cycle so liegt hohe Spannung an, sonst null. Erreicht der Zähler die Period wird er wieder zurück gesetzt.

- Motor
  Die Motor Klasse funktioniert analog zur ADCSensor Klasse als Wrapper für eine PWM. Dabei werden Duty Cycle und Period gesetzt. Das Verhältnis Duty Cycle zu Period entspricht dem Verhältnis Ein- zu Intervallzeit.

2.3 Beschreibung der Schnittstellen

- Fußschalter: Gibt die Steuerung des TivSegs frei und schützt den Anwender vor eventuellen Schäden.
- Lenkstange: Dient zur Richtungssteuerung. Das TivSeg fährt in die Richtung, in die die Stange geneigt wird.

Kapitel 3
Realisierung

3.1 Hardware Abstraction Layer

Zu Deutsch Hardwareabstraktionsschicht oder kurz HAL ist eine Schicht des Betriebssystems, die als Einzige auf die Hardware zugreifen kann. Sie bietet abstraktere Befehlsstrukturen und vereinfacht dadurch die Programmierung.
Die restliche Software nutzt die HAL, um Befehle umzusetzen.

3.1.1 Implementierung

Die Beschreibung der einzelnen Funktionen in den jeweiligen Modulen sind im Anhang "My Project", erzeugt durch Doxygen, in dieser Dokumentation zu finden.

3.1.2 Zusammenfügen der Module

Die Stärke von Github liegt in der Versionskontrolle, die wir nutzten um die Module zusammenzufügen. Dabei wurden die unabhängig voneinander bearbeiteten Äste des Projekts zusammengefügt (gemergt) und auftretende Konflikte manuell gelöst.

Kapitel 4
Test

4.1 Test der HAL

Die beiden Module PWM und Timer konnten auf dem EI-Board getestet werden. Beim PWM wurde die Funktion getestet, indem die LEDs in unterschiedlichen Helligkeitsstufen leuchten sollten. Beim Timer wurde die erfolgreiche Initialisierung durch eine LED angezeigt und das Hochzählen durch eine blinkende LED realisiert. Bei den Tests traten hauptsächlich Schreibfehler und Unaufmerksamkeiten auf, die meist bei der Kompilierung festgestellt wurden.

4.2 Abschlusstest

Da die Simulation nur funktioniert, wenn alle Module funktionsfähig sind, bildet die Simulation gleichzeitig auch den Logiktest.
Nach einigen Versuchen im Simulator funktionierte die abschließende Version des Quellcodes in Woche sechs konform mit der Musterausgabe.

Kapitel 5
Zusammenfassung und Ausblick

5.1 Zusammenfassung

Zusammenfassend kann man sagen, dass das TivSeg nun für den täglichen Gebrauch verwendet werden kann.
Nach einer Einarbeitungszeit und guter Zusammenarbeit zwischen den Teammitglieder konnte die Aufgabenstellung gelöst werden. Der Umgang mit dem Datenblatt wurde immer vertrauter und man lernte wie sich die Hexadezimal-Adressen der einzelnen Komponenten zusammensetzen.
Das Konzept der hardwarenahen Programmierung mit Register und Bits wurde erarbeitet und vertieft.
Erfreulich ist die Praxisnähe des Projekts, da man das eigene Resultat auch selbst am entwickelten Produkt ausprobieren darf, was bestimmt mit jeder Menge Spaß verbunden ist :P.

5.2 Ausblick

Die Fortbewegung als Grundfunktion des TivSegs funktioniert nun. Für die unterschiedlichen Einsatzgebiete könnten dies weitere Erweiterungsmöglichkeiten sein:

- Touristenbranchen
  Bei Stadtführungen sind Fotos schöne Erinnerungen. Damit man nicht immer absteigen muss, wenn ein schönes Motiv festgehalten werden soll, könnte man beispielsweise eine GoPro(TM) (Actionkamera) anbringen und diese mit einem Fernauslöser, angebracht an den Haltegriffen, bestücken. So könnte schnell und ohne absteigen Schnappschüsse geschossen werden und kein Motiv geht mehr verloren.

- Polizei
  Für Patrouillenfahrten sollte das TivSeg auch geländetauglich sein. Hierfür müssten Reifen mit tieferen Profilen bereitgestellt werden und eine Assistent zur Unterstützung der Gewichtsausgleichung entwickelt werden.

- Unigelände
  Eines der wichtigsten Komponenten im Uni-Alltag ist der Kaffee. Da wäre es doch schade, wenn der Kaffee noch schnell getrunken werden muss, damit man noch pünktlich mit dem TivSeg in der nächsten Vorlesung erscheint. Da wäre eine Kaffeehalterung doch ideal. Mit einem Sensor könnte die Füllhöhe bestimmt werden und somit berechnet werden, in welchem Winkel maximal das TivSeg gebeugt werden kann, damit der Kaffee nicht ausgeschüttet wird.

Kapitel 6
Anhang

My Project

Generated by Doxygen 1.8.9.1

Sun Jan 18 2015 12:26:51

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

Class Documentation

2.1 ADC Class Reference

`#include <ADC.h>`

Public Member Functions

- bool **init** ()

  *init resettet den **ADC** (p. 3) zunaechst und setzt anschliessend die **ADC** (p. 3) clock und die Startup time. Danach werden die 10 bit Konvertierung und der sleep Modus eingestellt. Zusaetzlich wird der Hardware Trigger deaktiviert.*
- bool **enableInPinSelector** (unsigned long channelID, bool enabled)

  Falls enabled = true, wird hier festgelegt, dass eine Peripheral Function den Pin kontrolliert. Darueber hinaus werden die Peripheral Mux Register ueberprueft und gegebenenfalls gesetzt, oder geloescht. Falls enabled = false, kontrolliert der Gpio den Pin.
- unsigned long **getChannelValue** (unsigned long channelID, bool getAverage=false, unsigned long number↩ OfConversionsForAverage=0)

  *Hier wird der aktuelle Wert des **ADC** (p. 3) im Last Converted Data Register ausgelesen. Falls getAverage true ist, wird eine Schleife aufgerufen, die in jedem Durchlauf zunaechst eine Konvertierung startet, anschliessend wartet, bis die Konvertierung abgechlossen ist und dann die Konvertierten Werte aufsummiert. Am ende wird noch der Mittelwert gebildet.*
- void **cleanUpChannel** (unsigned char channelID)

Public Attributes

- unsigned long **ID**
- signed long **offsetValue**
- float **ADCSlopeFactor**
- bool **useADCZeroOffset**
- bool **useADCSlopeFactor**

2.1.1 Detailed Description

Die **ADC** (p. 3) Klasse ist hauptsaechlich dazu da, den **ADC** (p. 3) des u-controllers zu initialiesieren, die Ausgangspins zu aktivieren und konkret Signale zu konvertieren und konvertierte Werte aus den Registern auszulesen (Methode: getChannelValue).

Zudem kann hier eingestellt werden, ob ein ausgewaehlter Pin durch eine Peripheral Function, oder durch den GPIO kontrolliert wird (Methode: enableInPinSelector).

Definition at line 66 of file ADC.h.

The documentation for this class was generated from the following files:

- Sensor/ADC.h
- Sensor/ADC.cpp

2.2 ADCSensor Class Reference

In der **ADCSensor** (p. 4) Klasse sind primaer set und get Methoden implementiert, um fest zu legen, welcher **ADC** (p. 3) Kanal verwendet wird, ob ein slope factor, oder ein offset verwendet werden und um die entsprechenden Einstellungen aus zu lesen. Darueber hinaus wird in der getIntegerValue die **ADC** (p. 3) Methode getChannel↩ Value verwendet, um aus einem Speziellen Kanal einen Wert aus zu lesen und gegebenenfalls einen Offset zu subtrahieren. Mit den Methoden dieser Klasse werden den Variablen in der **Segway** (p. 13) Klasse ihre Werte zugewiesen.

`#include <ADCSensor.h>`

Public Member Functions

- bool **init** (**Configuration::s_ADCSensorConfig** ∗thisADCSensorConfig_, **ADC** ∗ADCController_)

  Uebergibt die Werte aus thisADCSensorConfig_ an ADCController_.
- long **getIntegerValue** (bool average=false, unsigned long numberOfValuesForAverage=0)

  *Verwendet die **ADC** (p. 3) Methode getChannelValue, um den Wert des ADCSensors auszulesen.*
- void **setZeroOffset** (bool active, signed long offset)
- bool **getZeroOffsetIsActive** ()
- signed long **getZeroOffset** ()
- float **getFloatValue** (bool average, unsigned long numberOfValuesForAverage)

  *Verwendet die **ADC** (p. 3) Methode getChannelValue, um den Wert des ADCSensors auszulesen. Gibt das Ergebnis allerdings als float aus.*
- void **setSlopeFactor** (bool active, float factor)
- bool **getSlopeFactorIsActive** (void)
- float **getSlopeFactor** (void)
- void **setChannelID** (unsigned long newChannelID)
- unsigned long **getChannelID** (void)

2.2.1 Detailed Description

In der **ADCSensor** (p. 4) Klasse sind primaer set und get Methoden implementiert, um fest zu legen, welcher **ADC** (p. 3) Kanal verwendet wird, ob ein slope factor, oder ein offset verwendet werden und um die entsprechenden Einstellungen aus zu lesen. Darueber hinaus wird in der getIntegerValue die **ADC** (p. 3) Methode getChannel↩ Value verwendet, um aus einem Speziellen Kanal einen Wert aus zu lesen und gegebenenfalls einen Offset zu subtrahieren. Mit den Methoden dieser Klasse werden den Variablen in der **Segway** (p. 13) Klasse ihre Werte zugewiesen.

Definition at line 16 of file ADCSensor.h.

The documentation for this class was generated from the following files:

- Sensor/ADCSensor.h
- Sensor/ADCSensor.cpp

2.3 Configuration Class Reference

This class contains static variables only, which hold the configuration parameters for all other classes used by the segway project.

`#include <Configuration.h>`

Classes

- struct **s_ADCSensorConfig**
- struct **s_gpioMultiplexData**
- struct **s_GPIOSensorConfig**
- struct **s_MotorConfig**
- struct **s_PWMConfig**
- struct **s_StatusLED**
- struct **s_UARTConfig**

Static Public Member Functions

- static void **init** ()

  Initializes all configuration variables.

Static Public Attributes

- static unsigned long **Oscillator_Freq** = 0
- static unsigned long **CPUCLK** = 0
- static unsigned long **PBACLK** = 0
- static unsigned long **PWMCLK** = 0
- static unsigned long **ADCCLK** = 0
- static unsigned char **Timer_Channel** = 0
- static unsigned char **Timer_Clock_Connection** = 0
- static **s_PWMConfig leftPWMConfig** = {}
- static **s_PWMConfig rightPWMConfig** = {}
- static **s_MotorConfig leftMotorConfig** = {}
- static **s_MotorConfig rightMotorConfig** = {}
- static unsigned char **Motor_enabledPinPort** = 0
- static unsigned long **Motor_enabledPinPin** = 0
- static bool **Motor_enabledPinEnabledValue** = 0
- static **s_GPIOSensorConfig footSwitchConfig** = {}
- static unsigned long **ADC_Internal_Clock** = 0
- static **s_gpioMultiplexData ADC_gpioMultiplexData** [ADC_NUM_CONFIGURED_CHANNELS]
- static **s_ADCSensorConfig orientationAccelerometerConfig** = {}
- static **s_ADCSensorConfig orientationGyrometerConfig** = {}
- static **s_ADCSensorConfig orientationGyrometerReferenceConfig** = {}
- static **s_ADCSensorConfig steeringPotentiometerConfig** = {}
- static **s_ADCSensorConfig batteryVoltageSensorConfig** = {}
- static **s_UARTConfig rs232UARTConfig** = {}
- static **s_UARTConfig bluetoothUARTConfig** = {}
- static **s_StatusLED redStatusLEDConfig**
- static **s_StatusLED greenStatusLEDConfig**

2.3.1 Detailed Description

This class contains static variables only, which hold the configuration parameters for all other classes used by the segway project.

In **Configuration.h** (p. **??**) the variables and structs are declared. In **Configuration.cpp** (p. **??**) the variables are defined and initialized with zero. In **init()** (p. 5) the variables are set to the configuration values.

This behavior allows calculations to be made within **init()** (p. 5).

Definition at line 17 of file Configuration.h.

2.3.2 Member Data Documentation

2.3.2.1 Configuration::s_gpioMultiplexData Configuration::ADC_gpioMultiplexData [static]

Initial value:

```
= {


}
```

Definition at line 118 of file Configuration.h.

The documentation for this class was generated from the following files:

- Configuration/Configuration.h
- Configuration/Configuration.cpp

2.4 DebugMode Class Reference

Public Member Functions

- void **main** ()

2.4.1 Detailed Description

Definition at line 11 of file DebugMode.h.

The documentation for this class was generated from the following files:

- DebugMode/DebugMode.h
- DebugMode/DebugMode.cpp

2.5 GPIOSensor Class Reference

Diese Klasse wird benutzt um allgemein Pins zu steuern bzw. abzufragen, z.B. den Fußschalter.

```
#include <GPIOSensor.h>
```

Public Member Functions

- **GPIOSensor** ()

  Konstruktor wird nicht benutzt.

- void **init** (**Configuration::s_GPIOSensorConfig** *thisGPIOSensorConfig_)

  Schaltet den GPIO Pin frei und den Glitch Filter an. Optional auch den Pull-Up Widerstand.
- bool **getValue** ()

2.5.1 Detailed Description

Diese Klasse wird benutzt um allgemein Pins zu steuern bzw. abzufragen, z.B. den Fußschalter.

Definition at line 64 of file GPIOSensor.h.

2.5.2 Constructor & Destructor Documentation

2.5.2.1 GPIOSensor::GPIOSensor ( )

Konstruktor wird nicht benutzt.

Destruktor setzt den Pull-Up Widerstand und den Glitch-Filter zurück.

Definition at line 7 of file GPIOSensor.cpp.

2.5.3 Member Function Documentation

2.5.3.1 bool GPIOSensor::getValue ( )

Returns

Den aktuellen, binären Wert des Pins.

Definition at line 41 of file GPIOSensor.cpp.

2.5.3.2 void GPIOSensor::init (**Configuration::s_GPIOSensorConfig** * *thisGPIOSensorConfig_*)

Schaltet den GPIO Pin frei und den Glitch Filter an. Optional auch den Pull-Up Widerstand.

Parameters

thisGPIO↩ SensorConfig↩ _	Initialisierungseinstellungen

Definition at line 26 of file GPIOSensor.cpp.

The documentation for this class was generated from the following files:

- Sensor/GPIOSensor.h
- Sensor/GPIOSensor.cpp

2.6 Motor Class Reference

Motor (p. 7) class for AVR32UC3B offers.

`#include <Motor.h>`

Public Member Functions

- **Motor** ()

*Konstruktor Creates a new **PWM** (p. 10) object that provides the HAL.*

- ~**Motor** ()

  Destruktor wird nicht benutzt.
- bool **init** (**Configuration::s_MotorConfig** *thisMotorConfig_)

  Wendet die gegebenen Einstellungen.
- bool **setSpeed** (unsigned char ratioOn)

  *Leitet den Aufruf an den **PWM** (p. 10) weiter.*
- unsigned char **getSpeed** ()

  *Leitet den Aufruf an den **PWM** (p. 10) weiter.*
- void **setDirection** (bool forward)

  Legt die Richtung fest.

Static Public Member Functions

- static void **initEnablePin** ()

  Initialises the pin that is used for enabling/disabling the motor.
- static void **setEnabled** (bool enabled)

  *Schaltet den **Motor** (p. 7) frei / sperrt den **Motor** (p. 7).*
- static bool **getIsEnabled** ()

2.6.1 Detailed Description

Motor (p. 7) class for AVR32UC3B offers.

This class uses the **PWM** (p. 10) class and controls some GPIO-Pins to provide motor control. The motor speed can be set between 0 and 255, but is limited to Configuration::PWM_maxPWMRatio. The motor direction can be set to forward or backward. The motors can be enabled and disabled, which means enabling/disabling the H Bridges of all motors.

In this class "speed" is the same as "PWM": a value from 0 to 255 representing the **PWM** (p. 10) ratio.

Attention: all motors share the same enable pin. Attention: when class is destroyed or cleanUp() is called, the enable pin will be unininitalized.

Definition at line 20 of file Motor.h.

2.6.2 Member Function Documentation

2.6.2.1 bool Motor::getIsEnabled ( ) [static]

Returns

Ob der **Motor** (p. 7) freigegeben ist

Definition at line 95 of file Motor.cpp.

2.6.2.2 unsigned char Motor::getSpeed ( )

Leitet den Aufruf an den **PWM** (p. 10) weiter.

Parameters

Der	zu setzende Wert

Returns

Die **PWM** (p. 10) Antwort

Definition at line 64 of file Motor.cpp.

2.6.2.3 bool Motor::init (Configuration::s_MotorConfig * *thisMotorConfig_*)

Wendet die gegebenen Einstellungen.

Parameters

*thisMotor↩ Config_*	Initialisierungseinstellungen

Returns

true

Definition at line 37 of file Motor.cpp.

2.6.2.4 void Motor::setDirection (bool *forward*)

Legt die Richtung fest.

Parameters

forward	Forwärts

Definition at line 74 of file Motor.cpp.

2.6.2.5 void Motor::setEnabled (bool *enabled*) [static]

Schaltet den **Motor** (p. 7) frei / sperrt den **Motor** (p. 7).

Parameters

enabled	

Definition at line 85 of file Motor.cpp.

2.6.2.6 bool Motor::setSpeed (unsigned char *ratioOn*)

Leitet den Aufruf an den **PWM** (p. 10) weiter.

Parameters

Der	zu setzende Wert

Returns

Die **PWM** (p. 10) Antwort

Definition at line 53 of file Motor.cpp.

The documentation for this class was generated from the following files:

- Antrieb/Motor.h
- Antrieb/Motor.cpp

2.7 PWM Class Reference

Pulse Width Modulation.

`#include <PWM.h>`

Public Member Functions

- **PWM** ()
  Konstruktor wird nicht benutzt.
- ~**PWM** ()
  Destruktor wird nicht benutzt.
- bool **init** (**Configuration::s_PWMConfig** *thisPWMConfig_)
  *Setzt den Multiplex Wert auf **PWM** (p. 10). Setzt die Duty Cycle Period nach der gegebenen Frequenz.*
- bool **setChannelPWMRatio** (unsigned char ratioOn, bool capRatioOn=false)
  Setzt den Duty Cycle Wert im gegebenen Verhältnis zur Periode durch das Schreiben in das Channel Update Register.
- unsigned char **getChannelPWMRatio** ()
- bool **isChannelEnabled** ()
- bool **setChannelEnabled** (bool enabled)
  Schaltet den Kanal frei / sperrt den Kanal.

2.7.1 Detailed Description

Pulse Width Modulation.

Diese Klasse bietet Funktionen um die Motoren auf Hardwareebene anzusteuern.

Definition at line 63 of file PWM.h.

2.7.2 Member Function Documentation

2.7.2.1 unsigned char PWM::getChannelPWMRatio ( )

Returns

Den aktuellen Duty Cycle Wert im Bereich [0, 255].

Definition at line 83 of file PWM.cpp.

2.7.2.2 bool PWM::init (Configuration::s_PWMConfig * *thisPWMConfig_*)

Setzt den Multiplex Wert auf **PWM** (p. 10). Setzt die Duty Cycle Period nach der gegebenen Frequenz.

Parameters

*thisPWM↩ Config_*	Initialisierungseinstellungen

Definition at line 27 of file PWM.cpp.

2.7.2.3 bool PWM::isChannelEnabled ( )

Returns

Ob der Kanal freigegeben ist

Definition at line 91 of file PWM.cpp.

2.7.2.4 bool PWM::setChannelEnabled (bool *enabled*)

Schaltet den Kanal frei / sperrt den Kanal.

Parameters

enabled	

Definition at line 101 of file PWM.cpp.

2.7.2.5 bool PWM::setChannelPWMRatio (unsigned char *ratioOn,* bool *capRatioOn* = `false`)

Setzt den Duty Cycle Wert im gegebenen Verhältnis zur Periode durch das Schreiben in das Channel Update Register.

Parameters

ratioOn	Der zu setztende Wert
capRatioOn	Ob der ratioOn Wert auf ein Maximum maxPWMRatio beschränkt werden soll

Definition at line 65 of file PWM.cpp.

The documentation for this class was generated from the following files:

- Antrieb/PWM.h
- Antrieb/PWM.cpp

2.8 Configuration::s_ADCSensorConfig Struct Reference

Public Attributes

- unsigned long **ADCChannelID**
- signed long **zeroOffset**
- float **slopeFactor**
- bool **useZeroOffset**
- bool **useSlopeFactor**

2.8.1 Detailed Description

Definition at line 58 of file Configuration.h.

The documentation for this struct was generated from the following file:

- Configuration/Configuration.h

2.9 Configuration::s_gpioMultiplexData Struct Reference

Public Attributes

- bool **configured**
- unsigned char **port**
- unsigned long **pin**
- unsigned char **multiplexRegisterValue**

2.9.1 Detailed Description

Definition at line 52 of file Configuration.h.

The documentation for this struct was generated from the following file:

- Configuration/Configuration.h

2.10 Configuration::s_GPIOSensorConfig Struct Reference

Public Attributes

- unsigned char **port**
- unsigned long **pin**
- bool **pullupEnabled**

2.10.1 Detailed Description

Definition at line 47 of file Configuration.h.

The documentation for this struct was generated from the following file:

- Configuration/Configuration.h

2.11 Configuration::s_MotorConfig Struct Reference

Public Attributes

- unsigned char **directionPinPort**
- unsigned long **directionPinPin**
- bool **directionPinForwardValue**
- **s_PWMConfig** * **PWMConfig**

2.11.1 Detailed Description

Definition at line 41 of file Configuration.h.

The documentation for this struct was generated from the following file:

- Configuration/Configuration.h

2.12 Configuration::s_PWMConfig Struct Reference

Public Attributes

- unsigned char **channelID**
- unsigned char **maxPWMRatio**
- unsigned long **frequency**
- unsigned char **GPIO_port**
- unsigned char **GPIO_pin**
- unsigned char **GPIO_multiplexRegisterValue**

2.12.1 Detailed Description

Definition at line 33 of file Configuration.h.

The documentation for this struct was generated from the following file:

- Configuration/Configuration.h

2.13 Configuration::s_StatusLED Struct Reference

Public Attributes

- unsigned char **port**
- unsigned long **pin**

2.13.1 Detailed Description

Definition at line 85 of file Configuration.h.

The documentation for this struct was generated from the following file:

- Configuration/Configuration.h

2.14 Configuration::s_UARTConfig Struct Reference

Public Attributes

- volatile char * **usart_address**
- unsigned long **baudRate**
- unsigned char **charlength**
- unsigned char **paritytype**
- unsigned char **channelmode**
- unsigned char **stopbits**
- unsigned char **RX_GPIO_port**
- unsigned char **RX_GPIO_pin**
- unsigned char **RX_GPIO_multiplexRegisterValue**
- unsigned char **TX_GPIO_port**
- unsigned char **TX_GPIO_pin**
- unsigned char **TX_GPIO_multiplexRegisterValue**

2.14.1 Detailed Description

Definition at line 69 of file Configuration.h.

The documentation for this struct was generated from the following file:

- Configuration/Configuration.h

2.15 Segway Class Reference

Contains main segway functionality as there are initialization and the controlling algorithm.

`#include <Segway.h>`

Public Member Functions

- **Segway** ()

  Constructor. Does minimal initialization.
- void **timerFunction** ()

  Function called by the timer. It contains the control algorithm.
- void **main** ()

  Main loop.

2.15.1 Detailed Description

Contains main segway functionality as there are initialization and the controlling algorithm.

This class creates and initializes helper objects of sensor, motor and communication classes. Then, it uses these objects to execute the segway controlling algorithm. Debug data is provided via bluetooth using the **UART** (p. 16) class.

Definition at line 20 of file Segway.h.

2.15.2 Member Function Documentation

2.15.2.1 void Segway::main ( )

Main loop.

Initializes helper objects, then starts the timer containing the controlling algorithm. Before starting the timer, it is made sure that the foot switch is not pressed.

While running, this function sends debug data via the bluetooth interface. Different debug values can be enabled by uncommenting them here and in **timerFunction()** (p. 14).

Definition at line 341 of file Segway.cpp.

2.15.2.2 void Segway::timerFunction ( )

Function called by the timer. It contains the control algorithm.

The function first receives the sensor values and then calculates and sets the motor's **PWM** (p. 10) according to the controlling algorithm.

Definition at line 25 of file Segway.cpp.

The documentation for this class was generated from the following files:

- Segway/Segway.h
- Segway/Segway.cpp

2.16 Simulation Class Reference

Public Member Functions

- void **main** ()

2.16.1 Detailed Description

Definition at line 11 of file Simulation.h.

The documentation for this class was generated from the following files:

- Segway/Simulation.h
- Segway/Simulation.cpp

2.17 Timer Class Reference

Public Member Functions

- bool **prepareTimer** (unsigned long frequency)
- bool **initTimer** (unsigned long frequency)
- void **setIsTimerEnabled** (bool enabled)
- void **setIsTimerInterruptEnabled** (bool enabled)
- bool **getIsTimerEnabled** (void)
- bool **getIsInterruptEnabled** (void)

Static Public Member Functions

- static void **resetInterruptFlag** (void)

2.17.1 Detailed Description

Definition at line 65 of file Timer.h.

2.17.2 Member Function Documentation

2.17.2.1 bool Timer::getIsInterruptEnabled (void)

Der Rueckgabewert liefert, ob ein Interrupt aktiviert wurde.

Definition at line 67 of file Timer.cpp.

2.17.2.2 bool Timer::getIsTimerEnabled (void)

Der Rueckgabewert liefert, ob der **Timer** (p. 15) aktiviert wurde.

Definition at line 63 of file Timer.cpp.

2.17.2.3 bool Timer::initTimer (unsigned long *frequency*)

Wenn die Methode prepareTimer richtig ausgefuehrt wurde wird ein Interrupt aktiviert.

Definition at line 34 of file Timer.cpp.

2.17.2.4 bool Timer::prepareTimer (unsigned long *frequency*)

In dieser Methode wird die Kurve 2 (WAVSEL 2) gewaehlt, um das Hochzaehlen des Timers zu realisieren. Diese Kurve ist im Daten Blatt auf Seite 484 dargestellt. Die Kurve ist eine Saegezahnkurve mit einem variablen RC-Wert als Maximalwert. Ausserdem wird die Frequenz uebergeben, mit welcher dann der Maximale RC-Wert berechnet wird und somit auch die Haeufigkeit der Interrupts pro Sekunde.

Definition at line 20 of file Timer.cpp.

2.17.2.5 void Timer::resetInterruptFlag (void) [static]

Liest das Interrupt-Statusregister um den Interrupt zurueckzusetzen.

Definition at line 59 of file Timer.cpp.

2.17.2.6 void Timer::setIsTimerEnabled (bool *enabled*)

Diese Methode aktiviert,deaktiviert und startet den **Timer** (p. 15).

Definition at line 42 of file Timer.cpp.

2.17.2.7 void Timer::setIsTimerInterruptEnabled (bool *enabled*)

Diese Methode aktiviert oder deaktiviert einen Interrupt.

Definition at line 51 of file Timer.cpp.

The documentation for this class was generated from the following files:

- Timer/Timer.h
- Timer/Timer.cpp

2.18 UART Class Reference

Public Member Functions

- bool **init** (**Configuration::s_UARTConfig** ∗thisUARTConfig_)
- bool **enableInPinSelector** (bool enabled)
- bool **isDataAvailable** ()
- unsigned long **getData** ()
- void **sendChar** (unsigned long data)
- void **sendString** (const char ∗text)
- void **sendNumber** (long number)
- void **sendNumber** (unsigned long number)

2.18.1 Detailed Description

Definition at line 7 of file UART.h.

The documentation for this class was generated from the following files:

- UART/UART.h
- UART/UART.cpp