HSP Projektbericht

Philipp Eidenschink, Florian Laufenböck, Tobias Schwindl
Matrikelnummern : 3080919, 2894759, 3080498

22. April 2017

Inhaltsverzeichnis

Todo list

1. Einleitung

Dieser Projektbericht beschreibt die Tätigkeiten der Autoren im Laufe des HSP[1] im Wintersemester 2016/2017. Diese beinhalten im wesentlichen die Ersetzung der kompletten Hardwarearchitektur des ALF und dessen Raspberry Pi auf eine neuere, verbesserste Hardwarearchitektur, um mehr Leistungsreserven zu besitzen.

Motivation für das Ersetzen des Raspberry Pi

- Leistungsreserven: Da die alte Hardwareplattform des Raspberry Pi keine ausreichende Performace für zusätzliche Anwendungen, wie zum Beispiel SLAM Algorithmen, besitzt, wurde die Entscheidung getroffen, eine komplett neue Hardware zu erstellen.
- Um eine möglichst hohe Flexibilität zu erreichen, wurde dabei auf ein FPGA gesetzt. Somit ist es einfach möglich neue Anwendungen hinzuzufügen oder bestehende Anwendungen zu erweitern.

1.1. Lesehinweise

Dieses Dokument ist in mehrere Kapitel gegliedert. Bevor dieses Dokument gelesen und verstanden werden kann folgen hier einige Lesehinweise:

- Coderepository - Der gesamte Code und alle relevante Dokumenation zu dem Projekt befindet sich aktuell auf Github. Der Link zum aktuellen Stand ist https://github.com/Alabamajack/Garfield.
- Weitere Dateien - Leider beschränkt Github die maximale Dateigröße auf 100MB. Aus diesem Grund liegen Dateien, die größer als 100MB sind, auf dem Laborlaufwerk unter dem Verzeichnis. Diese Daten werden im Dokument extra gennant.
- Pfade - Alle Dateipfade, die im Dokument genannt werden und für die keine weiteren Informationen angegeben sind, beziehen sich auf das root-Verzeichnis des Coderepositories. Weitere Pfade die verwendet werden sind:
  - Pfad im Linux Kernel: Diese Pfade sind absolute Pfade innerhalb einer bestimmten Version des Linux Kernels (nur Releases, keine Pre-Releases etc.). Solche Pfade haben den Prefix $LINUX\_VX.X$ wobei $X.X$ die Version des Linux Kernel bezeichnet, der verwendet wurde.

[1] Hauptseminar Projektstudium

– Pfade auf dem HPS[2] System - Dies sind Linux Distributionspfade. Alle verwendeten Pfade haben als root-Verzeichnis das *home*-Verzeichnis des Standardbenutzers *ubuntu*. Als Prefix dafür wird *HPS* genannt. Sollte ein übergeordneter Pfad zum *home*-Verzeichnis bezeichnet werden ist der Prefix *HPS_ boot*.

[2]Hard Processor System

1.2. Eingesetzte Tools

Im folgenden Abschnitt soll ein kurzer Überblick über die verschiedenen eingesetzten Tools gegeben werden. Diese Beschreibung ist nicht vollständig da Grundkentnisse (wie kompilieren eines Linux Kernels aus den Sourcen) vorausgesetzt werden.

1.2.1. FPGA Entwicklung

1.2.1.1. Quartus

Das Tool Quartus bildet die Grundlage um für Altera (bzw. inzwischen Intel) FPGAs[3] entwickeln zu können. Im Prinzip ist es eine Sammlung von verschiedenen Tools, die über eine GUI gesteuert werden. Alle relevanten Prozesse (Building, Generieren von IP-Cores, Systemanalyze etc.) sind auch (u.U. sogar mächtiger) als Kommandozeilentools verfügbar. Das vollständige Handbuch ist unter Quartus Handbuch zu finden (Achtung: 1939 Seiten!, und das ist nur der erste Teil). Als Überblick und um mit dem Garfield Projekt zu starten gibt es einige nützliche Links und Tutorials wie z.B. Altera University Programm - Start. Eine weitere, sehr empfehlenswerte Anlaufstelle bei Problemen oder auf der Suche nach Application Notes, Tutorials, HowTos und auch Vorträgen ist rocketboards.org. Dabei handelt es sich um die offizielle Open-Source Sammlung rund um Intel FPGAs.

Die OTH hat eine Reihe von Lizenzen für die gesamte Altera Toolchain (Quartus, SoC EDS, IP-Cores etc.). Wird das Garfield Projekt mit der unlenzierten Version syntethisiert, kompiliert Quartus automatisch einen "Ablaufzeitstempel" mit ins Design. Der NIOS2 Prozessor und einige IP[4]-Cores sind damit nur ca. 30 Minuten lauffähig! Die Lizenzen verwaltet Herr Altmann. Dieser hat auch mind. einen WLAN-USB Stick an dessen MAC-Addresse die Lizenz gebunden ist. Es ist nicht empfehlenswert die Lizenz an eine feste MAC-Addresse eines privaten PCs zu binden, da diese Lizenz dann für andere Studenten verbraucht wäre. Die Installation von Quartus sollte im Hochschulnetz erfolgen da die Downloadgröße ca. 20GB beträgt.

1.2.1.2. Qsys

Bei QSYS handelt es sich um Intels System Integrations Tool. Es ist eine sehr abstrakte Variante sich ein komplettes FPGA System zusammenzuklicken und automatisch jegliche Hardwarebeschreibungen, evtl. notwendige Treiber für den NIOS2 etc. zu generieren. Nach der Generierung entsteht ein großer IP-Core, der abschließend in die eigene Pinbeschreibung eingebettet werden muss. Auch für dieses Tool kann am besten wieder auf ein Tutorial QSYS Tutorial oder auf rocketboards.org hingewiesen werden. Da das Garfield Projekt auch zwei selbstgeschriebenen Hardwarekomponenten beinhaltet, müssen die Pfade für diese QSYS bekanntgemacht werden. Dies ist nicht nur notwendig für die

[3] Field Programmable Gate Arrays
[4] Intellectual Property

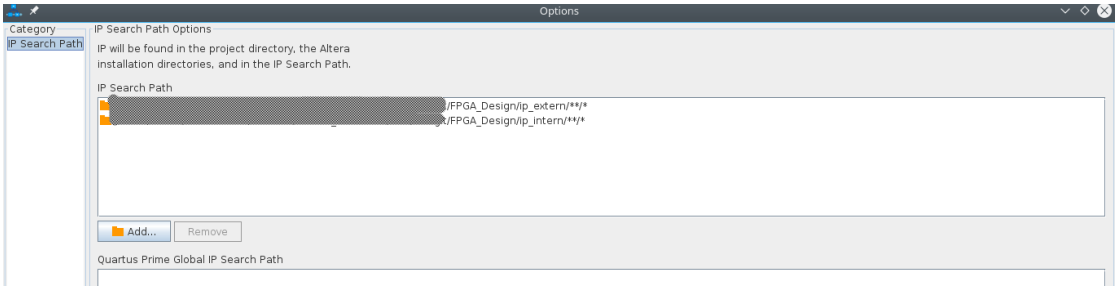


Abbildung 1.1.: Einstellungen für die selbstgeschriebenen IP-Cores

Person, die das Hardwaredesign anpasst/syntethisiert, sondern auch für Beteiligte, die Code für den NIOS2 schreiben wollen und auf die HAL[5]-Generierung des Tools angewiesen sind. Dazu müssen in QSYS die Einstellungen wie in Abbildung 1.1 dargestellt. Bei geöffnetem QSYS muss unter `Tools->Options->IP Search Path` die Pfade zu den IP-Cores angegeben werden.

1.2.1.3. Quartus Programmer

Der Quartus Programmer dient dazu, entweder das FPGA direkt mit der entsprechenden Image Datei (`*.sof`-Endung) oder das angeschlossene Flash mit einer `*.jic` Datei zu flashen. In den entsprechenden Quartus Tutorials sind auch kleine HowTos enthalten, wie mit dem Programmer umzugehen ist. Auf der für das Board zugeschnittene System-CD befindet sich auch eine `DE0-Nano-SoC_User_manual.pdf`. In diesem ist auch beschrieben, wie man eine Datei für den Flash erstellt und dies auf den Flash lädt. Um außerdem die Applikation für den NIOS2 auf den Flash zu programmieren müssen mit dem NIOS2-Programmer zwei Dateien erstellt werden, die jeweils mit *.flash enden. Eine Datei ist dabei für das Hardwaredesign, die andere für den Applikationscode der bei Start geladen wird. Eine Beschreibung des NIOS2-Programmers kann unter diesen Link gefunden werden.

1.2.2. Eclipse NIOS2 - Beschränkungen, besondere Einstellungen

Das in der Version 16.1 verwendete Quartus mit dem mitgeliefertem GCC Compiler hat einige Einschränkungen bezüglich der Verwendung von einigen C++ Features. Alle in den C++ Standardbibliotheken vorhandenen STL Container, z. B. std::vector, std::stack, std::map usw., sind nicht benutzbar. Außerdem ist es nicht möglich die std::string Klasse zu benutzen. Die Verwendung solcher Features führt dazu, dass der Speicher nicht mehr ausreicht. Für die Verwendung dieser Klassen sind auf dem NIOS2 mit der aktuellen Toolchain ca 700KB RAM nötig, allerdings sind nur 128KB RAM vorhanden. Dies wurde vom ALTERA Support Team direkt bestätigt mit der Angabe, dass der Einsatz von C++ im Vergleich zu C im Moment nicht effizient möglich ist und externer Speicher

[5]Hardware Abstraction Layer

für die Verwendung von C++ angeraten ist. Alle anderen Features hingegen sind, soweit bekannt, ohne Einschränkungen einsetzbar.

Für die Unterstützung von c++11 ist eine kleine manuelle Anpassung des Makefiles nötig, welches die Toolchain mit Erstellung eines BSP automatisch generiert. In der Sektion

- # Arguments only for the C++ compiler.

ist die Ergänzung folgenden Flags nötig: -std=c++11; da diese Einstellung über die GUI aktuell nicht erhalten bleibt. Die Sektion sollte dann folgendermaßen aussehen:

- # Arguments only for the C++ compiler.
  APP_CXXFLAGS := $(ALT_CXXFLAGS) $(CXXFLAGS)
  -std=c++11

Diese Einstellung ist auch unbedingt nötig, da einige c++11 Features verwendet wurden und demzufolge ohne diesem Flag die Applikation nicht erfolgreich kompiliert. Der Compiler sollte auch c++14 Features unterstützen, allerdings werden solche in der aktuellen Codebasis nicht verwendet.

2. Hardware

2.1. Schaltplan

Abbildung 2.1 zeigt den erstellten Schaltplan des HSP. Dieser soll nachfolgend kurz beschrieben werden.
Alle ein- und ausgeheneden Signale mit Ausnahme der SPI Pins zur Ansteuerung des Displays (vgl. Abb. 2.1 JP4) werden über Levelshifter geführt. Dies ermöglicht es zum einen alle Signale an die jeweils notwendigen Pegel anzupassen und zum anderen den maximalen vom FPGA bereitgestellten Strom nicht zu überschreiten. Dadurch ergeben sich die zwei Logikpegel 3,3V und 5V im System. Über den IIC Port werden alle Ultraschallsensoren und die MPU6050 angesprochen. Es wurden die internen pull-up Widerstände des IIC Ports aktiviert um dessen Funktionsfähigkeit sicherzustellen. Über den PWM Generator wird die Lenkung und der Motortreiber für die Geradeausfahrt angesteuert. Zur Ansteuerung der Beleuchtung und dem Setzen der Richtung des Motors werden einfache GPIO Pins benutzt. Der Schaltplan enthält zudem die Ansteuerung des Rotary Encoders zum Messen der Drehzahl des Motors. Da dieser jedoch unerwarteterweise nicht funktionsfähig war, sind die betrefenden Stellen im Schaltplan entsprechend gekennzeichnet.

2.2. FPGA Design

Die Beschreibung des FPGA wird, soweit möglich, mit dem Systemintegrationstool QSYS, das Teil der Quartus Toolchain ist, durchgeführt. Das Mapping zwischen QSYS-System und Pins wird klassisch in VHDL beschrieben. Das Top-Level-File des Systems ist `FPGA_Design/Garfield_Design/Garfield.vhdl`. Dort wird das von QSYS erzeugt System und einige kleine IP-Cores zusammengeführt und auf definierte Aus-/Eingänge geführt. Diese Ein-/Ausgänge werden dann über den *Pin-Planner* auf die physikalischen Pins geführt.
Im Projektverzeichnis befinden sich alle Dateien, die für den FPGA Teil relevant sind, unter `FPGA_Design`. Die Struktur ab diesem Ordner ist wie folgt aufgebaut:

- `Datasheets` - Einie Datenblätter und Application Notes zu dem FPGA Teilprojekt
- `Garfield_Design` - In diesem Ordner befinden sich die Quartus Projektdateien, Konfigurationsdateien und das QSYS Projekt.
- `ip_extern` - Eine Sammlung von externen IP-Cores, die im Projekt verwendet wurden. Es befinden sich dort nur die IP-Cores, die nicht von Altera stammen oder nicht direkt in QSYS verfügbar sind.

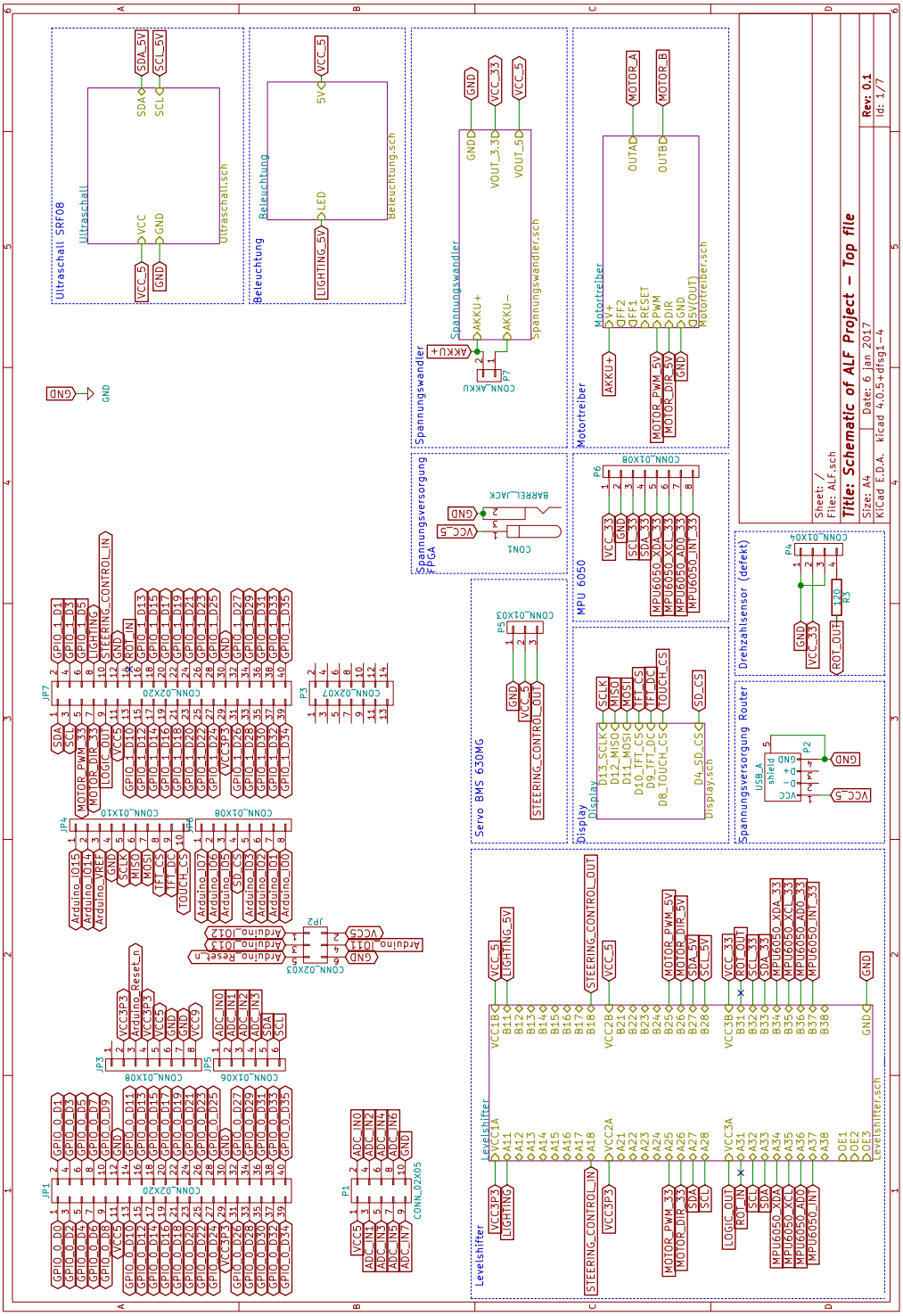


Abbildung 2.1.: Schaltplan mit FPGA und verwendeter Peripherie

- `ip_intern` - Alle IP-Cores, die für dieses Projekt entwickelt wurden.
- `output_files` - In jedem Unterordner innerhalb dieses Ordners befinden sich FPGA Images und die entsprechenden Konfigurationsdateien um ein Softwareprojekt dafür zu bauen.

Im folgenden werden die alle Systemkomponenten, die für das Garfield-Projekt erzeugt wurden, beschrieben.

2.3. IP-Cores

Abbildung 2.2 zeigt eine Übersicht der eingesetzten IP-Cores und deren Verbindung zur Außenwelt. Ausgenommen sind die IP-Cores, die für die Kommunikation mit dem HPS benötigt werden. Die nachfolgende Tabelle beschreibt die Funktion der einzelnen IP-Cores im Detail und Besonderheiten dazu.

Name	**Beschreibung**
SPI-0	Stellt einen SPI-Datenbus mit 24MHz Clock-Frequenz zur Verfügung. Es werden insgesamt 3 Chipselect Signale zur Verfügung gestellt, wobei aktuell nur eines für das Display benutzt wird. In der aktuellen Ausbaustufe wird nur das Display auf dem Arduino-Header auf dem FPGA angesteuert.
I2C-0	Stellt einen I2C Datenbus zur Verfügung. Der IP-Core stammt von opencores.org und wurde manuell integriert. Er stellt u.a. eine eine in Software änderbare Clock-Frequenz zur Verfügung und bindet die Ultraschallsensoren und die MPU-6050 an das System an.
GPIO-X	Die verschiedenen GPIO Cores dienen dazu einfache Peripherie anzubinden. Dazu gehören die LEDs, die Dip-Switches, die Buttons und generische IOs, die im Projekt benötigt werden um z.B. die Drehrichtung des Motors einzustellen.
PWM X	Die beiden PWM IP-Cores erzeugen Signale zur Geschwindigkeitssteuerung und für den Lenkmotor.
Rotary-Encoder	Der Rotary Encoder zählt die steigenden Flanken des Drehzahlencoders. Durch Abfragen des Ergebnisregisters in regelmäßigen festen Zeitinervallen kann die aktuelle Geschwindigkeit, die an den Rädern anliegt, gemessen werden. Leider funktioniert der Drehzahlencoder aktuell nicht. Um das Signal zu nutzen müsste man die Hardware neu aufbauen bzw. ersetzen.

Clock & PLL[6]	Die externe Referenzclock taktet mit 50 MHz. Dieses Signal wird über eine PLL allen beteiligten IP-Cores bereitgestellt. Auch die FPGA-HPS Bridges werden mit dem 50MHz Signal gespeist. Einzige Ausnahme bildet der SPI-0 Core. Um eine Frequenz von 24MHz zu erreichen (die maximale Frequenz mit der das Display angesprochen werden darf) wird ein vielfaches dieser Frequenz benötigt. Das nächsthöhere verfügbare vielfache der 24MHz sind 48MHz. Die selbst geschriebenen IP-Cores sind von der Frequenz der PLL abhängig. Erhöht man die Frequenz der PLL auf z.B. 100MHz um den einzelne Funktionen eine höhere Frequenz zur Verfügung zu stellen, muss man die Frequenz in den IP-Cores manuell anpassen!
SysID	Mit der System ID (in Kombination mit einem Zeitstempel) kann man das Hardware Design eindeutig identifizieren. Dies ist hilfreich wenn mehrere Hardware- und Softwareversionen existieren, die parallel entwickelt werden. Um Zugriffsfehler auf Register oder ähnliches zu vermeiden, kann die Software die System-ID nutzen um Funktionen ab- bzw. zuzuschalten.
JTAG-UART	Mit Hilfe dieses Cores kann man printf ähnliche Ausgaben für Debug-Ausgaben an einen angesteckten PC schicken.
System-timer	Der Systemtimer ist ein kontinuierlich laufender Timer, der sich alle 1ms automatisch erhöht. Außerdem erzeugt er ein Interrupt, das FreeRTOS zur internen Zeitbestimmung nutzt.
NIOS2	Hierbei handelt es sich um eine Softcore-CPU. Diese wird von Altera zur Verfügung gestellt (inkl. Toolchain) und kann unbegrenzt benutzt werden (mit entsprechender Lizenz). Es handelt sich um eine 32-bit RISC[7] Architektur die durchaus eine weite Verbreitung im industriellen Umfeld genießt. Weiter Informationen dazu findet man unter https://www.altera.com/products/processors/overview.html
Onchip Memory	Ein einfacher IP-Core, der Speicherbausteine auf dem FPGA nutzt um RAM(hier genutzt) oder ROM (nicht genutzt) zu erzeugen. Dieser Speicher kann dann von einem Prozessor (hier der NIOS2) als Instruktions- und Datenspeicher genutzt werden. In der aktuellen Ausbaustufe ist die Speichergröße mit 128kB angegeben.
EPCS Flash Controller	Dabei handelt es sich um einen einfachen IP-Core um den auf dem Flashspeicher abgelegten Programmcode von diesem zu laden und auszuführen.

Abbildung 2.3 zeigt die IP-Cores, die für die Kommunikation zwischen FPGA und HPS benötigt/eingesetzt werden. Auf der linken Seite der Abbildung ist das HPS System illustriert. Auf dieser Seite sind im wesentlichen drei Hardwareeinheiten an der Kommunikation beteiligt:

[6] Phase-locked loop
[7] Reduced Instruction Set Computer

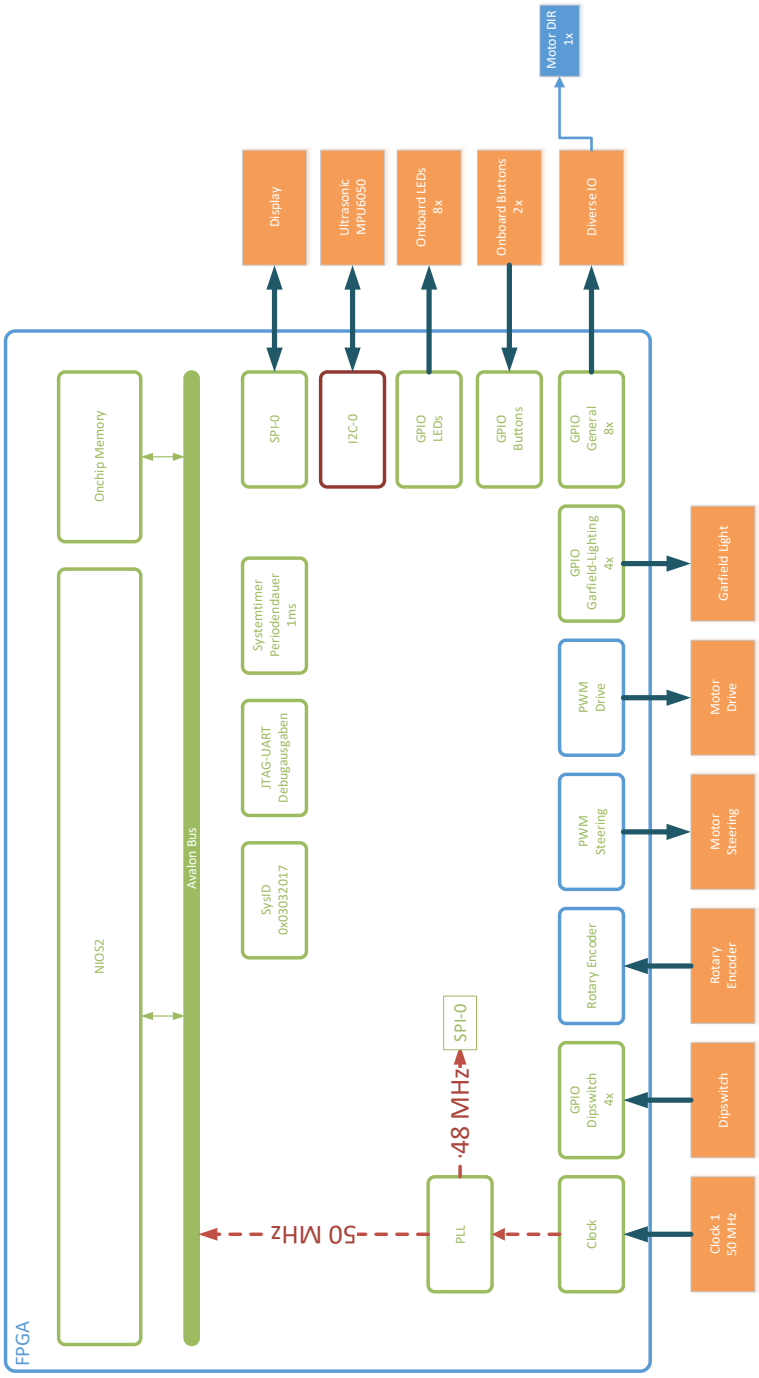


Abbildung 2.2.: Übersicht der (meisten) eingesetzten IP-Cores. Die Abbildung verzichtet auf die Darstellung der Teile, die für die Kommunikation mit dem HPS zuständig sind. Blau umrandet sind Cores, die im Rahmen des Projekts selbst progammiert wurden, grün diejenigen, die Teil der Altera Toolchain sind und rot externe IP-Cores von opencores.org

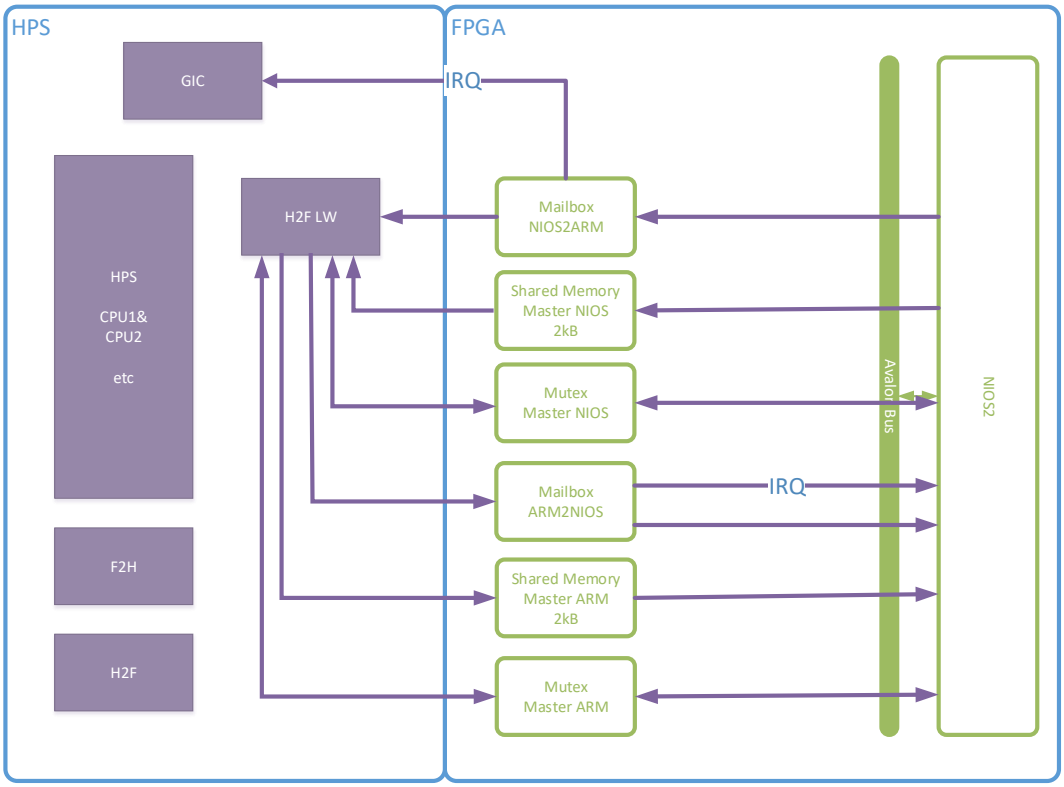


Abbildung 2.3.: IP-Cores und deren Kontrollfluss, die an der Kommunikation zwischen NIOS2 und HPS beteiligt sind.

- GIC - Der ARM *General Interrupt Controller* : Dieser Controller ist ein sehr mächtiger Interrupt Controller, der u.a. die Interruptverarbeitung an die einzelnen CPUs verteilt. Insgesamt stehen 64 Interrupts zur Verfügung, die aus dem FPGA heraus ausgelöst werden können. Auf dem eingesetzten Cyclone V beginnen diese mit der Interrupt ID 72 vom GIC. Genauere Informationen zum GIC kann man entweder auf der Homepage von ARM oder unter [1] erhalten.

- H2F LW - Die HPS2FPGA Leightweight Bridge : Dies ist eine der drei Bridges, mit denen zwischen FPGA und HPS kommuniziert werden kann. Dies ist keine High-Performance Bridge, es ist aber keine großen Änderungen notwendig, das System auf eine der anderen Bridges umzubauen. Diese Bridge **muss** aktiviert werden, bevor über sie kommuniziert werden kann. Ist die Bridge nicht aktiviert, treten *Segmentation Faults* auf (kein gültiger Speicherbereich). Im Prinzip befindet sich "hinter"der Bridge ein Speicherbereich, der durchgehend addressiert werden kann um direkt in Register zu schreiben. Auch lesende Zugriffe daraus können erfolgen [2].

- CPUs - Die ARM A9 Applikationsprozessoren dienen zur Verarbeitung der Interrupts bzw. zum triggern der einzelnen IP-Cores.

Es folgt eine Beschreibung der IP-Cores, die für die Kommunikation gebraucht werden. Da die Kommunikationseinheiten in beide Kommunikationsrichtungen gleich aufgebaut sind, beschränkt sich die Beschreibung auf einen Richtung:

- Mailbox X2Y : Die Mailbox ist ein einfacher IP-Core der Nachrichten von einem Buspartner (X, z.B. NIOS2) einem anderen Buspartner (Y, z.B. ARM) zur Verfügung stellt. Es gibt also einen Transmitter und einen Receiver. Beide sind über eigene Interfaces (und damit über ihren eigenen Addressbereich) an die Mailbox angeschlossen. Die Nachrichtenübermittlung erfolgt mit Hilfe von zwei Registern:
  - Command Register - Dieses Register kann vom Empfänger nur gelesen werden. Es dient dazu, ein Kommando oder Nachricht an den Empfänger zu senden. Ein schreibender Zugriff auf dieses Register vom Sender löst das zugehörige Interrupt aus, dass vom Empfänger verarbeitet werden muss.
  - Pointer Register - In diesem Register wird die Addresse, in der die eigentliche Nachricht im Speicher steht, übertragen. Sollen nur ganz kleine Nachrichten (4 oder 8 Bytes) übertragen werden, kann man das Pointer und Command Register dazu benutzen, die Nachricht zu übertragen. In diesem Projekt wird aber die eigentliche Nachricht im Shared Memory übertragen, in der Mailbox nur die Addresse im Shared Memory und ein Kommando im Command Register

  Eine detailierte Beschreibung des Cores findet sich unter [5, 470ff]

- Shared Memory Master X - Dieser Speicher, der wie der Arbeitsspeicher des NIOS2 direkt im FPGA synthetisiert wird, dient der Nachrichtenübermittlung. Dort

werden die Nutzdaten einer Nachricht von X reingeschrieben und können zu einem späteren Zeitpunkt vom Empfänger Y ausgelesen werden. Es wurde sich bewusst dazu entschieden zwei Shared Memory zu benutzen um jegliche Kollisionen zu vermeiden bzw. die Fehlersuche zu vereinfachen. Die Größe beider Speicherbereiche beträgt jeweils 2kB. Dies reicht für die aktuellen Nachrichten leicht aus. Zu einem späteren Zeitpunkt können die Bereiche auch noch vergrößert werden, sollte der Speicher nicht groß genug sein Nachrichten zu übertragen.

- Mutex Master X - Dies ist ein spezieller IP-Core, der auch als Teil des Altera IP-Core Katalogs zur Verfügung stellt. Dieser hat nur ein Register, das hier betrachtet werden soll und erlaubt einen atomaren Mutex Zugriff auf geteilte Resourcen. Die geteilte Resource, die über diesen Mutex gesperrt wird ist der zugehöriger Shared-Memory. Die Referenz für diesen Core ist ebenfalls [5, 319ff]. Das Register besteht aus zwei Teilen: die oberen 16 Bit werden als Speicherplatz für die CPU-ID (der NIOS2 hat die ID 0x03, der ARM immer 0x01) benutzt. In die unteren 16 Bit kann ein beliebiger Wert gespeichert werden. Ein lesender Zugriff auf den Mutex ist immer möglich. Ein schreibender Zugrif ist nur möglich wenn
  - Die CPU-ID mit der CPU-ID übereinstimmt, dessen Wert man in das Register schreiben will.
  - (oder) Der Wert (untere 16-Bit) Null ist.

  Man kann also nur schreibend auf den Mutex zugreifen, wenn einem der Mutex bereits gehört oder der Mutex frei (=0) ist. Nach einem schreibenden Zugriff muss der Registerwert mit dem Wert der geschrieben wurde verglichen werden. Stimmen beide Werte überein, hat der Schreiber den Mutex gelockt, andernfalls ist der atomare Lock fehlgeschlagen.

2.4. Address-Map

Abbildung 2.4 zeigt die Addressen die im System benutzt werden und den zugehörigen Addressbereich der verfügbar ist. Diese Addressmap ist auch im QSYS-Projekt des Projektes verfügbar.

2.5. Eigenentwickelte IP-Cores

2.5.1. Der PWM-Generator

generiert ein PWM Signal auf die Ausgabeleitung. Der Registerzugriff ist in Tabelle 2.3 dargestellt

2.5.2. Der Rotary Encoder

IP-Core kann steigende Flanken von einer externen Flanke zählen, hat ein auslesbares Ergebnisregister (siehe Tabelle 2.7)und ein Controlregister (siehe Tabelle 2.5).

System: Garfield_system **Path:** mailbox_arm2nios_0

	fpga_only_master.master	...	hps_0.h2f_lw_axi_master	hps_only_master.master	nios2_gen2_0.data_mas...	nios2_gen2_0.instruction_...
timer_0_nios2.s1					0x0000_0000 - 0x0000_001f	
spi_0.spi_control_port					0x0000_0020 - 0x0000_003f	
Garfield_lighting.s1					0x0000_0060 - 0x0000_006f	
Garfield_GPIO.s1					0x0000_0070 - 0x0000_007f	
Drive_PWM.avalon_slave_0					0x0000_0080 - 0x0000_0087	
Steering_PWM.avalon_slave_0					0x0000_0090 - 0x0000_0097	
Rotary_Encoder_0.avalon_slav...					0x0000_00a0 - 0x0000_00a7	
mailbox_arm2nios_0.avmm_m...					0x0000_00c0 - 0x0000_00cf	
i2c_opencores_0.avalon_slav...					0x0000_8000 - 0x0000_801f	
sysid_fpga.control_slave	0x0001_0000 - 0x0001_0007		0x0001_0000 - 0x0001_0007		0x0001_0000 - 0x0001_0007	
mailbox_nios2arm_0.avmm_m...					0x0001_0030 - 0x0001_003f	
Onboard_LED.s1					0x0001_0050 - 0x0001_005f	
Onboard_DipSW.s1					0x0001_0080 - 0x0001_008f	
Onboard_Button.s1					0x0001_00c0 - 0x0001_00cf	
onchip_memory2_nios2.s1					0x0002_0000 - 0x0003_ffff	0x0002_0000 - 0x0003_ffff
nios2_gen2_0.debug_mem_sl...					0x0004_0800 - 0x0004_0fff	0x0004_0800 - 0x0004_0fff
jtag_uart_nios2.avalon_jtag_sl...					0x0004_1000 - 0x0004_1007	
shared_memory_mutex_mast...			0x0005_0000 - 0x0005_0007		0x0005_0000 - 0x0005_0007	
shared_memory_master_hps_...			0x0006_0000 - 0x0006_07ff		0x0006_0000 - 0x0006_07ff	
shared_memory_mutex_mast...			0x0008_0000 - 0x0008_0007		0x0008_0000 - 0x0008_0007	
shared_memory_master_nios_...			0x0009_0000 - 0x0009_07ff		0x0009_0000 - 0x0009_07ff	
hps_0.f2h_axi_slave				0x0000_0000 - 0xffff_ffff		
mailbox_arm2nios_0.avmm_m...			0x0002_0000 - 0x0002_000f			
mailbox_nios2arm_0.avmm_m...			0x0007_0000 - 0x0007_000f			

Abbildung 2.4.: Übersicht über die Addressen und Addressbereiche im System

Bit	Name	Access	Reset Value	Description
7 ... 0	control	RW	0	sets the dutycylce of the PWM signal generator
31 ... 7	-	R	0	not used

Tabelle 2.3.: Registermap des PWM Cores

Bit	Name	Access	Reset Value	Description
0	enable	RW	0	Enable bit for the core
1	clear	W	0	Clear bit. clears the result register and set it to 0; Must not be manually set to 0 after clearing. With the next rising edge of the clock it goes down on itself.
2	reset	W	0	Resets the whole core and set all values to default. At a read operation, it is always 0
15 ... 3	not accessable	-	0	-
16	error	R	0	Indicates an error within the counting process. You should reset the core!
31 ... 17	not accessable	-	0	-

Tabelle 2.5.: Registermap des Controlregisters des Rotary Encoder

Bit	Name	Access	Reset Value	Description
31 ... 0	result	R	0	Result of the counting process

Tabelle 2.7.: Registermap des Ergebnisregisters des Rotary Encoder

3. Software

Die Software unterteilt sich insgesamt in 3 Teile.

- Headquarter (HQ): Linux System das mit dem Fahrzeug über WLAN kommuniziert
- ARM: Linux ARM System, dass die Netzwerkaufgaben, sprich Kommunikation, mit dem HQ übernimmt
- NIOS: Softcore μController, der sonstige Periphere anspricht auf dem ein Echtzeitbestriebsystem (FreeRTOS) läuft. Die Software hiervon setzt sich zusammen aus /Software/common/ARM_NIOS_HQ/, /Software/common/ARM_NIOS und Software/Software_NIOS2/*.

3.1. HQ

Nachfolgend wird die Umsetzung der Headquarter-Software „Garfield Control“ beschrieben. Diese ermöglicht sowohl die Steuerung des Fahrzeuges, als auch die Visualisierung der vom Fahrzeug bzw. den angebrachten Sensoren erfassten Daten.

3.1.1. Funktionalitäten

Zur Herstellung der Verbindung zum Comm Gateway lassen sich die verwendete IP-Adresse und der Port im Einstellungsdialog festlegen. Auch die Verbindung zu einem Playstation 3 Controller lässt sich dort definieren. Durch Verlassen des Einstellungsdialoges mit dem OK-Button werden alle Einstellungen gespeichert und anschließend versucht eine Verbindung zum Controller herzustellen. Hat dies nicht geklappt, wird eine entsprechende Nachricht in der Statusleiste angezeigt. Durch Aktivieren der Debugausgaben lässt sich außerdem die Funktionalität der Steuerung mithilfe des Controllers testen. Im Anschluss lässt sich die Socketverbindung zum Comm Gateway durch Klicken des Connect-Buttons starten.

Ist eine Verbindung zum Comm Gateway hergestellt, lässt sich das Fahrzeug steuern und alle empfangenen Daten visualisieren. Dies wird nachfolgend kurz erläutert.

Die Steuerung des Fahrzeuges ist am komfortablesten durch Benutzung des laboreignen Playstation 3 Controller möglich. Zudem lässt es sich durch Benutzung der Tastatur oder der direkten Bedienung der GUI Elemente mit der Maus bedienen. Dabei sind folgende Befehle möglich:

- Die Geschwindigkeit in Fahrtrichtung (vorwärts oder rückwärts) lässt sich am Controller durch Betätigung von R2 (vorwärts) oder L2 (rückwärts) setzen. Dabei werden die vom Controller übermittelten Geschwindigkeitswerte in einen Wertebereich von 0 - 255 übersetzt und anschließend zusammen mit der Richtungsangabe übertragen. Durch Drücken der Taste W (vorwärts) oder S (rückwärts) auf der Tastatur oder einen Klick auf den entsprechenden Button der Anwendung lässt sich die maximale Geschwindigkeit ebenfalls setzen.
- Die Lenkung des Fahrzeuges lässt sich durch Bedienung des linken Sticks auf dem Controller steuern. Dabei werden Werte zwischen -90° und 90° versendet. Durch Verwendung der Tasten D oder A oder Klicken auf den entsprechenden Button lässt sich außerdem der maximale Lenkeinschlag setzen.
- Die am Fahrzeug angebrachte Beleuchtung lässt sich durch Betätigung des Triangle-Buttons, Klicken auf L oder einen Klick auf die entsprechende Checkbox in der Anwendung aktivieren oder deaktivieren.

Die Visualisierung der vom Fahrzeug übertragenen Daten umfasst neben der durch den rotary encoder gemessenen Geschwindigkeit, alle Daten der MPU6050, wie die Temperatur sowie Beschleunigung und Neigung des Fahrzeuges an allen 3 Achsen. Diese empfangenen Werte werden in der Applikation entsprechend dargestellt. Bis auf die Beschleunigung, welche durch einen sich der jeweiligen Achse anpassenden Punkt in einem Koordinatensystem dargestellt wird, werden alle Werte in Textausgabefeldern angezeigt.

3.1.2. Umsetzung

Die Anwendung Garfield Control wurde mithilfe von Qt für Linux entwickelt. Alle für die Socketverbindung gemeinsam genutzten Funktionen sind unter `Software/common/ARM_HQ` bzw. `Software/common/ARM_NIOS_HQ` abgelegt. Die eigenständigen Softwarebestandteile sind unter `Software/Software_HQ/Garfield_Control` vorhanden. Zur einfachen Benutzung der Anwendung wurden alle notwendigen Bibliotheken und Plugins mithilfe von linuxdeployqt zusammengefügt und als zip-Datei im Repository abgelegt. Somit lässt sich Garfield Control auch ohne eine Installation von Qt Paketen benutzen.
Die Verbindung zu einem Playstation 3 Controller wurde mithilfe einer API, welcher der Device Name übergeben wird, umgesetzt (Github Repository). Ist ein Controller mit der Anwendung verbunden, so wird dieser alle 20ms nach aufgetretenden Events abgefragt. Ebenfalls alle 20ms wird die Aktualisierung der Visualisierung der Beschleunigungswerte durchgeführt. Diese zyklischen Aufgaben werden unter zuhilfename von QTimern gestartet. Dazu lässt sich der Ablauf der Timer mit dem Aufruf der entsprechenden Methode verknüpfen.
Wurde eine Socketverbindung mit dem Comm Gateway hergestellt, so werden zwei seperate Threads gestartet, welche alle 20ms versuchen Informationsdaten zu empfangen, bzw. Steuerungsbefehle zu senden. Dazu wurden die gemeinsam verwendeten Klassen

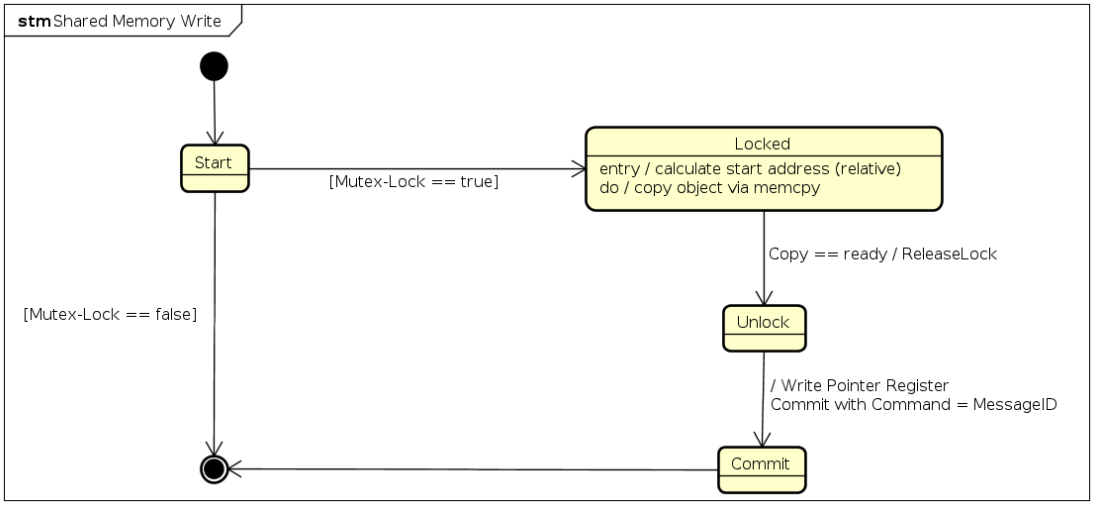


Abbildung 3.1.: Schreibvorgang in den Shared Memory

`Alf_Drive_Info` und `Alf_Drive_Command` benutzt, sodass sichergestellt ist, dass alle beteiligten Kommunikationspartner die gleiche Datenstruktur benutzen können.

3.2. ARM

3.2.1. Mailbox Kommunikation ARM ↔ NIOS2

Die Kommunikation über die in das FPGA programmierte Mailbox (siehe 2.3) wird über eine abstrakte Klassenimplementierung in C++ dargestellt. Die Klasse *Alf_ SharedMemoryComm* (zu finden unter Software/common/ARM_NIOS/) stellt einige *Write* und *Read* Funktionen zur Verfügung, die mit verschiedenen Objekten umgehen können. Durch dieses Vorgehen ist sichergestellt, das der Aufrufer als Übergabeparameter nur bestimmte Objekte (die sauber definiert sind) übergeben kann, die auch verarbeitet werden können. Die Kommunikation erfolgt asynchron und ist intern gepuffert. Die Klasse dient als Abstraktionsschicht in beide Richtungen. Sowohl die Empfangsrichtung als auch die Senderichtung werden über die Klasse abgebildet, sodass der Aufrufer keinerlei interne Informationen über die Hardware und die Implementierung haben muss. Nachfolgend werden die beiden wesentlichen Operationen (*Write* und *Read*) dargestellt und beschrieben.

Write Der schreibende Zugriff auf den Shared Memory ist in Abbildung 3.1 illustriert und hat einen einfachen Ablauf. Nachdem sich der Schreiber den Lock auf den Shared Memory über den Mutex Core geholt hat kann er Daten in diesen Bereich schreiben. Der Speicher wird dabei linear durchlaufen. Der Speicher wird dabei wie ein Ringspeicher behandelt. Ist am oberen Ende des Speichers nicht mehr genug Platz für die neue Nachricht wird wieder bei der relativen Addresse Null angefangen zu schreiben. Sind alle Daten in den Speicher geschrieben wird der Mutex wieder freigelassen. Im Speicher steht

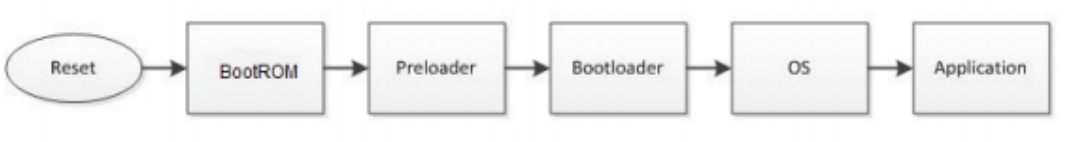


Abbildung 3.2.: Typsicher Bootvorgang des ARM A9 Dualcore Prozessors [4]

jetzt ein Abbild des Objekts, das übertragen werden soll. Es werden also nur Nutzdaten in den Speicher geschrieben. Anschließend wird die Startaddresse der Daten in das Pointer Register der Mailbox geschrieben. In das Command Register wird die eindeutige Nachrichten ID, die innerhalb des Garfield Projekts definiert ist, geschrieben. Dieser Schreibvorgang ist die letzte Instruktion zum Nachrichtenaustausch aus Sendersicht.

Read Der lesende Zugriff auf den Shared Memory ist komplizierter und unterteilt sich in zwei Abschnitte.

Der erste Teil der lesenden Kommunikation besteht aus dem **Interrupt**. Von dem Mailbox IP-Core wird bei einem schreibenden Zugriff auf das Command Register automatisch ein Interrupt erzeugt. Innerhalb des *ReadInterruptHandler* wird nicht der Shared Memory ausgelesen, sondern nur die Nachricht aus der Mailbox gespeichert. Die Klasse hält zusätzlich zu jedem Objekt, das verschickt bzw. empfangen werden soll einen kleinen Ringpuffer. Dieser dient dazu, die Addressen der Objekte im Shared Memory zu puffern. Tritt nun das Interrupt auf, wird zuerst das Pointer Register zwischengespeichert. Anschließend wird das Command Register, in dem der Nachrichtentyp übertragen wird, ausgelesen und anhand des Nachrichtentyps die Addresse des Pointer Registers in den passenden Puffer geschrieben. Dieses Vorgehen sorgt dafür, dass diese Funktion, die in einem Interrupthandler ausgeführt wird, sehr schnell wieder beendet ist.

Der zweite Teil besteht aus einem **Hintergrundtask** der zyklisch durchlaufen wird. Innerhalb dieses Task werden die Werte eines Objekts, die ausgelesen werden sollen, über eine *Read* Funktion bei Bedarf überschrieben. Bei Bedarf deswegen, weil die Werte nur überschrieben werden, wenn aktuellere Werte im Shared Memory stehen. Sind aktuelle Werte verfügbar, ist der Klasseninterne Ringpuffer, der die Addressen für die Nutzdaten enthält, nicht leer. Es wird über die Addresse aus dem Shared Memory gelesen und anschließend diese Nachricht (also die Addresse) aus dem Ringpuffer entfernt.

3.2.2. HPS Startvorgang und Software

Der folgende Abschnitt beschreibt die Bestandteile die für den Betrieb des ARM Dualcore notwendig sind und wie diese konfiguriert und kompiliert werden.

Eine sehr gute und übersichtliche Beschreibung des Bootvorgangs der A9 Kerne ist in [4] beschrieben. Der im GarfieldProjekt verwendeter Bootvorgang ist in Abbildung 3.2

dargestellt. Die ersten beiden Schritte (BootRom, Preloader) sollen hier nicht weiter beschrieben werden da keine manuelle Anpassung daran notwendig ist. Als Referenz für den Buildvorgang des gesamten Systems diente https://eewiki.net/display/linuxonarm/DE0-Nano-SoC+Kit. Eine komplette Kopie des Tutorials befindet sich auch im Projektverzeichnis unter *Software/Software_ARM/Linux/website/*. Wenn Änderungen an der im Tutorial beschriebenden Vorgehensweise notwendig sind werden diese erwähnt.

3.2.2.1. Bootloader

Als Bootloader kommt der beliebte U-Boot [8] in einer leicht angepassten Variante zum Einsatz. Wie im Tutorial beschrieben werden einige Startvariablen(unter anderem das zu ladende Linux Device Tree Binary) hinzugefügt um von der SD Karte zu booten. U-Boot lädt nach dem Start dann automatisch zunächst den

3.2.2.2. Linux Device Tree

Der Linux Device Tree ist ein Bestandteil des Linux Kernels um eine Abstraktionsschicht zwischen Hardware (Pinouts, Speicher, Interrupts) und dem Linux Kernel zu schaffen. Durch Einsatz des Device Tree kann ein gleiches Kernel Binary auf verschiedenen Hardwareversionen und sogar komplett verschiedenen Boards benutzt werden. Der Device Tree ist eine textuelle Beschreibung der Hardware die kompiliert wird und dem Kernel beim Startvorgang übergeben wird. Ein Auszug aus einem solchen Device Tree zeigt der Code 3.1. Die Spezifikation des Device Trees kann man unter http://www.devicetree.org/specifications/ einsehen.

Listing 3.1: Auszug aus socfpga.dtsi [6, Version 4.7, `arch/arm/boot/dts/socfpga.dtsi`]

```
cpus {
        #address-cells = <1>;
        #size-cells = <0>;
        enable-method = "altr,socfpga-smp";

        cpu@0 {
                compatible = "arm,cortex-a9";
                device_type = "cpu";
                reg = <0>;
                next-level-cache = <&L2>;
        };
        cpu@1 {
                compatible = "arm,cortex-a9";
                device_type = "cpu";
                reg = <1>;
                next-level-cache = <&L2>;
        };
```

[8] https://www.denx.de/wiki/U-Boot

```
};
```

Listing 3.2: Notwendige Änderungen am Device Tree

```
&fpga_bridge0 {
        bridge-enable = <1>;
};

&fpga_bridge1 {
        bridge-enable = <1>;
};

&fpga_bridge2 {
        bridge-enable = <1>;
};

/* we are extending the soc device with a specific interrupt! */
/{
        soc {
                mbox_rx: mailbox@0x00070000 {
                        compatible = "altr,mailbox-1.0";
                        reg = <0x70000 0x8>;
                        interrupt-parent = < &intc >;
                        interrupts = <GIC_SPI 60 4>;
                        #mbox-cells = <1>;
                };
        };

};
```

Die Änderungen, die für den Device Tree im GarfieldProjekt nötig sind, sind im Codeausschnitt 3.2 gezeigt. Es existiert außerdem eine patch-Datei (`Software/Software_ARM/socfpga_cyclone5_de0_sockit_garfield.patch`), mit der der geänderte Device Tree direkt in die heruntergeladenen Kernelsourcen gepatcht und kompiliert werden kann. Die vorgenommen Änderungen ergeben sich wie folgt

- fpga_brigdes - Durch die hinzugefügten Einträge `fpga_bridgeX` werden die verschieden verfügbaren (Achtung: Ist nur für Kernel Version 4.7 so gültig, ältere Kernel haben u.U. weniger verfügbare Bridges) Bridges beim Laden des Device Tree aktiviert.

- mbox_rx - Mit diesem Eintrag wird das Interrupt, ausgelöst durch den Mailbox IP-Core (siehe 2.3) als verfügbare Hardwareeinheit dem Kernel bekanntgemacht. Interessante Attribute sind
  - `@` - Die Zahl nach dem @ entspricht der Physikalischen Addresse der Mailbox.

Da die Adresse im weiteren Verlauf nicht verwendet wird ist der exakte Wert nicht von Bedeutung.

- `compatible` - Mit diesem Attribut wird dem Interrupt ein Name gegeben. Wäre im Kernel ein generischer Hardwaretreiber für diese Mailbox vorhanden könnte dieser zur Verarbeitung genutzt werden. Wichtig für das Projekt ist der Name trotzdem, da damit das Zuordnen der Interrupt-Service-Routine zu dem Interrupt geschieht.
- `interrupt-parent` - Damit wird der Interrupt Controller (in unserem Fall der ARM GIC, der in einer inkludierten Beschreibungsdatei beschrieben wird) identfiziert, der das Interrupt aufnimmt und die weitere Abarbeitung in die Wege leitet.
- `interrupts` - Damit wird das Interrupt, das am GIC ausgelöst wird (Nummer 60) bekannt gemacht. Die Nummer ergibt sich aus:
  - Das 60te Interrupt der *Shared Peripheral Interrupt* am GIC entspricht der 92ten Interruptleitung am GIC.
  - Die 92ten Interruptleitung entspricht der 20ten Interruptleitung, die vom FPGA verfügbar ist.

  [3].

Die Änderungen werden direkt in die Datei `LINUX_V4.7/arch/arm/boot/dts /socfpga_cyclone5_de0_sockit.dts` geschrieben.

3.2.2.3. Betriebssystem

Linux wird als Betriebssystem von U-Boot geladen. Im Projekt kommt eine leicht angepasste Version der im Tutorial beschriebenen Linux Version vor. Die Änderungen sind marginal und betreffen:

- Das USB-Subsystem - Zum Betrieb des Lidar[9] ist es notwendig einige Punkte des USB Subsystem während des Linux Kompiliervorgang (make menuconfig) auszuwählen und zu kompilieren. Im groben handelt es sich um die USB-OTG Unterstützung, das ACM Subsystem und noch einige kleinere Änderungen. Auch für diesen Schritt existiert eine patch Datei im Verzeichnis *Software/Software_ ARM/Linux/*.
- Das `uio` Modul - Mit diesem Modul ist es möglich mit Interrupts im User-Space von Linux zu arbeiten. Dazu wird das Modul mit dem Namen, der im Device Tree angegeben wurde, " altr ,mailbox−1.0", geladen. Dieses Modul erzeugt anschließend ein Gerät unter `/dev`, dass dann im User-Space genutzt werden kann. Auch diese Änderung ist in der patch Datei mit angegeben.

Als Distribution kommt eine minimale Version von Ubuntu zum Einsatz (Alternativ kann auch Debian verwendet werden). Diese Distribution wurde gewählt um einen kleine

[9] Light detection and ranging

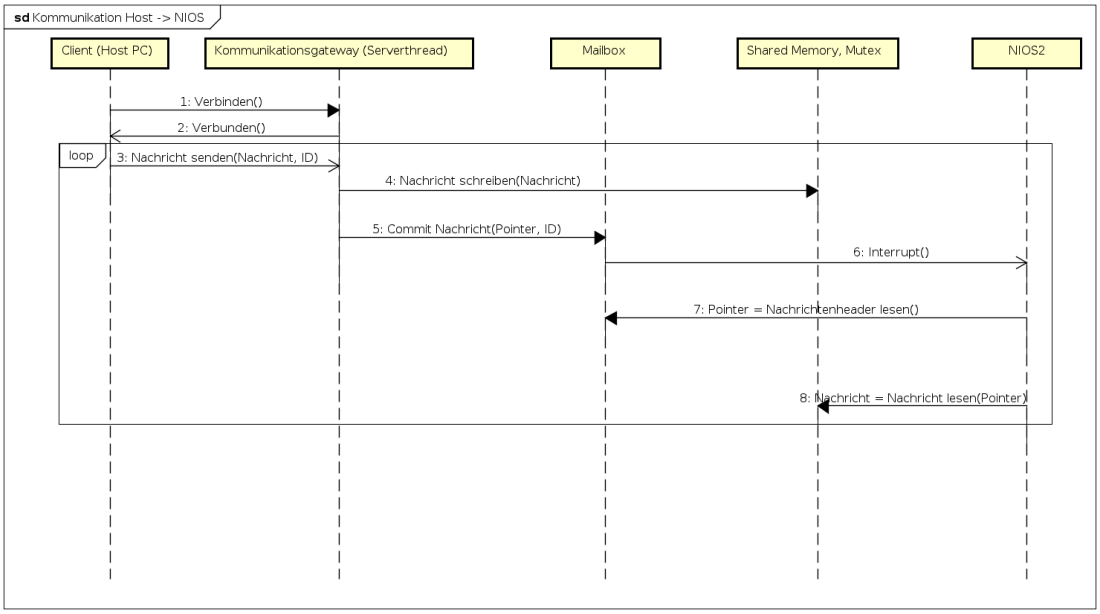


Abbildung 3.3.: Kommunikationsablauf zwischen Host PC und NIOS

Footprint (im Leerlauf insgesammt nur ca. 35MB Arbeitsspeicherverbrauch) mit dem Komfort einer "normalen" Linux Distribution inklusiver einem Softwarerepository zur einfachen Installation von Software zu verbinden. Sobald Linux gestartet ist kann die (hier) entscheidene Applikation gestartet werden.

3.2.2.4. Kommunikationsgateway

Das Kommunikationsgateway ist die Schnittstelle um zwischen einem Host-PC und dem NIOS2 kommunizieren zu können. Die Sourcedateien befinden sich unter `Software /Software_ARM/Gateway`. Dieses Programm wird später auch ausgeführt um Daten austauschen zu können. Neben der Initialisierung der Hardware, dem Öffnen eines Serverports, um sich mit dem Client zu verbinden und Daten auszutauschen, besteht dieses Programm aus insgesamt drei Threads:

- readData - Dieser Thread behandelt den Nachrichtenaustausch in Richtung Host PC → NIOS2. Der Thread wartet auf eingehenden Nachrichten (im Moment nur Fahrkommandos), parst und castet diese in das entsprechende Objekt und schreibt das Objekt dann über die Klasse *Alf_SharedMemoryComm* in den Shared Memory mit dem NIOS2. Dieser Kommunikationsweg ist auch in Abbildung 3.3 dargestellt.
- HardwareReadHandler - Dieser Thread ist alleine für die Abarbeitung der Interrupts im User-Space zuständig. Die Abarbeitung des Interrupts erfolgt nur durch Speicherung der Meta Informationen einer Nachricht (Nachrichten-ID und Pointer

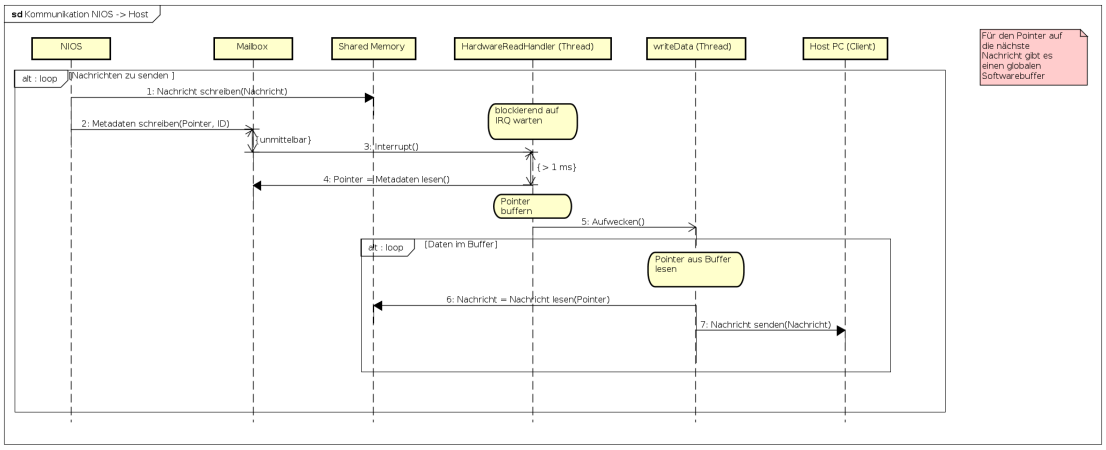


Abbildung 3.4.: Kommunikationsablauf zwischen NIOS und Host PC

im Shared Memory). Wurde eine gültige Nachricht verarbeitet wird der folgende Thread aktiviert.

- writeData - Dieser Task wird durch den HardwareReadHandler Thread bei einer gültigen Nachricht in einem Buffer aufgeweckt. Anschließend sendet der Thread die Nachricht über die Socketverbindung an den Client und blockiert abermals bis eine Nachricht eingeht. Abbildung 3.4 verdeutlicht den Nachrichtaustausch mit der Kommunikationsrichtung NIOS2 → Host PC.

3.2.3. Empfohlener Buildvorgang und Abweichungen zum Tutorial

Der schnellste und unkomplizierteste Weg zu einem funktionierenden Linux Image auf SD-Karte ist:

1. Das Tutorial unter https://eewiki.net/display/linuxonarm/DE0-Nano-SoC+Kit mit einer beliebigen (Mindestgröße 4GB) SD Karte durchführen. Als Kernelversion sollte unbedingt die Version 4.7 gewählt werden, da sich die im Projektrepository befindlichen Dateien darauf beziehen. Nach diesem Schritt kann man den Bootvorgang von U-Boot und Linux testen und versuchen sich im Ubuntu einzuloggen. Anschließend werden die Änderungen durchgeführt.

2. Den Linux Devcie Tree patchen. Dazu in einem Linux-Terminal

```
patch socfpga-kernel-dev/KERNEL/arch/arm/boot/dts/
    socfpga_cyclone5_de0_sockit.dts
    socfpga_cyclone5_de0_sockit_garfield.patch #dieses
    Verzeichnis ist nach dem Tutorial verfuegbar
```

3. Das Linux Konfigurationsfile patchen

```
patch socfpga-kernel-dev/KERNEL/.config Garfield_Kernel.
    patch)
```

4. Anschließend über das Skript `TutorialPath/socfpga-kernel-dev/tools/rebuild.sh` (oder manuell über den entsprechenden make-Befehl) den Kernel inkl. Device Tree neu kompilieren

5. Den Linux Kernel und die Device Trees wie im Tutorial (die Punkte `Copy Kernel Image` und `Copy Kernel Device Tree Binaries`) auf die SD Karte kopieren.

Anschließend kann man die SD-Karte wieder entnehmen und das FPGA wieder einschalten. Das System sollte wie gewohnt starten. Um zu überprüfen, ob die Operationen funktioniert haben sollten folgenden Befehle die gleiche Ausgabe wie in Abb. 3.5a und 3.5b haben. Für die notwendigen Schritte befindet sich im Home-Verzeichnis des Standardbenutzers außerdem ein Skript das ausgeführt werden kann und die notwendigen Befehle ausführt.

3.2.4. FPGA Programmierung

Die Programmierung des FPGA Subsystems erfolgt unmittelbar während des Systemstarts und ist aktuell vom Softwarestart abgekoppelt. Das Image für das FPGA wird dabei aus dem auf dem Board integrierten Flash-Speicher geladen. Die möglichen alternativen Konfigurationen sind übersichtlich im User-Manual zu dem `DE0-Nano` Board aufgelistet. Das Manual dazu findet man auf der System CD, die unter Link verfügbar ist.

Auf dem Flash befindet sich sowohl das Image für das FPGA als auch das executable für den nach dem flashen im FPGA verfügbaren NIOS2.

3.2.5. Applikationsstart

Für die Kommunikation zwischen Host PC und NIOS2 ist das starten des Kommunikationsgateway erforderlich. Dies befindet sich unter `HPS/bin/Comm_Gateway` und ist das Kompilat des oben beschriebenen HSP_ARM_Gateway. Damit das Programm Ordnungsgemäß funktionieren kann müssen noch einige Schritte durchgeführt werden, die hier kurz erklärt werden.

Listing 3.3: Listing

```
sudo -s # superuserrechte erlangen, Passwort = temppwd
rmmod uio_pdrv_genirq
rmmod uio
modprobe uio_pdrv_genirq of_id="altr,mailbox-1.0"
ls /dev/
bin/Comm_Gateway
```

```
root@arm:~# cat /proc/interrupts
           CPU0       CPU1
 16:      55559      65430     GIC-0  29 Edge      twd
 17:          0          0     GIC-0 199 Level     timer0
 26:          0          0     GIC-0 207 Level     ff706000.fpga-mgr
 27:        333          0     GIC-0 152 Level     eth0
 28:          0          0     GIC-0 190 Level     ffc04000.i2c
 29:          0          0     GIC-0 191 Level     ffc05000.i2c
 31:      26538          0     GIC-0 171 Level     dw-mci
 36:        986          0     GIC-0 194 Level     serial
 38:          0          0     GIC-0 160 Level     ffb40000.usb, ffb40000.usb, dwc2_hsotg:usb1
 40:          0          0     GIC-0  92 Level     mailbox
IPI0:         0          0  CPU wakeup interrupts
IPI1:         0          0  Timer broadcast interrupts
IPI2:      2042      24164  Rescheduling interrupts
IPI3:         4          3  Function call interrupts
IPI4:         0          0  CPU stop interrupts
IPI5:         0          0  IRQ work interrupts
IPI6:         0          0  completion interrupts
Err:          0
```

(a) Eintrag mailbox interrupt

```
root@arm:~# ls /dev/
autofs            mmcblk0           ptype   tty21  tty43  tty8     usbmon1
block             mmcblk0p1         ptypf   tty22  tty44  tty9     vcs
btrfs-control     mmcblk0p2         ram0    tty23  tty45  ttyp0    vcs1
bus               network_latency   ram1    tty24  tty46  ttyp1    vcs2
char              network_throughput random  tty25  tty47  ttyp2    vcs3
console           null              shm     tty26  tty48  ttyp3    vcs4
cpu_dma_latency   psaux             stderr  tty27  tty49  ttyp4    vcs5
cuse              ptmx              stdin   tty28  tty5   ttyp5    vcs6
disk              ptp0              stdout  tty29  tty50  ttyp6    vcsa
fd                pts               tty     tty3   tty51  ttyp7    vcsa1
full              ptyp0             tty0    tty30  tty52  ttyp8    vcsa2
fuse              ptyp1             tty1    tty31  tty53  ttyp9    vcsa3
gpiochip0         ptyp2             tty10   tty32  tty54  ttypa    vcsa4
gpiochip1         ptyp3             tty11   tty33  tty55  ttypb    vcsa5
gpiochip2         ptyp4             tty12   tty34  tty56  ttypc    vcsa6
i2c-0             ptyp5             tty13   tty35  tty57  ttypd    watchdog
i2c-1             ptyp6             tty14   tty36  tty58  ttype    watchdog0
initctl           ptyp7             tty15   tty37  tty59  ttypf    zero
input             ptyp8             tty16   tty38  tty6   ttyS0
kmem              ptyp9             tty17   tty39  tty60  ttyS1
kmsg              ptypa             tty18   tty4   tty61  uinput
log               ptypb             tty19   tty40  tty62  uio0
mem               ptypc             tty2    tty41  tty63  urandom
memory_bandwidth  ptypd             tty20   tty42  tty7   usbmon0
root@arm:~#
```

(b) Eintrag in /dev/

Abbildung 3.5.: Einträge zur Überprüfung der Funktionsweise der Mailbox

Der Codeausschnit 3.3 zeigt alle durchzuführenden Schritte, die notwendig sind um das Kommunikationsgatway ordnungsgemäß zu starten. Die ersten beide Befehle entfernen evtl. geladenene Module, die beim startup mit falschen Parametern geladen werden. Der dritte Befehl lädt das Linux-Modul um mit Interrupts im User-Space umgehen zu können und erstellt dafür ein Gerät mit dem Namen `HSP_boot/dev/uio0`, dass im Kommunikationsgatway verwendet wird. Hier ist auch der Name, der im Linux Device Tree angegeben wurde, von Bedeutung, da über diesen das richtige Interrupt ausgewählt wird. Mit dem vierten Befehl sollte vor Ausführung des Gateways überprüft werden ob das Gerät `HSP_boot/dev/uio0` auch wirklich verfügbar ist. Ist dies der Fall, kann die Applikation mit Superuserrechten gestartet werden. Die Reihenfolge ist dabei

1. Starten des FreeRTOS auf dem NIOS2 (wird i.d.R. automatisch bei Systemstart durch Laden aus dem externen Flash durchgeführt).
2. Konfigurieren des Linux und starten der Applikation (eine Serververbindung wird angeboten).
3. Anschließendes Verbinden der Applikation auf dem Host-PC mit dem Server. Daten werden ab diesem Zeitpunkt ausgetauscht.

3.2.6. Beenden der Applikation

Aktuell ist keinerlei Fehlerbehandlung für das Beenden der Client-Server Verbindung (Host-PC $\leftrightarrow$ HPS) vorhanden. Beendet der Client die Kommunikation hängt sich das gesamte System auf (Grund: Interrupts werden nicht mehr angenommen und der interne Bus wird durch die Mailbox blockiert!). Es ist also unbedingt notwendig die Applikation auf dem laufenden Linux mittels eines SIGINT (STRG+C) zu beenden. Alternativ kann man dem Programm auch auf der Kommandozeile oder durch z.B. htop das Signal schicken. Mit dem Signal sendet Linux einen speziellen Befehl an den NIOS2 wodurch dieser die Kommunikation auch beendet. Werden wieder Befehle geschickt, wird die Kommunikation von beiden Seiten wieder aufgenommen.

3.3. μController - NIOS2

3.3.1. Operating System - FreeRTOS

Dieses Kapitel beschreibt wie das verwendete Echtzeitbetriebsystem zu benutzen ist. Ein FreeRTOS Port für den NIOS2 in der Version 9.0.0 wird derzeit eingesetzt. Dabei wurden nur minimale Änderungen an der Standardkonfiguration vorgenommen, die allerdings jederzeit erweiterbar ist, sofern das durch die Applikation nötig ist. Die Website von FreeRTOS mit Download und kompletter Spezifikation ist unter http://www.freertos.org/ zu finden.

3.3.1.1. Tasks

Die verwendeten Tasks auf der NIOS2 Plattform sind unter /Software/Software_NIOS2/tasks/ in den tasks_nios Dateien zu finden. Folgende Tasks sind dabei im Moment vorhanden:

- readUltraSonic: Dieser Task ist dafür zuständig die vier verwendeten und über IIC angeschlossenen (vorne zwei, hinten zwei) SRF08 Ultraschallsensoren auszulesen. Im Moment werden nur wenige der möglichen Ultraschalldaten verwendet, die dann dafür sorgen, dass im setMotor_and_Steering Task langsamer (wenn im Nahbereich eines Hindernisses) gefahren bzw. komplett stehen geblieben wird (wenn direkt vor dem Hinderniss). Ein Ultraschallmodul kann insgesamt 17 verschieden weit entfernte Hindernisse erkennen. Im Moment wird nur das erste und damit das nächste Hinderniss beachtet. Falls die Ultraschalldaten auch zur Positionsbestimmung mitverwendet werden sollen, kann es hilfreich sein auch die restlichen Informationen der Ultraschallmodule zu verwenden. Dabei sollte dann allerdings die IIC Kommunikation genau überprüft werden. Diese kann unter Umständen länger als ein FreeRTOS Tick dauern, was eventuell zu Problemen führen kann. Das FreeRTOS stellt allerdings Möglichkeiten bereit, den Scheduler anzuhalten bzw. einzelne Bereiche atomar auszuführen. Eine weitere Möglichkeit ist die Aufteilung des abfragens der einzelnen Ultraschallsensoren in mehrere Tasks. An dieser Stelle soll nur auf ein solches Problem hingewiesen werden, die konkrete Umsetzung bleibt den Gruppen selbst überlassen. Die Parameter für die Einstellungen der Entfernungen (Nahbereich bzw. Notstop) können durch Anpassung der Konfigurationsparameter in der tasks_nios.cpp angepasst werden (auch in der Doxygen zu finden).

- readMPU: Dieser Task hat die Aufgabe den mpu6050 Sensor auszulesen. Dabei werden die ausgelesenen Werte im Moment nur grafisch angezeigt. An anderer Stelle werden diese Informationen nicht benutzt. Können eventuell für eine spätere Positionsbestimmung bzw. Aufzeichung einer Wegstrecke verwendet werden.

- readRotary: Dieser Task liest den RotaryEncoder aus, der auf dem Fahrzeug montiert ist. Auch diese Daten werden im Moment nur für die grafische Oberfläche benutzt. Auch diese Daten können später dafür verwendet werden, um eine Wegstrecke zu ermitteln und dann damit eine genaue Position auszurechnen.

- setMotor_and_Steering: Dieser Task ist für das Setzen des Lenkwinkels und des Motor PWM Signals zuständig. Dabei werden die Informationen aus den Ultraschallmodulen verwendet, um auf Hindernisse reagieren zu können. Ansonsten werden alle Daten, die aus dem HQ auf den ARM und damit in den Shared Memory Bereich kommen ausgelesen und als Geschwindigkeit und Lenkwinkel gesetzt. Dabei wird auch ein Timeout Mechanismus verwendet, der dafür sorgt, falls keine neuen Daten mehr kommen, weil die z. B. die Verbindung zwischen HQ und Fahrzeug abgebrochen ist, das Fahrzeug langsamer wird und schlussendlich auch stehen bleibt. Sobald die Verbindung wiederhergestellt ist kann sofort ohne Verzögerung weitergefahren werden. Ein manueller Reset oder ähnliches ist nicht notwendig.

Tabelle 3.1.: Taskkonfiguration

Taskname	*Priorität*	*Zykluszeit*	*verwendete Variablen*
readUltraSonic	3	75 ms	write: global_us_front_left_data write: global_us_front_right_data write: global_us_rear_left_data write: global_us_rear_right_data
readMPU	2	50 ms	write: global_acc_data write: global_gyro_data write: global_temp_data write: global_drive_info
readRotary	2	50 ms	write: global_drive_info
setMotor_and_Steering	3	20 ms	read: sharedMem(Alf_Drive_Command)
setDriveInfo	1	200 ms	write: sharedMem(global_drive_info)

- setDriveInfo: Dieser Task schreibt alle gesammelten Fahrinfos (mpu6050 Daten + Geschwindigkeit) in den Shared Memory Bereich zwischen ARM und NIOS, die dann vom ARM Linux System ausgelesen werden können.

Die komplette Taskkonfiguration ist in Tabelle 3.1 zu sehen. Dabei ist die Priorität im FreeRTOS von 0 zur höchsten (konfigurierbaren) Priorität festgelegt. Das bedeutet das 0 die kleinste Priorität ist, die z. B. auch der Idle Task vom FreeRTOS Kernel besitzt, der aufgerufen wird, wenn kein anderer Task im ready Status ist. Die Zykluszeit gibt an, nach wie vielen 1/1000 Sekunden der Task wieder aufgerufen wird. Dabei ist das ganze nur mit geringer Abweichung fest konfigurierbar. Das heißt ein eingestellter Wert von 20ms wird meist nie genau getroffen - diese geringen Abweichungen stellen im Moment allerdings kein Problem dar. Die verwendeten Variablen geben an auf welche in der tasks_nios.cpp definierten Variablen der Task schreibend und lesend zugreift. Die komplette Task Kommunikation, die sich auf ein Minimum beschränkt wird aktuell über lokale bzw. teilweise über globale Variablen erledigt. Das Fahren und die Ultraschallsensoren sind aktuell am wichtigsten da auf diese Ereignisse sofort reagiert werden soll. Alle anderen Bereiche haben eine geringere Priorität. Die Ultraschallmodule können in der Standardeinstellung (6 Meter Reichweite) alle 65ms ein neues Ergebnis liefern. Da allerdings die Genauigkeit des FreeRTOS Zyklus nicht so genau war, wurde hier ein leicht größerer Wert gewählt. Würde ein geringeren Wert gewählt müssen zuerst die Ultraschallsensoren abgefragt werden, ob diese bereits mit der Messung fertig sind. Da dieses Abfragen auf keinen Fall passieren soll, der IIC sollte so wenig wie möglich am Stück benutzt werden, da nur ein IIC Modul vorhanden ist und auch der mpu6050 über den selben Bus angeschlossen ist. Somit könnten verschiedene IIC Transmissionen kollidieren, wenn ein Task länger auf den Bus zugreift.

Ein Task kann mittels xTaskCreate() erzeugt werden. Der Scheduler wird dann mit dem Befehl vTaskStartScheduler() gestartet. Von diesem Zeitpunkt an werden alle vorher erzeugten Task zyklich aufgerufen. Eine genauere Dokumentation zu den jeweiligen APIs des FreeRTOS ist im Repository unter /Datasheets im Reference Manual zu finden. Das

zyklische Aufrufen eines Tasks ist mit verschiedenen Methoden möglich. Die in diesem Projekt verwendete Methode wird nun kurz in 3.4 aufgezeigt.

Listing 3.4: Taskzyklus erzeugen

```
TickType_t xLastWakeTime;
// bestimmt die Zyklusfrequenz des Tasks in
const TickType_t xFrequency = 20;

while(1)
{
  // hier wird gewartet bis der Zyklus vollstaendig ist
  vTaskDelayUntil( &xLastWakeTime, xFrequency );
  // alle Taskoperationen koennen hier ausgefuehrt werden
}
```

3.3.1.2. RTOS Config

Hier wird die genaue RTOS Konfiguration beschrieben, die für das OS auf dem NIOS2 benutzt wurde. Die Konfiguration ist in /Software/Software_NIOS2/os/ in der FreeRTOSConfig.h zu finden. Die einzelnen Parameter werden im Reference Manual genauer erläutert. Hier werden nur die wichtigsten verwendeten Parameter erklärt.

- configUSE_PREEMPTION: wenn auf 1 gesetzt, ermöglicht das dem Scheduler das unterbrechen der Tasks wenn zu einem FreeRTOS Tick ein höherpriorer Task als ready markiert ist, als der aktuelle. Wenn auf 0 gesetzt kann der Task nur durch sich selbst beendet/unterbochen werden (z.B. ein Zyklus fertig, warten auf den nächsten ready Status). Da das Fahren und der Ultraschall jeweils sehr wichtig sind, wurde dieses Verfahren gewählt.

- configTICK_RATE_HZ: definiert die FreeRTOS Tick Rate. Ein Wert von 200Hz bedeutet das 200mal in der Sekunde ein FreeRTOS Tick auftritt an denen der Scheduler aktiv wird und einen Kontextwechsel einleitet bzw. den nächsten Task aufruft, der bereit ist.

- configCPU_CLOCK_HZ: Hier wird die Rate angegeben, mit der der interne Systemtimer (aktuell: TIMER_0_NIOS2_BASE) arbeitet den das FreeRTOS benutzt. Muss nicht der CPU Frequenz entsprechen.

- configIDLE_SHOULD_YIELD: wenn auf 1, wird der Idle Task, den das FreeRTOS automatisch implementiert sofort wieder durch den Scheduler unterbrochen und der nächste Task, der ready ist kann sofort aufgerufen werden. Andernfalls bleibt der Idle Task mindestens einen kompletten FreeRTOS Tick lang aktiv, was dazu führt das alle Tasks insgesamt eine gerechtere Aktivzeit erhalten. Da dies in unserem Fall nicht nötig ist, wurde dieser Parameter auf 1 gesetzt.

ZU BEACHTEN:
Um das FreeRTOS auf dem NIOS2 Core zu benutzen sind weiterhin zwei Dinge zu beachten. Da bei jedem Erstellen eines BSPs in der system.h das Makro
ALT_ENHANCED_INTERRUPT_API_PRESENT
definiert wird, der NIOS2 Port damit aber nicht umgehen kann, muss dieses Makro durch
ALT_LEGACY_INTERRUPT_API_PRESENT
ersetzt werden, ansonsten meldet der Linker undefinierte Verweise. Des Weiteren ist unter /Software/Software_NIOS2/os/Source/portable/ die port.c zu finden. In dieser wird der Timer definiert, der dafür zuständig ist die FreeRTOS ticks zu erzeugen, die z. B. auch der Scheduler benutzt. Sollte dieser Systemtimer geändert werden, aus welchen Gründen auch immer, muss dieses File mit angepasst werden und alle vorkommenden TIMER_0_NIOS2_BASE Einträge auf den neuen Timer angepasst werden.

3.3.2. NIOS2 - Treiber / Hardware Abstraction

Hier wird kurz der Aufbau der HAL bzw. der Aufbau der jetzt zur Verfügung stehenden Treiber des NIOS2 Cores erläutert. Genauere Dokumentation ist in der Doxygen Dokumentation zu finden, die im Anhang mitgeliefert wird.

3.3.2.1. Display

Das Display welches in diesem Projekt verwendet wurde, besitzt neben dem graphischen Display einen Mikro SD Karten Slot und einen Touchscreen, welche über SPI angesprochen werden. Letztere werden zum aktuellen Zeitpunkt vom Display Treiber nicht unterstützt. Der Display Treiber basiert auf der Adafruit GFX Library und läuft zum aktuellen Zeitpunkt mit der maximalen Geschwindigkeit von 24 MHz. Bevor das Display benutzt werden kann, muss es initialisiert werden. Im Anschluss lässt sich eine Zeile auf das Display schreiben. Der Treiber fügt an den Beginn der Zeile die aktuelle Zeilennummer hinzu. Ist das Display bereits durch vorherige Zeilen gefüllt, so wird automatisch von oben neu begonnen. Der Methode zum Schreiben einer Zeile muss somit lediglich die Farbe und Größe übergeben werden.

3.3.2.2. Motor

Die Klasse Drive ist für die Ansteuerung des Motors zuständig. Dabei ist es möglich die maximale Geschwindigkeit im Bereich zwischen 0% und 100% zu begrenzen. Eine weitere Funktion ermöglicht das Setzen der Richtung und der Geschwindigkeit, das dann in ein PWM Signal für den Motor umgerechnet wird.

3.3.2.3. Lenkung

Die Lenkung des Fahrzeuges wird mithilfe der Klasse Steering ermöglicht. Der verwendete Servo Blue Bird BMS-630MG besitzt einen maximalen Winkel von ±60°, welcher sich durch die Init() Funktion begrenzen lässt. Der Servo arbeitet mit einer PWM Frequenz von 125 Hz und erreicht seine maximale Winkel bei einer Periodendauer von 900 bzw.

2100 μs (Siehe Datenblatt im Repository für weitere Details). Die Funktion Set() setzt dann den tatsächlichen Winkel.

3.3.2.4. MPU6050

Das mpu6050 Modul ist über den IIC Bus angeschlossen und kann die 3 Beschleunigungsachsen, 3 Drehachsen und die Temperatur auslesen. Das Modul muss vor der Verwendung einmalig initalisiert werden. Dabei ist zu beachten, dass der AD0 Pin die IIC Adresse des mpu6050 Bausteines hardwaremäßig verändert.

3.3.2.5. Ultraschall

Die vier zur Verfügung stehenden Ultraschallsensoren sind über den gleichen IIC Bus, wie das MPU6050 Modul angebunden. Diese Geräte benötigen keine Initialisierung, d.h. sind sofort nach dem Anschließen an den Bus einsatzbereit. Allerdings ist zu beachten, dass nur Geräte an den Bus angeschlossen werden, die unterschiedliche IIC Adressen haben (Funktion zur Änderung der IIC Adresse ist vorhanden), da es sonst zu undefiniertem Verhalten auf dem Bus kommt. Für das ändern der Address sollte nur ein Gerät angeschlossen sein.

4. Hilfreiche Verweise

Nachfolgend werden einige Links inklusiver kurzer Beschreibung aufgezählt, die während der Projekts eine hilfreiche Anlaufstelle für Problemlösungen waren.

- Ubuntu on DE0 - Das im Projekt verwendete Tutorial um eine Ubuntu Distribution auf dem SoC[10] lauffähig zu machen.
- Multicore - Ein Multicore Beispiel für mehrere NIOS2 Prozessoren und die Inter Prozess Kommunikation
- Multiprozessor I - Ein Beispiel wie man ein Multicoresystem für den Cyclone V mit den HPS und mindestens einem NIOS2 aufbauen kann.
- Multiprozessor II - Ein Beispiel für die Kommunikation zwischen mehreren Prozessoren.
- Device Tree for Dummies
- Interrupts - Ein Beispiel wie man auf dem FPGA Interrupts für den HPS erzeugt und im Linux nutzt.
- Wie man Linux Interrupts im Userspace verarbeitet

[10] System-on-a-Chip

5. Probleme und zukünftige Arbeitspakete

5.1. Aktuelle Probleme

Zum Zeitpunkt der Ab-/Übergabe des Garfield Projekts bestehen noch einige kleine und umfangreichere Probleme. Manche dieser Probleme sollten mit einer hohen Priorität behandelt werden, andere Probleme stören nur die Bedienbarkeit und haben deswegen eine niedrigere Priorität.

5.1.1. Spannungs- und Stromversorgung

Dies ist ein Problem, dass mit höherer Prioriät, bevorzugt am Anfang des nächsten Projekts, behandelt werden sollte. Im wesentlichen gibt es zwei größere Probleme:

- Immer wieder kehrende Spannungseinbrüche nach Starten des Motors.
- Eine nicht ausreichende Spannungsversorgung durch kleine RC-Akkus mit einer Sollspannung von $7,2$ Volt.

Für die Spannungseinbrüche konnte bis jetzt wieder eine sinnvolle Erklärung noch ein Lösungsvorschlag erreicht werden. Die Einbrüche treten sowohl an der Versorgung durch ein Labornetzteil auf (wobei nicht jedes Labornetzteil die gleichen Symptome zeigt) als auch bei Speisung durch einen großen RC Akku mit 12V Sollspannung auf. An einem Labornetzteil treten die Einbrüche zwar wesentlich weniger oft auf, aber auch dort treten sie nicht-deterministisch auf. Die Fehler treten auch wesentlich öfter auf als dies im Vorgängerprojekt mit einem Rasperry Pi der Fall war. Die aktuelle Lösung braucht zwar etwas mehr Leistung (zwischen 0.5 und 1 Watt), aber sowohl der RC Akku als auch ein Labornetzteil können normalerweise mit solchen Problemen umgehen

Das Problem mit den kleinen RC Akkus (für die sich eine Akkuhalterung auf dem Auto befindet) kann sich ebenfalls nicht erklärt werden. Die $7,2V$ werden nur zur Versorung des Motors benötigt, ansonsten sind nur 5V und $3.3V$ notwendig. Diese beiden Spannungen werden aber über Konstantspannungskonverter erzeugt, die beide relativ unempfindlich gegen höhere, auch schwankende Spannungen sind.

Als erster Ansatzpunkt kann nur ein Hinweise/Lösungsstrategie empfohlen werden: Die Platine, die immer wieder wechselnden Bedürfnissen angepasst wurde, sollte überprüft werden. Hier ist die Unvoreingenommenheit einer anderen Bearbeitungsgruppe mit

Sicherheit von großem Vorteil. Die Nachfolgegruppe kann alle Verbindungen überprüfen und, obwohl ein Schaltplan vorhanden ist, über jede Verbindung nachdenken und überprüfen ob die Beschaltung so Sinn machen wie sie aktuell gelöst sind. An diesem Punkt kann man auch nochmal über Stützkondensatoren an relevanten Stellen nachdenken. Unter Umständen ist es dann auch hilfreich eine neue Platine zu erstellen.

5.1.2. Motortreiber

Der Motortreiber hat während der Tests immer wieder sporadische Aussetzer gehabt und funktionierte nach einem Neustart manchmal nicht richtig. Als Übergangslösung wurde an dem Motortreiber ein Reset-Button angebracht, der, sollte der Motortreiber nicht mehr funktionieren, gedrückt werden kann und alle Fehlerzustände des Motortreibers löscht und ihn wieder funktionieren lässt. Die Ursache für diese Fehler ist nicht offensichtlich, aber eventuell ist ein Zusammenhang mit den in 5.1.1 beschriebenen Fehlern zu beobachten.
Eine mögliche Lösung wäre die Fehlersignale und das Resetsignal mit dem FPGA zu verbinden und im NIOS2 zu verarbeiten. Dann kann eine saubere Fehlerbehandlung eine Resetauslösung durch einen μController erfolgen.

5.1.3. Sporadische Fehler in der Nachrichtenübertragung

In der Nachrichtenübertragung zwischen Host-PC und NIOS2 treten sporadische Fehler auf. Diese äußern sich in Nachrichtenpaketen (einzelne Werte, i.d.R. sind nicht alle Werte falsch), die den jeweiligen Maximalwert annehmen(z.B. uint8_t x = 255 anstatt uint8_t x = 0). Für diesen Fehlerfall konnten noch keine weiteren Beobachtungen gemacht werden, insbesondere nicht woher die falschen Nachrichten kommen (Entweder vom Host PC als Initiator der Nachricht oder vom Kommunikationsgateway das für das kopieren von Nachrichten in den Speicher verantwortlich ist). Die falschen Werte treten in sehr unregelmäßigen Abständen auf, es ist auch möglich das über mehrere Sitzungen hinweg kein einziger solcher Fehler auftritt. An anderen Tagen tritt dieser Fehler wiederrum regelmäßig im 10-Sekundentakt auf.

5.2. Mögliche Arbeitspakete

Nachfolgend sollen alle denkbaren Arbeitspakete aufgelistet werden, welche durch eine nachfolgende Gruppe behandelt werden können.

- Die Behebung aller oben beschriebenen Probleme, bildet ein Arbeitspaket für nachfolgende Gruppen und verhilft dem System zu einem noch reibungsloseren Arbeiten.
- Durch die leistungsstärkere Hardware wird es nun möglich, die Berechnung eines SLAM Algorithmus auf dem Fahrzeug durchzuführen, ohne an die Grenzen der Hardware zu stoßen. Damit wäre ein weiterer wichtiger Schritt zur Durchführung einer reelen autonomen Fahrt gemacht.

- Die weitere Optimierung des Systems bildet ein weiteres Arbeitspaket. Dabei ist eine Optimierung im Bereich der Kommunikation denkbar. Diese lässt sich beispielsweise komplett auf Basis von Interrupts durchführen, wodurch sich weitere Performancevorteile ergeben. Auch die Kommunikation über die Mailbox zwischen ARM und NIOS ließe sich weiter optimieren oder mit weiteren Funktionalitäten erweitern.
- Im Bereich des zukünftigen Einsatzgebietes - der autonomen Fahrt, lässt sich das System mit weiteren Features ausstatten, denkbar ist hier beispielsweise eine Kamera zur Wegerkennung, oder weitere Sensoren.

6. Zusammenfassung

Im Rahmen dieses HSP Projekts konnten die anfangs gesteckten Ziele weitgehend erreicht werden. Durch Ersetzen des Raspberry Pi durch eine flexible (FPGA) und leistungsstarke (ARM A9 Dualcore) Lösung, wurde eine Möglichkeit geschaffen sowohl Echtzeitbedingungen als auch low level IO Operationen (I2C, SPI, PWM etc.) von dem Hauptprozessoren zu entfernen. Durch diesen Schritt sind auf dem Hauptprozessor genug Leisutungsreserven frei um in zukünftigen Anwendungen Berechnungen, wie z.B. zur Berechnung von SLAM Algorithmen, direkt auf den Hauptprozessoren durchzuführen. Gleichzeitig hat das FPGA noch genug freie Logik zur Verfügung, um Teile von SLAM Algorithmen u.U. direkt auf Hardware rechnen zu können. Gleichzeitig konnten alle Funktionalitäten der Ausgangsbasis erhalten werden, wodurch dieses Projekt eine gute Grundlage für die Bearbeitung durch weitere Gruppen bildet.

Abbildungsverzeichnis

Abkürzungsverzeichnis

ALF	Autonomes Laser Fahrzeug
HAL	Hardware Abstraction Layer
HSP	Hauptseminar Projektstudium
Lidar	Light detection and ranging
ROS	Robot Operating System
RVIZ	ROS Visualization
RISC	Reduced Instruction Set Computer
SoC	System-on-a-Chip
FPGA	Field Programmable Gate Array
IP	Intellectual Property
HPS	Hard Processor System
PLL	Phase-locked loop

Literaturverzeichnis

[1] Altera. *Using the ARM Generic Interrupt Controller*. Application Note. Intel. URL: ftp://ftp.altera.com/up/pub/Altera_Material/14.0/Tutorials/Using_GIC.pdf.

[2] Altera/Intel. *SW Development for Altera SoC Devices Workshop*. 2016. URL: https://rocketboards.org/foswiki/pub/Documentation/WS1IntroToAlteraSoCDevices/WS_1_Intro_To_SoC_SW_Workshop.pdf.

[3] Philemon Favrod. "How to configure Linux to receive IRQs from the FPGA?" HowTo. Eidgenössische Technische Hochschule Lausanne. URL: https://wiki.epfl.ch/prsoc/documents/Cyclone_V_SoC_Linux_Interrupt-2.pdf.

[4] Altera Inc. *HPS SoC Boot Guide - Cyclone V SoC Development Kit*. 2016. URL: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/an/an709.pdf.

[5] Intel. *Embedded Peripherals IP User Guide*. URL: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_embedded_ip.pdf.

[6] Linus Torvalds und the Kernel Team. *Linux*. URL: kernel.org.

Anhang

A.

- /
  - Datasheets ..... Datenblätter zur verwendeten Hardware
  - Documentation ..... Dokumentation des Projekts inkl. Schaltplan und Protokoll
  - FPGA_Design ..... Projektverzeichnis der FPGA Beschreibung
    - Datasheets ..... Datenblätter zum verwendeten FPGA
    - Garfield_DesignQuartus Projekt-/ Konfigurationsdateien und QSYS-Projekt
    - ip_extern ..... Verwendete externe IP-Cores
    - ip_intern ..... Verwendete interne, selbstentwickelte IP-Cores
    - output_files ..... FPGA Images mit Konfigurationsdateien
  - Software ..... Software Projektverzeichnis
    - common ..... Gemeinsame Softwarebestandteile getrennt nach Verwendung
    - Software_ARM ..... ARM Software - Linux, Comm_Gateway und alf_urg
    - Software_HQ ..... HQ Software - Garfield Control und melmac_rviz
    - Software_NIOS2 ..... NIOS2 Software - FreeRTOS inkl. Treiber

Alf

1

Generated by Doxygen 1.8.11

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

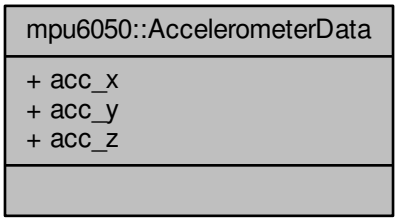
Class Documentation

4.1 mpu6050::AccelerometerData Struct Reference

AccelerometerData.

`#include <mpu6050.hpp>`

Collaboration diagram for mpu6050::AccelerometerData:



Public Attributes

- float **acc_x**
- float **acc_y**
- float **acc_z**

4.1.1 Detailed Description

AccelerometerData.

Definition at line 80 of file mpu6050.hpp.

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

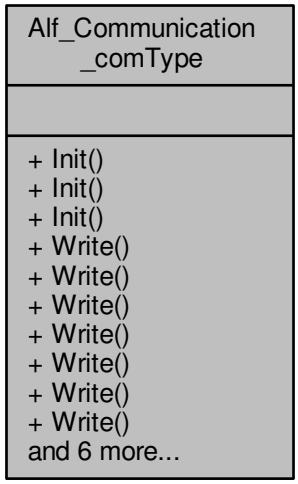
- mpu6050.hpp

4.2 Alf_Communication< _comType > Class Template Reference

CommunicationClass that handles all the communication. Possible template parameters are at the moment std↩ ::fstream, Client and Server. No other com-types are supported.

`#include <alf_communication.hpp>`

Collaboration diagram for Alf_Communication< _comType >:



Public Member Functions

- bool Init (const string &filename)

  Init, for communication as a file.
- bool Init (const string &server, const uint32_t &portno)

  Init, for communication as a client.
- bool Init (const uint32_t &portno)

  Init, for communication as a server.
- bool Write (std::fstream &file, const char *data, const uint32_t &len)

  Writes len bytes from data.
- bool Write (Client &cl, const char *data, const uint32_t &len)
- bool Write (Server &ser, const char *data, const uint32_t &len)
- alf_error Write (Alf_Urg_Measurements_Buffer &buffer)

  This function writes the a buffer to the communication type. Only calling the internal Write(Alf_Urg_Measurement&) function until the buffer is empty.
- alf_error Write (Alf_Urg_Measurement &meas)

  Creates a string with all, for our application, relevant information for one laser-scanner measurement. The structure of this string is described in Alf_Messages.ods, outside this inline documentation.
- alf_error Write (Alf_Drive_Command &command)

  Creates a string with all, for application relevant information for steering the Alf.
- alf_error Write (Alf_Drive_Info &info)

Creates a string with all, for application relevant information for driving infos.

- alf_error WriteInitMessage ()

  Writes the init message over the choosen communication type with information about the urg sensor.

- bool Read (std::fstream &file, char *readPtr, const uint32_t &len)

  Reads len bytes and stores them into readPtr.

- bool Read (Client &cl, char *readPtr, const uint32_t &len)
- bool Read (Server &ser, char *readPtr, const uint32_t &len)
- alf_error Read (Alf_Urg_Measurements_Buffer &readBuffer, alf_mess_types &msgType, const uint32_t &nrPackToRead=1)

  Function reads nrPackToRead Messages and stores them to the readBuffer. If the buffer has not enough free entries, no data is read and nothing is changed. Then another read is possible when the buffer has enough free entries.

- bool EndCommunication (void)

  Function to end the communication.

4.2.1 Detailed Description

template<class _comType>
class Alf_Communication< _comType >

CommunicationClass that handles all the communication. Possible template parameters are at the moment std::fstream, Client and Server. No other com-types are supported.

Definition at line 33 of file alf_communication.hpp.

4.2.2 Member Function Documentation

4.2.2.1 template<class _comType > bool Alf_Communication< _comType >::EndCommunication (void)

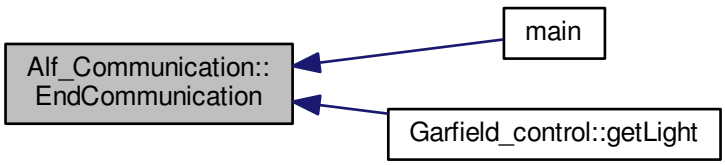
Function to end the communication.

Returns

- true if everything works, false otherwise

Definition at line 343 of file alf_communication.tpp.

Here is the caller graph for this function:



4.2.2.2 template<class _comType > bool Alf_Communication< _comType >::Init (const string & *filename*)

Init, for communication as a file.

Parameters

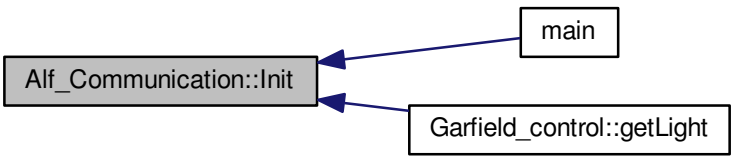
in	*filename*	- for the file, which will be used as communication

Returns

true when everything fine, false otherwise

Definition at line 26 of file alf_communication.tpp.

Here is the caller graph for this function:



4.2.2.3 template<class _comType > bool Alf_Communication< _comType >::Init (const string & *server,* const uint32_t & *portno*)

Init, for communication as a client.

Parameters

in	*server*	- the server IP as a string for the connection
in	*portno*	- the portnumber for the communication

Returns

true when everything fine, false otherwise

Definition at line 39 of file alf_communication.tpp.

4.2.2.4 template<class _comType > bool Alf_Communication< _comType >::Init (const uint32_t & *portno*)

Init, for communication as a server.

Parameters

in	*portno*	- the portnumber for the communication

Returns

true when everything fine, false otherwise

Definition at line 50 of file alf_communication.tpp.

4.2.2.5 template<class _comType > bool Alf_Communication< _comType >::Read (std::fstream & *file*, char * *readPtr*, const uint32_t & *len*)

Reads len bytes and stores them into readPtr.

Parameters

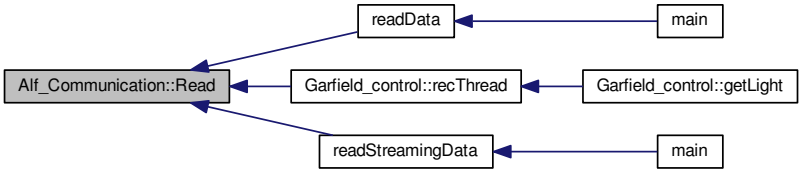
in	*file*	- the fstream from which shall be readed
in,out	*readPtr*	- where the function shall store the readed data
in	*len*	- how much bytes shall be readed

Returns

-

Definition at line 158 of file alf_communication.tpp.

Here is the caller graph for this function:



4.2.2.6 template<class _comType > bool Alf_Communication< _comType >::Read (Client & *cl*, char * *readPtr*, const uint32_t & *len*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

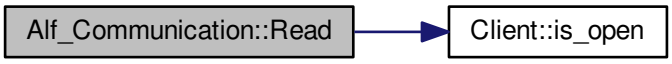
in	*cl*	- the client socket where the data shall be readed
in,out	*readPtr*	- where the function shall store the readed data
in	*len*	- how much bytes shall be readed

Returns

-

Definition at line 170 of file alf_communication.tpp.

Here is the call graph for this function:



4.2.2.7 template<class _comType > bool Alf_Communication< _comType >::Read (Server & *ser,* char * *readPtr,* const uint32_t & *len*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

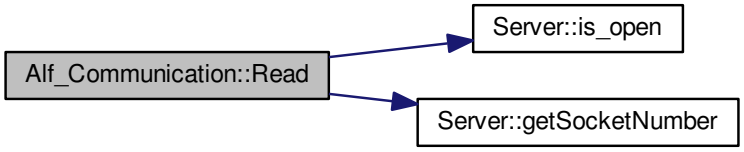
in	*ser*	- the server socket where the data shall be readed
in,out	*readPtr*	- where the function shall store the readed data
in	*len*	- how much bytes shall be readed

Returns

-

Definition at line 181 of file alf_communication.tpp.

Here is the call graph for this function:



4.2.2.8 template<class _comType > alf_error Alf_Communication< _comType >::Read (Alf_Urg_Measurements_Buffer & *readBuffer,* alf_mess_types & *msgType,* const uint32_t & *nrPackToRead* = 1)

Function reads nrPackToRead Messages and stores them to the readBuffer. If the buffer has not enough free entries, no data is read and nothing is changed. Then another read is possible when the buffer has enough free entries.

Parameters

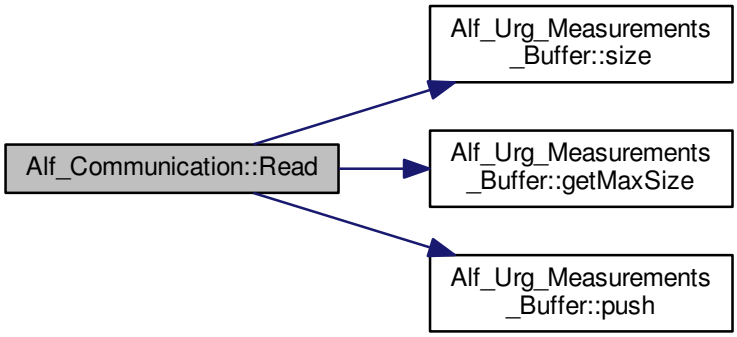
in	*readBuffer*	- This buffer is the memory location for the read data
in	*nrPackToRead*	- default is one packet, otherwise this is the number of packets which will be read

Returns

the first error that occurred or ALF_NO_ERROR when successful

Definition at line 192 of file alf_communication.tpp.

Here is the call graph for this function:



4.2.2.9 template<class _comType > bool Alf_Communication< _comType >::Write (std::fstream & *file,* const char * *data,* const uint32_t & *len*)

Writes len bytes from data.

Parameters

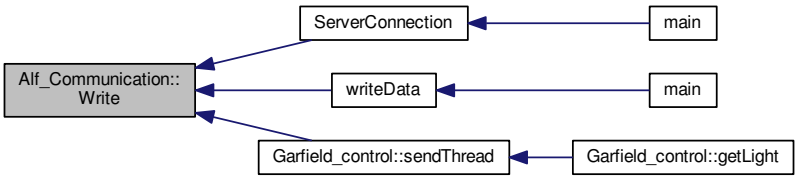
in	*file*	- the fstream, where the bytes should be written
in	*data*	- the pointer to the data which shall be written to the file
in	*len*	- number of bytes from data, which should be written

Returns

- true when everything is fine, false otherwise

Definition at line 60 of file alf_communication.tpp.

Here is the caller graph for this function:



4.2.2.10 template<class _comType > bool Alf_Communication< _comType >::Write (Client & *cl,* const char * *data,* const uint32_t & *len*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

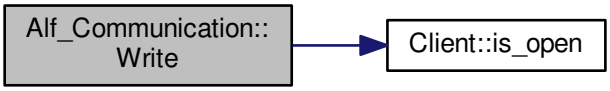
in	*cl*	- the client, writes to a socket
in	*data*	- the pointer to the data which shall be written to the file
in	*len*	- number of bytes from data, which should be written

Returns

- true when everything is fine, false otherwise

Definition at line 70 of file alf_communication.tpp.

Here is the call graph for this function:



4.2.2.11 template<class _comType > bool Alf_Communication< _comType >::Write (Server & *ser*, const char * *data*, const uint32_t & *len*)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

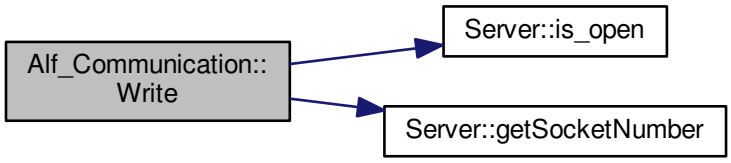
in	*ser*	- the server, writes to a socket
in	*data*	- the pointer to the data which shall be written to the file
in	*len*	- number of bytes from data, which should be written

Returns

- true when everything is fine, false otherwise

Definition at line 80 of file alf_communication.tpp.

Here is the call graph for this function:



4.2.2.12 template<class _comType > alf_error Alf_Communication< _comType >::Write (Alf_Urg_Measurements_Buffer & *buffer*)

This function writes the a buffer to the communication type. Only calling the internal Write(Alf_Urg_Measurement&) function until the buffer is empty.

Parameters

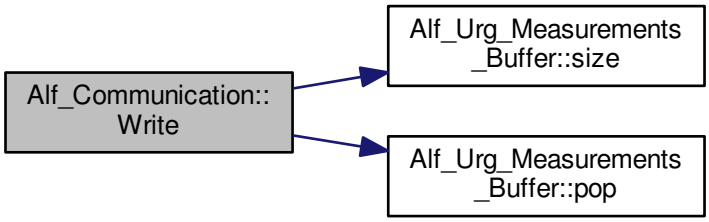
in,out	*buffer*	- the queue which includes all of the measurmenets which was taken to the moment, the function is called. It will be changed on calling this function.

Returns

- alf_error code

Definition at line 90 of file alf_communication.tpp.

Here is the call graph for this function:



4.2.2.13 template<class _comType > alf_error Alf_Communication< _comType >::Write (Alf_Urg_Measurement & *meas*)

Creates a string with all, for our application, relevant information for one laser-scanner measurement. The structure of this string is described in Alf_Messages.ods, outside this inline documentation.

Parameters

in	*meas*	- one laser scanner measurement

Returns

Definition at line 103 of file alf_communication.tpp.

4.2.2.14 template<class _comType > alf_error Alf_Communication< _comType >::Write (Alf_Drive_Command & *command*)

Creates a string with all, for application relevant information for steering the Alf.

Parameters

in	*Command*	- command object

Returns

Definition at line 122 of file alf_communication.tpp.

4.2.2.15 template<class _comType > alf_error Alf_Communication< _comType >::Write (Alf_Drive_Info & *info*)

Creates a string with all, for application relevant information for driving infos.

Parameters

in	*Info*	- Info object

Returns

Definition at line 138 of file alf_communication.tpp.

4.2.2.16 template<class _comType > alf_error Alf_Communication< _comType >::WriteInitMessage ( )

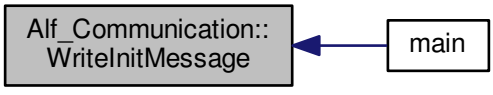
Writes the init message over the choosen communication type with information about the urg sensor.

Returns

- ALF_NO_ERROR if all is ok and it works
- ALF_CANNOT_SEND_MESSAGE if the communication does not work

Definition at line 357 of file alf_communication.tpp.

Here is the caller graph for this function:



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

- alf_communication.hpp
- alf_communication.tpp

4.3 Alf_Data Class Reference

contains all the data about the vehicle which could be exchanges between the vehicle and other applications so serves as interface between a controller and the hardware

`#include <alf_data.hpp>`

Collaboration diagram for Alf_Data:

Static Public Member Functions

- static bool Init_Data (float, float, float, int32_t, int32_t, int32_t)

  initialise the Alf_Data

Static Public Attributes

- static float urg_angle_min = 0.0

  the min angle which the urg laser scanner can provide
- static float urg_angle_max = 0.0

  the max angle which the urg laser scanner can provide
- static float urg_angle_increment = 0

  the increment between two measurments of the laser scanner
- static int urg_time_increment = 100

  the time between two measurements of the laser scanner in ms, with our laser scanner this is 100ms
- static uint32_t urg_range_min = 0

  the minimal distance the laser scanner can measure
- static uint32_t urg_range_max = 0

  the maximal distance the laser scanner can measure

4.3.1 Detailed Description

contains all the data about the vehicle which could be exchanges between the vehicle and other applications so serves as interface between a controller and the hardware

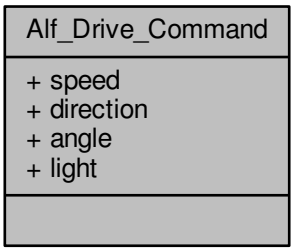
Definition at line 29 of file alf_data.hpp.

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

- alf_data.hpp
- alf_data.cpp

4.4 Alf_Drive_Command Class Reference

Collaboration diagram for Alf_Drive_Command:



Public Attributes

- uint8_t speed

  This variable holds the current speed (0 - 100%)
- uint8_t direction

  This is the direction to drive (0: forward, 1: backward)
- int8_t angle

  This is the currents steering angle (-90 - 90°)
- bool light

  This holds the state of the light.

4.4.1 Detailed Description

Definition at line 31 of file alf_data_info.hpp.

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

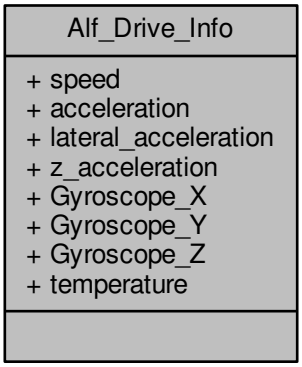
- alf_data_info.hpp

4.5 Alf_Drive_Info Class Reference

The Alf_Drive_Info class holds the Infos for steering the Alf.

`#include <alf_data_info.hpp>`

Collaboration diagram for Alf_Drive_Info:



Public Attributes

- uint8_t speed

  This is the current speed.
- float acceleration

  This is the acceleration of the car.
- float lateral_acceleration

  This is the lateral acceleration of the car.
- float z_acceleration

  This is the acceleration in Z direction.
- float Gyroscope_X

  This is the Gyroscope value x axis.
- float Gyroscope_Y

  This is the Gyroscope value y axis.
- float Gyroscope_Z

  This is the Gyroscope value z axis.
- float temperature

  This is the temperature.

4.5.1 Detailed Description

The Alf_Drive_Info class holds the Infos for steering the Alf.

Definition at line 11 of file alf_data_info.hpp.

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

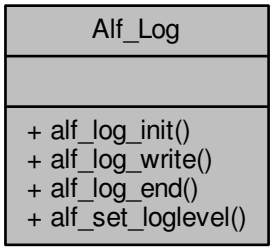
- alf_data_info.hpp

4.6 Alf_Log Class Reference

This class handle all the log informations. There will be always a log file, additional the log can be printed to standard output.

`#include <alf_log.hpp>`

Collaboration diagram for Alf_Log:



Static Public Member Functions

- static bool alf_log_init (const std::string &filename="dummy.alf_log", const alf_log_level_e &log_level=log_debug, const bool &console_output=false)

  Initialize the logging functionality (performed with a file)
- static bool alf_log_write (const std::string &log_entry, const alf_log_level_e &log_level=log_debug)

  Writes a log entry.
- static bool alf_log_end (void)

  close the logging
- static void alf_set_loglevel (const alf_log_level_e &log_level)

  Set the log level.

4.6.1 Detailed Description

This class handle all the log informations. There will be always a log file, additional the log can be printed to standard output.

Definition at line 45 of file alf_log.hpp.

4.6.2 Member Function Documentation

4.6.2.1 bool Alf_Log::alf_log_end (void) [static]

close the logging

Parameters

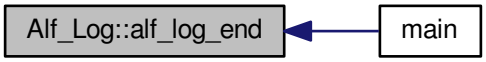
in	-	

Returns

true if successful otherwise false

Definition at line 52 of file alf_log.cpp.

Here is the caller graph for this function:



4.6.2.2 bool Alf_Log::alf_log_init (const std::string & *filename* = "dummy.alf_log", const alf_log_level_e & *log_level* = log_debug, const bool & *console_output* = false) [static]

Initialize the logging functionality (performed with a file)

Parameters

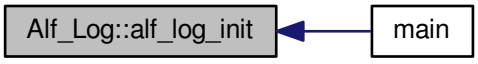
in	*filename*	Path to File
in	*loglevel*	All Messages with level above will be logged
in	*consoleoutput*	If true all messages will be printed on console ouptut

Returns

true if successful otherwise false

Definition at line 28 of file alf_log.cpp.

Here is the caller graph for this function:



4.6.2.3 bool Alf_Log::alf_log_write (const std::string & *log_entry*, const alf_log_level_e & *log_level* = log_debug) [static]

Writes a log entry.

Parameters

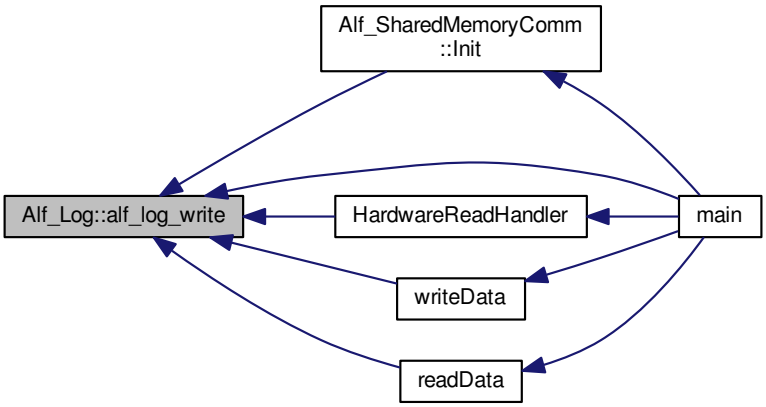
in	*log_entry*	the message to be logged
in	*log_level*	the significance of the message

Returns

true if successful otherwise false

Definition at line 63 of file alf_log.cpp.

Here is the caller graph for this function:



4.6.2.4 void Alf_Log::alf_set_loglevel (const alf_log_level_e & *log_level*) [static]

Set the log level.

Parameters

in	*log_level*	which messages should be logged from now on

Returns

-

Definition at line 93 of file alf_log.cpp.

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

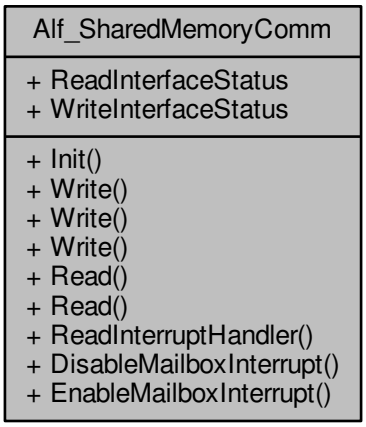
- alf_log.hpp
- alf_log.cpp

4.7 Alf_SharedMemoryComm Class Reference

Implementation for communcatiing via a shared memory section on the fpga. Abstraction for the mailbox, the hardware mutex and the shared memory in both directions.

`#include <alf_sharedmemory.hpp>`

Collaboration diagram for Alf_SharedMemoryComm:



Public Member Functions

- bool Init (uint32_t sh_mem_wr_addr, uint32_t wr_mutex_addr, uint32_t wr_mb_addr, uint32_t sh_mem_rd↩ _addr, uint32_t rd_mutex_addr, uint32_t rd_mb_addr, uint16_t cp_id, uint32_t addr_offset)

  Initialize the hardware communication with the shared memory.
- alf_error Write (const Alf_Drive_Info &drive)

  Writes an Alf_Drive_Info to the shared memory section.
- alf_error Write (const Alf_Drive_Command &drive)

  Write an Alf_Drive_Command to the shared memory section.
- alf_error Write (uint32_t &num)

  Writes a simple number into the mailbox. This number will be used within the mailbox comannd register and in the shared memory!
- alf_error Read (Alf_Drive_Command &drive)

  Reads one Alf_Drive_Command from the shared memory, if there is one to read. This function changes memory of the given object!
- alf_error Read (Alf_Drive_Info &drive)

Reads one Alf_Drive_Info from the shared memory if there is one to read. Returns an errorcode otherwise.

- void **ReadInterruptHandler** (void)
- void DisableMailboxInterrupt ()

  Disables the interrupt on receiving messages.
- void EnableMailboxInterrupt ()

  Enables all interrupts of the mailbox (at this moment: only on receiving messages)

Public Attributes

- bool ReadInterfaceStatus

  Flag to disable (=false) or enable (=true) the read interface.
- bool WriteInterfaceStatus

  Enables (=true) or disables (=false) the write operations to hardware. If set to false, all write operations Write will return the error #ALF_WRITE_SHARED_MEMORY_DISABLED.

4.7.1 Detailed Description

Implementation for communcatiing via a shared memory section on the fpga. Abstraction for the mailbox, the hardware mutex and the shared memory in both directions.

Definition at line 82 of file alf_sharedmemory.hpp.

4.7.2 Member Function Documentation

4.7.2.1 bool Alf_SharedMemoryComm::Init (uint32_t *sh_mem_wr_addr,* uint32_t *wr_mutex_addr,* uint32_t *wr_mb_addr,* uint32_t *sh_mem_rd_addr,* uint32_t *rd_mutex_addr,* uint32_t *rd_mb_addr,* uint16_t *cp_id,* uint32_t *addr_offset*)

Initialize the hardware communication with the shared memory.

Parameters

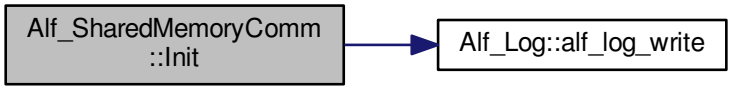
sh_mem_wr_addr	The base address of the shared memory where the instance of this class should write its data
wr_mutex_addr	The base address of the mutex which should lock all writes from this instance
wr_mb_addr	The base address of the mailbox which this instance should write his data
sh_mem_rd_addr	The base address of the shared memory where the instance of this class should read data (receiver)
rd_mutex_addr	The base address of the mutex where the instance is the receiver
rd_mb_addr	The base address of the mailbox where the instance is the receiver
cp_id	The cpu id which instantiate this class. Normally 0x01 for HSP and 0x03 for NIOS 2
addr_offset	The general address offset for all address defined. In NIOS2 this should be 0, within HPS on Cylcone V this is normally 0xff200000

Returns

true if all addresses could be mapped in a proper way and all read/write operation can be used, false otherwise

Definition at line 33 of file alf_sharedmemory.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

4.7.2.2 alf_error Alf_SharedMemoryComm::Read (Alf_Drive_Command & *drive*)

Reads one Alf_Drive_Command from the shared memory, if there is one to read. This function changes memory of the given object!

Parameters

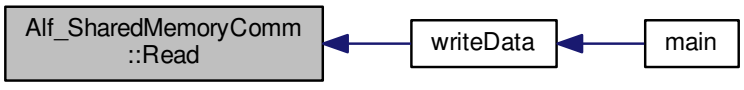
drive	The Alf_Drive_Command where the informations should be stored

Returns

One of ALF_ERROR_CODES

Definition at line 243 of file alf_sharedmemory.cpp.

Here is the caller graph for this function:



4.7.2.3 alf_error Alf_SharedMemoryComm::Read (Alf_Drive_Info & *drive*)

Reads one Alf_Drive_Info from the shared memory if there is one to read. Returns an errorcode otherwise.

Parameters

drive	The Alf_Drive_Info where the informations should be stored

Returns

One of #Alf_Drive_CODES

Definition at line 234 of file alf_sharedmemory.cpp.

4.7.2.4 alf_error Alf_SharedMemoryComm::Write (const Alf_Drive_Info & *drive*)

Writes an Alf_Drive_Info to the shared memory section.

Parameters

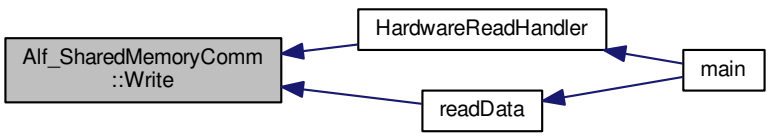
drive	The Alf_Drive_Info object which should be written to the shared memory via memcpj

Returns

One of ALF_ERROR_CODES

Definition at line 201 of file alf_sharedmemory.cpp.

Here is the caller graph for this function:



4.7.2.5 alf_error Alf_SharedMemoryComm::Write (const Alf_Drive_Command & *drive*)

Write an Alf_Drive_Command to the shared memory section.

Parameters

drive	The Alf_Drive_Command which should be written to the shared memory

Returns

One of ALF_ERROR_CODES

Definition at line 197 of file alf_sharedmemory.cpp.

4.7.2.6 alf_error Alf_SharedMemoryComm::Write (uint32_t & *num*)

Writes a simple number into the mailbox. This number will be used within the mailbox comannd register and in the shared memory!

Parameters

num	The number

Returns

One of ALF_ERROR_CODES

Definition at line 205 of file alf_sharedmemory.cpp.

4.7.3 Member Data Documentation

4.7.3.1 bool Alf_SharedMemoryComm::ReadInterfaceStatus

Flag to disable (=false) or enable (=true) the read interface.

Attention

Actual not used, just for completness

Definition at line 196 of file alf_sharedmemory.hpp.

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

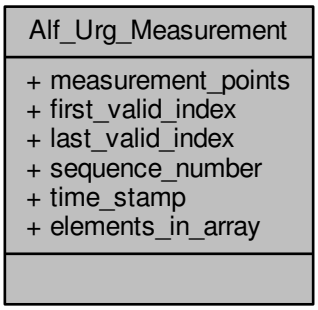
- alf_sharedmemory.hpp
- alf_sharedmemory.cpp

4.8 Alf_Urg_Measurement Class Reference

This class stands for **one** whole measurement of the laser scanner and provides additional informations It contains all measurement values, also this one, which are invalid in case of the datasheet.

`#include <alf_data.hpp>`

Collaboration diagram for Alf_Urg_Measurement:



Public Attributes

- long int measurement_points [elements_in_array]

  The storage for the measurement points. Each index represents one urg_angle_increment.
- uint32_t first_valid_index

  The first index of the measurement_points which should be used (derived from the data sheet)
- uint32_t last_valid_index

  The last index of the measurement_points which should be used.
- uint32_t sequence_number

  To provide a chronological sequence of the various measurements.
- long int time_stamp

  The timestamp of the measurement. Its no absolut time, just the internal counter, so several measurements can be set in an chronologically relation.

Static Public Attributes

- static constexpr uint32_t elements_in_array = URG_NUMBER_OF_MEASUREMENT_DATA + 1

  how much measurement points do we have for one measurement

4.8.1 Detailed Description

This class stands for **one** whole measurement of the laser scanner and provides additional informations It contains all measurement values, also this one, which are invalid in case of the datasheet.

Definition at line 54 of file alf_data.hpp.

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

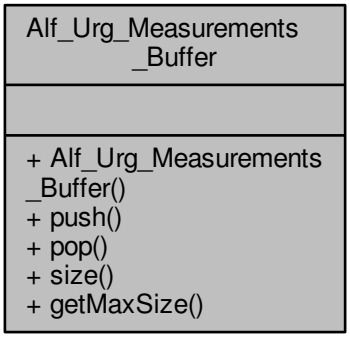
- alf_data.hpp

4.9 Alf_Urg_Measurements_Buffer Class Reference

This buffer can store a set of Alf_Urg_Measurement . It use the std::queue for storing the data and have a maximum size to determine the maximum RAM size which can be used.

`#include <alf_data.hpp>`

Collaboration diagram for Alf_Urg_Measurements_Buffer:



Public Member Functions

- Alf_Urg_Measurements_Buffer (uint32_t size=MAX_SIZE_OF_MEASUREMENT_BUFFER_DEFAULT)

  constructor for the Alf_Urg_Measurement_Buffer set _max_size to the given value or default to the macro MAX_SIZE_OF_MEASUREMENT_BUFFER_DEFAULT
- alf_error push (const Alf_Urg_Measurement &)

  append one Alf_Urg_Measurement to the buffer
- alf_error pop (Alf_Urg_Measurement *)

  pops one element of the buffer and stores it in the memory given by a pointer
- uint32_t size () const

  returns the actual size of the queue (so how much elements are stored within)
- uint32_t getMaxSize (void) const

  returns the maximal number of elements which could be stored

4.9.1 Detailed Description

This buffer can store a set of Alf_Urg_Measurement . It use the std::queue for storing the data and have a maximum size to determine the maximum RAM size which can be used.

Definition at line 74 of file alf_data.hpp.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Alf_Urg_Measurements_Buffer::Alf_Urg_Measurements_Buffer (uint32_t *size* = MAX_SIZE_OF_MEASUREMENT_BUFFER_DEFAULT)

constructor for the Alf_Urg_Measurement_Buffer set _max_size to the given value or default to the macro MAX_SIZE_OF_MEASUREMENT_BUFFER_DEFAULT

Parameters

in	*size*	- the size, default MAX_SIZE_OF_MEASUREMENT_BUFFER_DEFAULT

Returns

-

Definition at line 37 of file alf_data.cpp.

4.9.3 Member Function Documentation

4.9.3.1 uint32_t Alf_Urg_Measurements_Buffer::getMaxSize (void) const

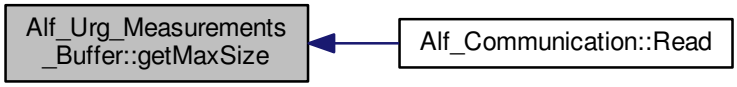
returns the maximal number of elements which could be stored

Returns

the maximal number of elements

Definition at line 70 of file alf_data.cpp.

Here is the caller graph for this function:



4.9.3.2 alf_error Alf_Urg_Measurements_Buffer::pop (Alf_Urg_Measurement * *a*)

pops one element of the buffer and stores it in the memory given by a pointer

Parameters

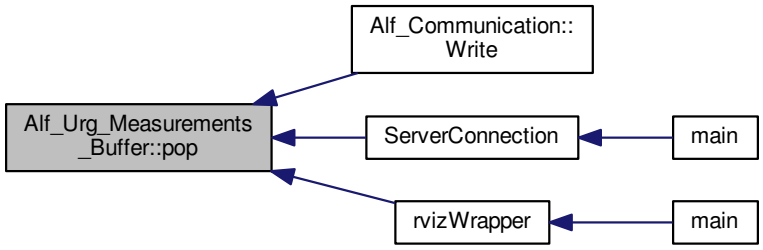
in,out	*a*	- the memory where the Alf_Urg_Measurement shall be stored

Returns

- ALF_NO_ERROR if everything works
- ALF_NOTHING_IN_BUFFER if there is no more element in the queue which could be removed

Definition at line 53 of file alf_data.cpp.

Here is the caller graph for this function:



4.9.3.3 alf_error Alf_Urg_Measurements_Buffer::push (const Alf_Urg_Measurement & *a*)

append one Alf_Urg_Measurement to the buffer

Parameters

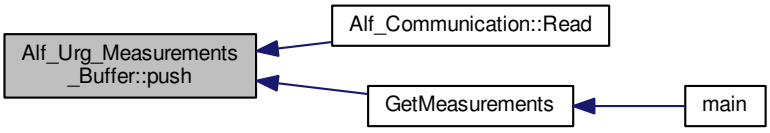
in	*a*	- the measurement

Returns

- ALF_NO_ERROR if the element can be appended to the queue
- ALF_BUFFER_IS_FULL if the queue is full and cannot store any additonal elements

Definition at line 41 of file alf_data.cpp.

Here is the caller graph for this function:



4.9.3.4 uint32_t Alf_Urg_Measurements_Buffer::size ( ) const

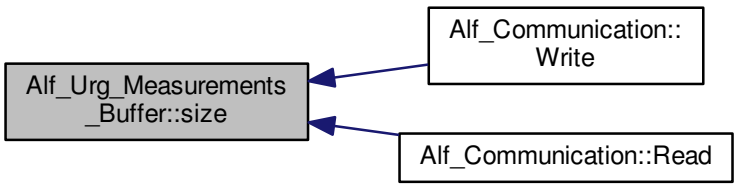
returns the actual size of the queue (so how much elements are stored within)

Returns

the number of elements

Definition at line 66 of file alf_data.cpp.

Here is the caller graph for this function:



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

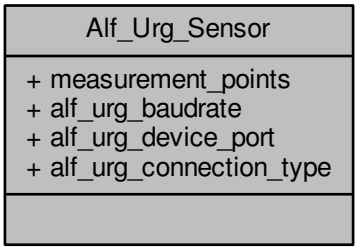
- alf_data.hpp
- alf_data.cpp

4.10 Alf_Urg_Sensor Class Reference

Represents the laser scanner on the alf vehicle and provide common settings etc.

`#include <alf_sensors.hpp>`

Collaboration diagram for Alf_Urg_Sensor:



Static Public Attributes

- static const uint16_t measurement_points = 768

  how much measurement points does the sensor have
- static const long alf_urg_baudrate = 115200

  the baudrate to communicate with the scanner
- static const std::string alf_urg_device_port = "/dev/ttyACM0"

  the port on which the scanner is connected with the hardware
- static const urg_connection_type_t alf_urg_connection_type = URG_SERIAL

  which communication type we use

4.10.1 Detailed Description

Represents the laser scanner on the alf vehicle and provide common settings etc.

Attention

this settings are only vaild with the URG-04LX

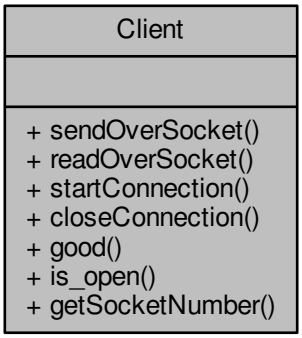
Definition at line 18 of file Software_ARM/alf_urg/alf_sensors.hpp.

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

- Software_ARM/alf_urg/alf_sensors.hpp
- Software_ARM/alf_urg/alf_sensors.cpp

4.11 Client Class Reference

Collaboration diagram for Client:



Public Member Functions

- alf_error sendOverSocket (const string &data)

  Sending the string to over the socket via the underlying linux functaion.
- alf_error readOverSocket (string &s)

  reads a string object over the socket. three conditions for read ending are given 1) if the end delimiter is reached ';' 2) no more readable data is available 3) more than 20 characters were read and no delimiter '|' or end delimiter ';' was read
- uint8_t startConnection (const uint32_t &_portno, const string &_server)

  starts the socket connection
- void closeConnection (void)

  closes the connection, communication is no longer possible
- bool good ()

  dummy function to satisfy the compiler (std::fstream, Server/Client all have the good() function so no explicit type handling must be done
- bool is_open ()

  returns the state of the socket connection
- int32_t **getSocketNumber** (void)

4.11.1 Detailed Description

Definition at line 16 of file Client_Server_impl.hpp.

4.11.2 Member Function Documentation

4.11.2.1 bool Client::is_open ( ) [inline]

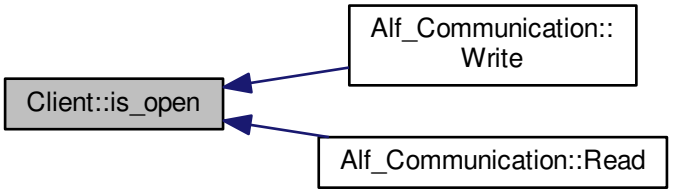
returns the state of the socket connection

Returns

true if connection is good, false otherwise

Definition at line 58 of file Client_Server_impl.hpp.

Here is the caller graph for this function:



4.11.2.2 alf_error Client::readOverSocket (string & *s*)

reads a string object over the socket. three conditions for read ending are given 1) if the end delimiter is reached ';' 2) no more readable data is available 3) more than 20 characters were read and no delimiter '|' or end delimiter ';' was read

Parameters

in	*the*	string data object were the read data is stored (no appending string gets overwritten)

Returns

- ALF_NO_ERROR if the read works
- ALF_CANNOT_READ_SOCKET if it does not work

Definition at line 55 of file Client_Server_impl.cpp.

4.11.2.3 alf_error Client::sendOverSocket (const string & *data*)

Sending the string to over the socket via the underlying linux functaion.

Parameters

in	*data*	- the string with the message which shall be transmitted

Returns

- ALF_NO_ERROR if the message can be transmitted
- ALF_SOCKET_NOT_READY if the socket is not initialised and
- ALF_CANNOT_SEND_MESSAGE if there are errors in the linux functionalitys, typical triggered by a too long message etc.

Definition at line 40 of file Client_Server_impl.cpp.

4.11.2.4 uint8_t Client::startConnection (const uint32_t & *_portno,* const string & *_server*)

starts the socket connection

Parameters

in	*the*	portnumber
in	*servername*	

Returns

1 if successful otherwise error < 0

Definition at line 15 of file Client_Server_impl.cpp.

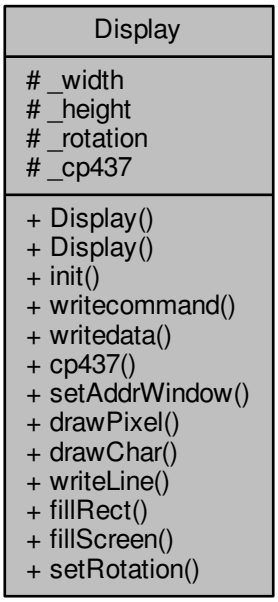
The documentation for this class was generated from the following files:

- Client_Server_impl.hpp
- Client_Server_impl.cpp

4.12 Display Class Reference

`#include <Display.hpp>`

Collaboration diagram for Display:



Public Member Functions

- Display ()=delete
- Display (alt_16 width, alt_16 height)
- void init (alt_u16 bgcolor)
- void writecommand (alt_u8 c)
- void writedata (alt_u8 c)
- void cp437 (bool x)
- void setAddrWindow (alt_u16 x0, alt_u16 y0, alt_u16 x1, alt_u16 y1)
- void drawPixel (alt_16 x, alt_16 y, alt_u16 color)
- void drawChar (alt_16 x, alt_16 y, unsigned char c, alt_u16 color, alt_u16 bg, alt_u8 size)
- void writeLine (const char ∗constline, alt_u16 color, alt_u8 size)
- void fillRect (alt_16 x, alt_16 y, alt_16 w, alt_16 h, alt_u16 color)
- void fillScreen (alt_u16 color)
- void setRotation (alt_u8 m)

Protected Attributes

- alt_16 _width
- alt_16 _height
- alt_u8 _rotation
- bool _cp437

4.12.1 Detailed Description

class Display wich contains all necessary methods for witing to LCD

Definition at line 128 of file Display.hpp.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 Display::Display ( ) [delete]

Delete the default constructor

4.12.2.2 Display::Display (alt_16 *width*, alt_16 *height*)

Constructor for Display class

Parameters

width	is for instancing the display (use 240 as width for used display)
height	is for instancing the display (use 240 as width for used display)

Definition at line 72 of file Display.cpp.

4.12.3 Member Function Documentation

4.12.3.1 void Display::cp437 (bool *x* = true)

Method for enabling/disabling cp437 charset

Parameters

x	enabled if true, disabled if false

Definition at line 340 of file Display.cpp.

4.12.3.2 void Display::drawChar (alt_16 *x*, alt_16 *y*, unsigned char *c*, alt_u16 *color*, alt_u16 *bg*, alt_u8 *size*)

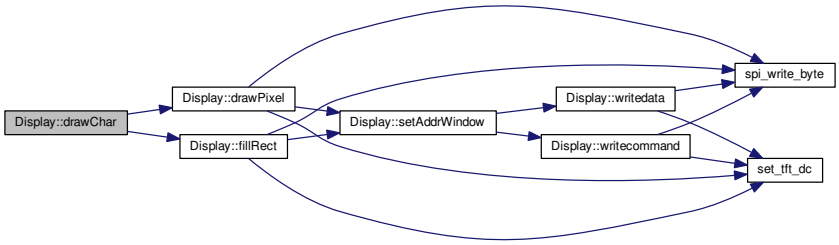
Method for drawing char to LCD

Parameters

x	is the x position for the character
y	is the y position for the character is the character to be printed is the textcolor is the background color is the textsize

Definition at line 224 of file Display.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

4.12.3.3 void Display::drawPixel (alt_16 *x*, alt_16 *y*, alt_u16 *color*)

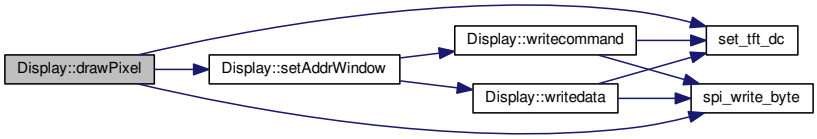
Draw Pixel Function for writing char on LCD

Parameters

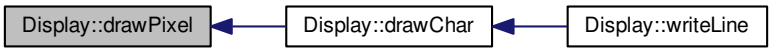
x	This is the x position of the pixel that should be written
y	This is the y position of the pixel that should be written
color	The color of the pixel

Definition at line 212 of file Display.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.12.3.4 void Display::fillRect (alt_16 *x*, alt_16 *y*, alt_16 *w*, alt_16 *h*, alt_u16 *color*)

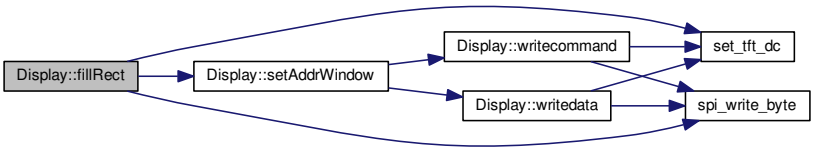
Method fill Rect creates a filled rectangle with one color. This funtcion is used by fillScreen

Parameters

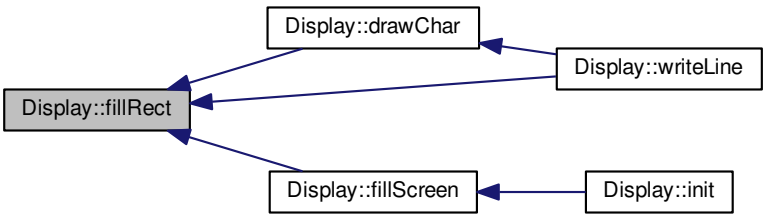
x	This is the x position where the rectangle should start
y	This is the y position where the rectangle should start
w	This is the width of the rectangle
h	This is the height of the rectangle
color	This is the color in which the rectangle should be filled

Definition at line 301 of file Display.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.12.3.5 void Display::fillScreen (alt_u16 *color*)

Method for filling Screen with one color. Function uses the fillRect() method

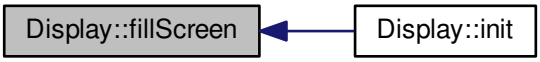
Parameters

color	The color for filling the screen

Definition at line 297 of file Display.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



4.12.3.6 void Display::init (alt_u16 *bgcolor*)

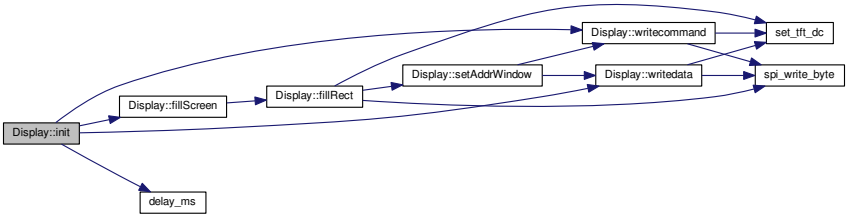
Init Function for initializing the LCD

Parameters

bgcolor	This bgcolor is used for filling the screen after initializing and for clearing lines to override

Definition at line 81 of file Display.cpp.

Here is the call graph for this function:



4.12.3.7 void Display::setAddrWindow (alt_u16 *x0*, alt_u16 *y0*, alt_u16 *x1*, alt_u16 *y1*)

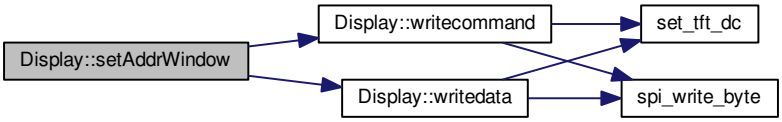
Method for setting the internal address for a x-y coordinate. It is used by drawPixel()

Parameters

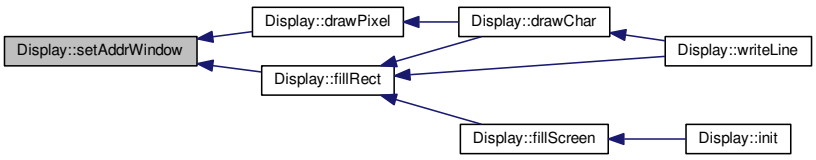
x0	The x0 position of the address window
y0	The y0 position of the address window
x1	The x1 position of the address window
y1	The y1 position of the address window

Definition at line 324 of file Display.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.12.3.8 void Display::setRotation (alt_u8 *m*)

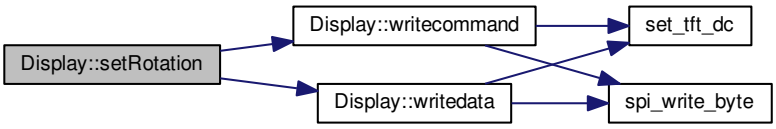
Rotate Screen with parameter m

Parameters

m	could take values from 0 to 3 (0: , 1: , 2: , 3:)

Definition at line 344 of file Display.cpp.

Here is the call graph for this function:



4.12.3.9 void Display::writecommand (alt_u8 *c*)

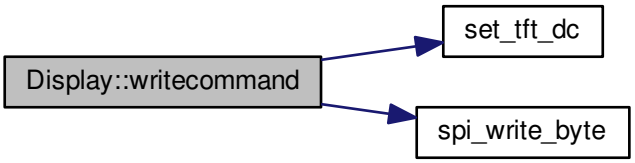
Method writecommand sends a command to the LCD DC Pin low to send a command

Parameters

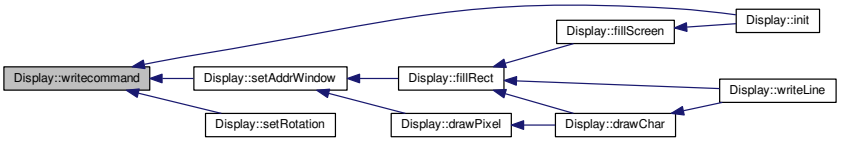
c	command byte to be written

Definition at line 200 of file Display.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.12.3.10 void Display::writedata (alt_u8 *c*)

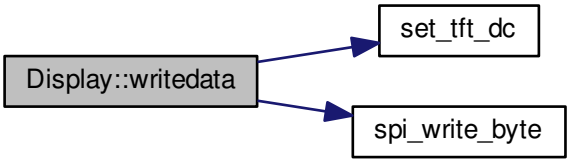
Method writedata sends data to the LCD DC Pin high to send data

Parameters

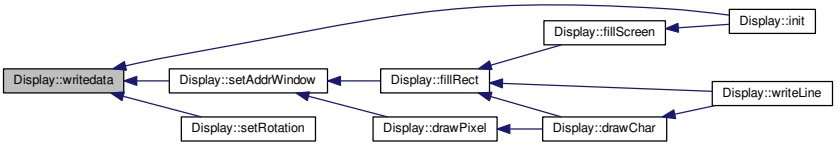
c	data byte to be written

Definition at line 206 of file Display.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.12.3.11 void Display::writeLine (const char * *constline,* alt_u16 *color,* alt_u8 *size*)

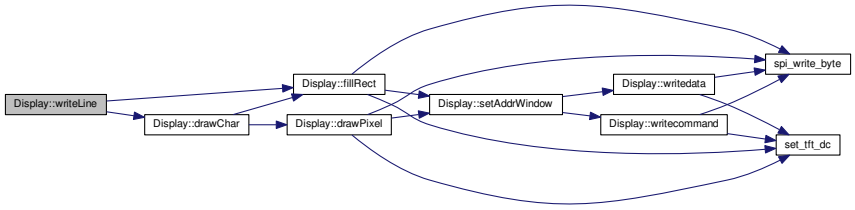
Method for writing a line to the screen. Bevor each line the current number is printed, to check the current line. After 100 lines the number is to 1 again

Parameters

constline	this is the line to be printed
color	The textcolor which should be printed
size	Is the textsize (1: normal. 2: double size. 3: triple size)

Definition at line 249 of file Display.cpp.

Here is the call graph for this function:



4.12.4 Member Data Documentation

4.12.4.1 bool Display::_cp437 [protected]

cp charset enabled or disabled

Definition at line 233 of file Display.hpp.

4.12.4.2 alt_16 Display::_height [protected]

height of the display

Definition at line 231 of file Display.hpp.

4.12.4.3 alt_u8 Display::_rotation [protected]

rotation of the dsiplay (0-3)

Definition at line 232 of file Display.hpp.

4.12.4.4 alt_16 Display::_width [protected]

width of the display

Definition at line 230 of file Display.hpp.

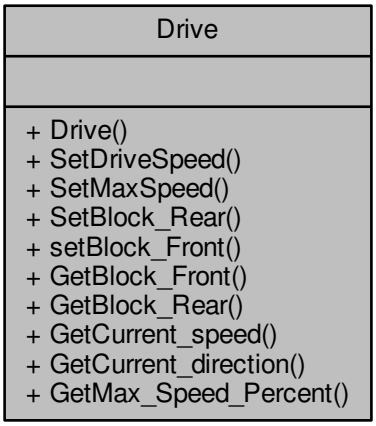
The documentation for this class was generated from the following files:

- Display.hpp
- Display.cpp

4.13 Drive Class Reference

`#include <Drive.hpp>`

Collaboration diagram for Drive:



Public Member Functions

- Drive ()=delete

Static Public Member Functions

- static void SetDriveSpeed (alt_u8 direction, alt_u8 speed)
- static void SetMaxSpeed (alt_u8 max_percent_speed)
- static void SetBlock_Rear (const bool val)

  set/get Methods for variables
- static void **setBlock_Front** (const bool val)
- static bool **GetBlock_Front** (void)
- static bool **GetBlock_Rear** (void)
- static alt_u8 **GetCurrent_speed** (void)
- static alt_u8 **GetCurrent_direction** (void)
- static alt_u8 **GetMax_Speed_Percent** (void)

4.13.1 Detailed Description

class Drive for setting the speed an direction and getting the speed from the rotary encoder

Definition at line 10 of file Drive.hpp.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 Drive::Drive ( ) [delete]

Delete the default constructor

4.13.3 Member Function Documentation

4.13.3.1 void Drive::SetDriveSpeed (alt_u8 *direction*, alt_u8 *speed*) [static]

Method SetDriveSpeed for setting the speed and direction to the motor. The speed value gets rescaled by the maximum percent speed given by SetMaxSpeed(alt_u8 max_percent_speed)

Parameters

in	*direction*	1: backwards, 0: forward
in	*speed*	value from 0 - 255 for setting speed. SPEED_PRESCALER is divided by speed for setting max speed

Definition at line 19 of file Drive.cpp.

4.13.3.2 void Drive::SetMaxSpeed (alt_u8 *max_percent_speed*) [static]

Method SetMaxSpeed for setting the max percentage speed. The speed value which is given by SetDriveSpeed gets rescaled to the maximum percentage speed value

Parameters

in	*max_percent_speed*	is the maximum percentage speed value.

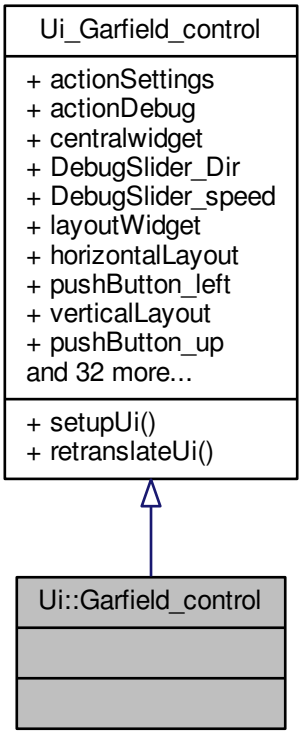
Definition at line 37 of file Drive.cpp.

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

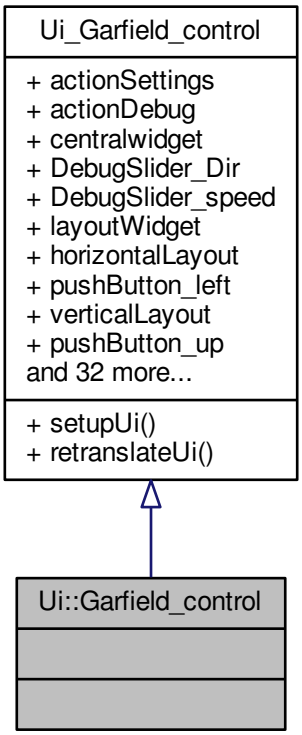
- Drive.hpp
- Drive.cpp

4.14 Ui::Garfield_control Class Reference

Inheritance diagram for Ui::Garfield_control:



Collaboration diagram for Ui::Garfield_control:



Additional Inherited Members

4.14.1 Detailed Description

Definition at line 311 of file ui_Garfield_control.h.

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

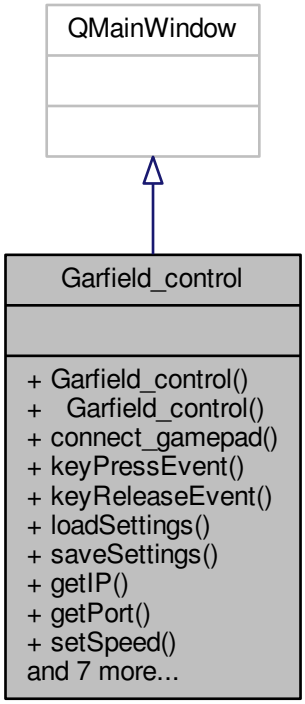
- ui_Garfield_control.h

4.15 Garfield_control Class Reference

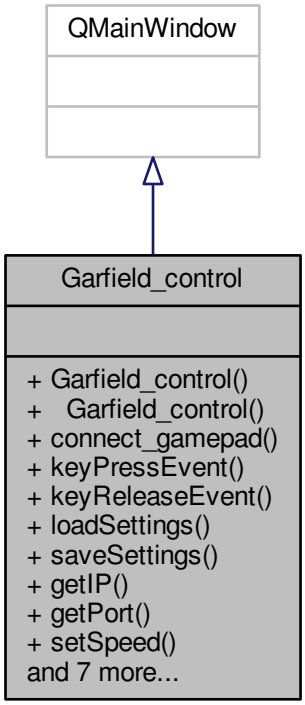
Garfield_control is the main class that provides all functionalities for the Garfield control program.

`#include <Garfield_control.h>`

Inheritance diagram for Garfield_control:



Collaboration diagram for Garfield_control:



Public Member Functions

- Garfield_control (QMainWindow ∗parent=0)

  The constructor.
- ∼Garfield_control ()

  The destructor.
- bool connect_gamepad ()

  connect_gamepad() function connects the gamepad. It takes the _Dev object which contains the device name
- void keyPressEvent (QKeyEvent ∗e)

  keyPressEvent handles all pressed keys which are necessary for controling the car. After that the keyPressEvent of the base class is called
- void keyReleaseEvent (QKeyEvent ∗e)

  keyReleaseEvent handles all released keys which are necessary for controling the car. After that the keyRelease↩ Event of the base class is called
- void loadSettings ()

  ladSettings() loads the settings file and stores all settings in its variables
- void saveSettings ()

  saveSettings() saves all settings to the Garfield.conf file if the settings window gets closed
- void getIP (QString ∗IP)

  getIP() is the getter function for the IP
- void getPort (int ∗Port)

  getPort() is the getter function for the Port

- void setSpeed (int speed)

  setSpeed() is the setter function for the Speed
- void setAngle (int angle)

  setAngle() is the setter function for the Angle
- void setLight (bool light)

  setLight() is the setter function for the Light
- void getSpeed (int &speed)

  getSpeed() is the getter function for the Speed
- void getAngle (int &angle)

  getAngle() is the getter function for the Angle
- void getLight (bool &light)

  getLight() is the getter function for the Light
- void sendThread ()

  sendThread() is executed in an extra thread. It handles all data that are sent over the socket
- void recThread ()

  recThread() is executed in an extra thread. It handles all data that are received over the socket

4.15.1 Detailed Description

Garfield_control is the main class that provides all functionalities for the Garfield control program.

Definition at line 90 of file Garfield_control.h.

4.15.2 Member Function Documentation

4.15.2.1 void Garfield_control::keyPressEvent (QKeyEvent * e)

keyPressEvent handles all pressed keys which are necessary for controling the car. After that the keyPressEvent of the base class is called

Parameters

in	*e*	- The QKeyEvent of the key that is pressed.

Definition at line 187 of file Garfield_control.cpp.

4.15.2.2 void Garfield_control::keyReleaseEvent (QKeyEvent * e)

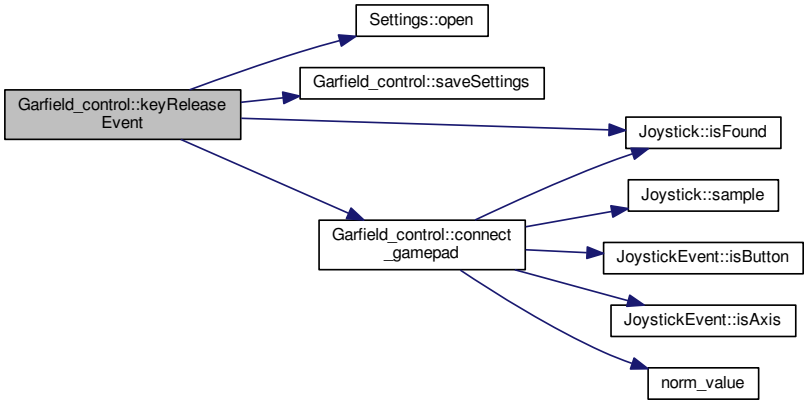
keyReleaseEvent handles all released keys which are necessary for controling the car. After that the keyRelease↩ Event of the base class is called

Parameters

in	*e*	- The QKeyEvent of the key that is released.

Definition at line 212 of file Garfield_control.cpp.

Here is the call graph for this function:



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

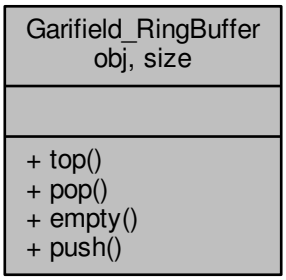
- Garfield_control.h
- Garfield_control.cpp

4.16 Garifield_RingBuffer< obj, size > Class Template Reference

Implementation of a ringbuffer with fixed size. If the queue is full, the oldest element will be overwritten.

```
#include <alf_sharedmemory.hpp>
```

Collaboration diagram for Garifield_RingBuffer< obj, size >:



Public Member Functions

- obj top ()

  returns the top element on the ring buffer (is the actualst)
- void pop ()

  Removes the top element of the ringbuffer. This element is the actualst element, next top element ist n-1.
- bool empty ()

  Is the ring buffer empty?
- void push (const obj &a)

  Pushs a element to the ring buffer. If the ring buffer is full, the oldest element in there will be overwritten.

4.16.1 Detailed Description

template<class obj, uint32_t size>
class Garifield_RingBuffer< obj, size >

Implementation of a ringbuffer with fixed size. If the queue is full, the oldest element will be overwritten.

Definition at line 22 of file alf_sharedmemory.hpp.

4.16.2 Member Function Documentation

4.16.2.1 template<class obj, uint32_t size> bool Garifield_RingBuffer< obj, size >::empty ( ) [inline]

Is the ring buffer empty?

Returns

true = empty, false = elements in the ring buffer

Definition at line 51 of file alf_sharedmemory.hpp.

4.16.2.2 template<class obj, uint32_t size> void Garifield_RingBuffer< obj, size >::push (const obj & *a*) [inline]

Pushs a element to the ring buffer. If the ring buffer is full, the oldest element in there will be overwritten.

Parameters

in	*a*	The element to push into.

Definition at line 59 of file alf_sharedmemory.hpp.

4.16.2.3 template<class obj, uint32_t size> obj Garifield_RingBuffer< obj, size >::top ( ) [inline]

returns the top element on the ring buffer (is the actualst)

Returns

the top element, could be of any datatype

Attention

a call to pop() is necessary if the element should removed from the ring buffer

Definition at line 30 of file alf_sharedmemory.hpp.

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

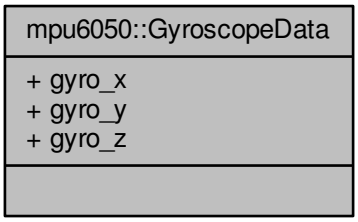
- alf_sharedmemory.hpp

4.17 mpu6050::GyroscopeData Struct Reference

GyroscopeData.

`#include <mpu6050.hpp>`

Collaboration diagram for mpu6050::GyroscopeData:



Public Attributes

- float **gyro_x**
- float **gyro_y**
- float **gyro_z**

4.17.1 Detailed Description

GyroscopeData.

Definition at line 88 of file mpu6050.hpp.

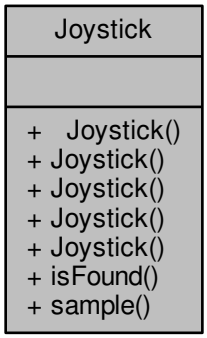
The documentation for this struct was generated from the following file:

- mpu6050.hpp

4.18 Joystick Class Reference

`#include <joystick.h>`

Collaboration diagram for Joystick:



Public Member Functions

- Joystick ()
- Joystick (int joystickNumber)
- Joystick (std::string devicePath)
- Joystick (std::string devicePath, bool blocking)
- bool isFound ()
- bool sample (JoystickEvent *event)

4.18.1 Detailed Description

Represents a joystick device. Allows data to be sampled from it.

Definition at line 103 of file joystick.h.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 Joystick::Joystick ( )

Initialises an instance for the first joystick: /dev/input/js0

Definition at line 28 of file joystick.cpp.

4.18.2.2 Joystick::Joystick (int *joystickNumber*)

Initialises an instance for the joystick with the specified, zero-indexed number.

Definition at line 33 of file joystick.cpp.

4.18.2.3 Joystick::Joystick (std::string *devicePath*)

Initialises an instance for the joystick device specified.

Definition at line 40 of file joystick.cpp.

4.18.2.4 Joystick::Joystick (std::string *devicePath,* bool *blocking*)

Initialises an instance for the joystick device specified and provide the option of blocking I/O.

Definition at line 45 of file joystick.cpp.

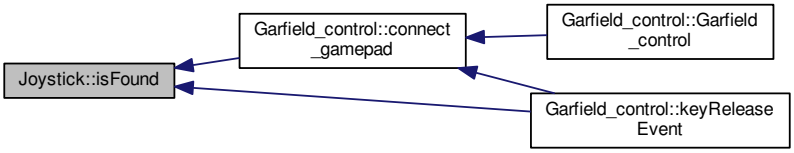
4.18.3 Member Function Documentation

4.18.3.1 bool Joystick::isFound ( )

Returns true if the joystick was found and may be used, otherwise false.

Definition at line 68 of file joystick.cpp.

Here is the caller graph for this function:

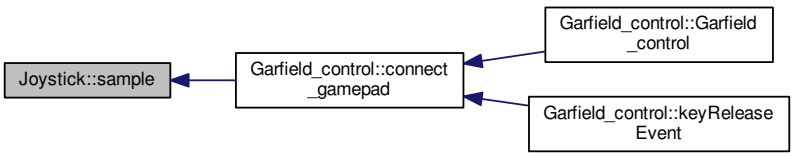


4.18.3.2 bool Joystick::sample (JoystickEvent * *event*)

Attempts to populate the provided JoystickEvent instance with data from the joystick. Returns true if data is available, otherwise false.

Definition at line 56 of file joystick.cpp.

Here is the caller graph for this function:



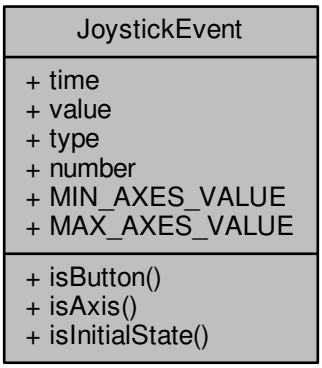
The documentation for this class was generated from the following files:

- joystick.h
- joystick.cpp

4.19 JoystickEvent Class Reference

`#include <joystick.h>`

Collaboration diagram for JoystickEvent:



Public Member Functions

- bool isButton ()
- bool isAxis ()
- bool isInitialState ()

Public Attributes

- unsigned int time
- short value
- unsigned char type
- unsigned char number

Static Public Attributes

- static const short MIN_AXES_VALUE = -32768
- static const short MAX_AXES_VALUE = 32767

Friends

- std::ostream & operator<< (std::ostream &os, const JoystickEvent &e)

4.19.1 Detailed Description

Encapsulates all data relevant to a sampled joystick event.

Definition at line 31 of file joystick.h.

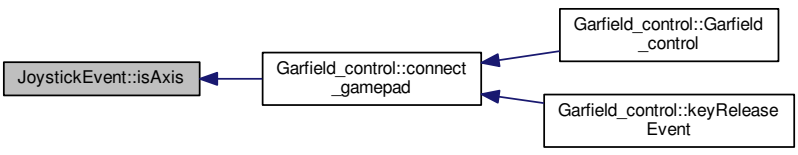
4.19.2 Member Function Documentation

4.19.2.1 bool JoystickEvent::isAxis ( ) [inline]

Returns true if this event is the result of an axis movement.

Definition at line 73 of file joystick.h.

Here is the caller graph for this function:

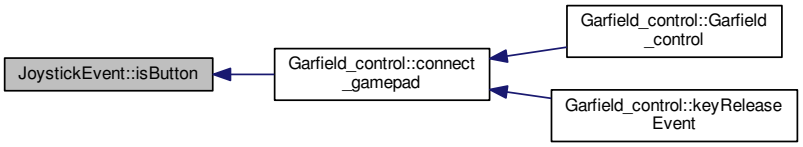


4.19.2.2 bool JoystickEvent::isButton ( ) [inline]

Returns true if this event is the result of a button press.

Definition at line 65 of file joystick.h.

Here is the caller graph for this function:



4.19.2.3 bool JoystickEvent::isInitialState ( ) [inline]

Returns true if this event is part of the initial state obtained when the joystick is first connected to.

Definition at line 82 of file joystick.h.

4.19.3 Friends And Related Function Documentation

4.19.3.1 std::ostream& operator<< (std::ostream & *os,* const JoystickEvent & *e*) [friend]

The ostream inserter needs to be a friend so it can access the internal data structures.

Stream insertion function so you can do this: cout << event << endl;

Definition at line 78 of file joystick.cpp.

4.19.4 Member Data Documentation

4.19.4.1 const short JoystickEvent::MAX_AXES_VALUE = 32767 [static]

Minimum value of axes range

Definition at line 38 of file joystick.h.

4.19.4.2 const short JoystickEvent::MIN_AXES_VALUE = -32768 [static]

Minimum value of axes range

Definition at line 35 of file joystick.h.

4.19.4.3 unsigned char JoystickEvent::number

The axis/button number.

Definition at line 60 of file joystick.h.

4.19.4.4 unsigned int JoystickEvent::time

The timestamp of the event, in milliseconds.

Definition at line 43 of file joystick.h.

4.19.4.5 unsigned char JoystickEvent::type

The event type.

Definition at line 55 of file joystick.h.

4.19.4.6 short JoystickEvent::value

The value associated with this joystick event. For buttons this will be either 1 (down) or 0 (up). For axes, this will range between MIN_AXES_VALUE and MAX_AXES_VALUE.

Definition at line 50 of file joystick.h.

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

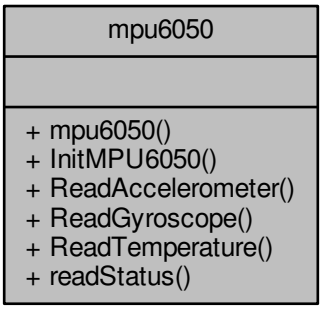
- joystick.h

4.20 mpu6050 Class Reference

represents the mpu6050 hardware device

`#include <mpu6050.hpp>`

Collaboration diagram for mpu6050:



Classes

- struct AccelerometerData

  AccelerometerData.
- struct GyroscopeData

  GyroscopeData.

Public Types

- using temp = float

  typedef for the temperature value

Public Member Functions

- mpu6050 (const MPU6050_Addresses deviceAddress)

  constructs a mpu6050 object with the given address
- alt_u8 InitMPU6050 (const AccelerometerSettings acc_sens, const GyroscopeSettings gyro_sens)

  initializes the mpu with the given settings
- alt_u8 ReadAccelerometer (AccelerometerData &acc_data)

  reads the current acc data
- alt_u8 ReadGyroscope (GyroscopeData &gyro_data)

  reads the current gyro data
- alt_u8 ReadTemperature (temp &temp_data)

  reads the current temperature
- alt_u8 readStatus (void)

  reads the status register and returns the current measurement status (temp, gyro and acc), the register is automatically reseted by the read operation

4.20.1 Detailed Description

represents the mpu6050 hardware device

Definition at line 74 of file mpu6050.hpp.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 mpu6050::mpu6050 (const MPU6050_Addresses *deviceAddress*)

constructs a mpu6050 object with the given address

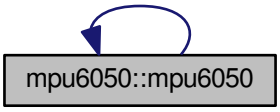
Parameters

in	*deviceAddress*	the used iic address for communication

Definition at line 9 of file mpu6050.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



4.20.3 Member Function Documentation

4.20.3.1 alt_u8 mpu6050::InitMPU6050 (const AccelerometerSettings *acc_sens,* const GyroscopeSettings *gyro_sens*)

initializes the mpu with the given settings

Parameters

in	*acc_sens*	the sensitivity for the accelerometer (between 2G and 16G)
in	*gyro_sens*	the sensitivity for the gyroscope (between 250° and 2000°)

Returns

currently return always 1; the idea was if the iic device acks the address set result to 0, but this mechanism is currently disabled

Definition at line 15 of file mpu6050.cpp.

4.20.3.2 alt_u8 mpu6050::ReadAccelerometer (AccelerometerData & *acc_data*)

reads the current acc data

Parameters

out	*acc_data*	provides the memory buffer for the data

Returns

s.a.

Definition at line 67 of file mpu6050.cpp.

4.20.3.3 alt_u8 mpu6050::ReadGyroscope (GyroscopeData & *gyro_data*)

reads the current gyro data

Parameters

out	*gyro_data*	provides the memory buffer for the data

Returns

s.a.

Definition at line 111 of file mpu6050.cpp.

4.20.3.4 alt_u8 mpu6050::readStatus (void)

reads the status register and returns the current measurement status (temp, gyro and acc), the register is automatically reseted by the read operation

Returns

1: if the current measurement is finished 0: no measurement is finished

Definition at line 175 of file mpu6050.cpp.

4.20.3.5 alt_u8 mpu6050::ReadTemperature (temp & *temp_data*)

reads the current temperature

Parameters

out	*temp_data*	provides the memory buffer for the data

Returns

s.a.

Definition at line 155 of file mpu6050.cpp.

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

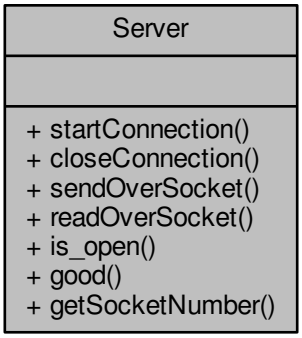
- mpu6050.hpp
- mpu6050.cpp

4.21 Server Class Reference

Represents the serverside of an communication for the whole application.

`#include <Client_Server_impl.hpp>`

Collaboration diagram for Server:



Public Member Functions

- alf_error startConnection (const uint32_t &)

  Trys to open the given port and listen to incoming connections It is using the underlying linux functions for socket handling.
- void closeConnection (void)

  Closing the binded socket and close the server connection.
- alf_error sendOverSocket (const string &)

  Sending the string to over the socket via the underlying linux functaion.
- alf_error readOverSocket (string &s)

  read from the underlying socket
- bool is_open ()

  returns the state of the socket connection
- bool good ()

  dummy function to satisfy the compiler (std::fstream, Server/Client all have the good() function so no explicit type handling must be done
- int32_t getSocketNumber (void)

  returns the socket handler id given from linux at initalisation of the socket

4.21.1 Detailed Description

Represents the serverside of an communication for the whole application.

Attention

at the moment this server implementation can only handle **ONE** connection!

Definition at line 74 of file Client_Server_impl.hpp.

4.21.2 Member Function Documentation

4.21.2.1 int32_t Server::getSocketNumber (void) [inline]

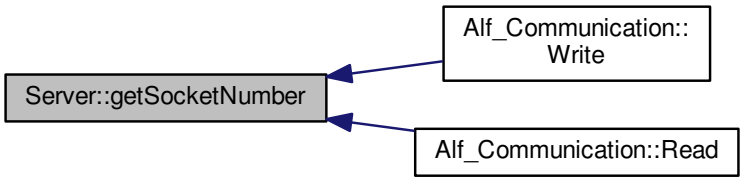
returns the socket handler id given from linux at initalisation of the socket

Returns

the socket handler number

Definition at line 121 of file Client_Server_impl.hpp.

Here is the caller graph for this function:



4.21.2.2 bool Server::is_open ( ) [inline]

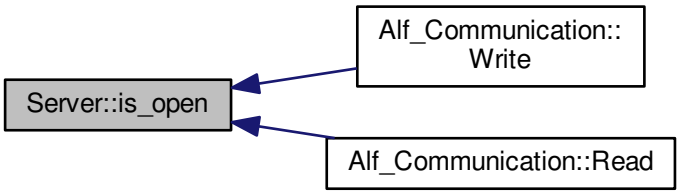
returns the state of the socket connection

Returns

true if connection is good, false otherwise

Definition at line 111 of file Client_Server_impl.hpp.

Here is the caller graph for this function:



4.21.2.3 alf_error Server::readOverSocket (string & *s*)

read from the underlying socket

Parameters

in	*s*	- a string reference

Returns

at this moment -> nothing

Attention

this is just the dummy function, the implementation of this function is missing

Definition at line 128 of file Client_Server_impl.cpp.

4.21.2.4 alf_error Server::sendOverSocket (const string & *data*)

Sending the string to over the socket via the underlying linux functaion.

Parameters

in	*data*	- the string with the message which shall be transmitted

Returns

- ALF_NO_ERROR if the message can be transmitted
- ALF_SOCKET_NOT_READY if the socket is not initialised and
- ALF_CANNOT_SEND_MESSAGE if there are errors in the linux functionalitys, typical triggered by a too long message etc.

Definition at line 150 of file Client_Server_impl.cpp.

4.21.2.5 alf_error Server::startConnection (const uint32_t & *portno*)

Trys to open the given port and listen to incoming connections It is using the underlying linux functions for socket handling.

Parameters

in	*portno*	- the portnumber on which the socket should be opened

Returns

- ALF_SOCKET_SERVER_NOT_READY if something goes wrong (the port is blocked, the function gets no socket handler from os etc.) and

- ALF_NO_ERROR if the port can be catched and the port is working

Definition at line 84 of file Client_Server_impl.cpp.

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

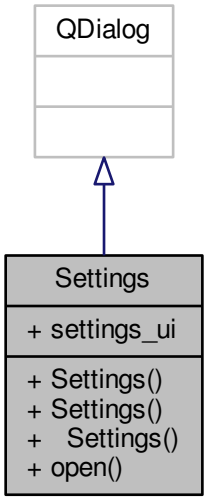
- Client_Server_impl.hpp
- Client_Server_impl.cpp

4.22 Settings Class Reference

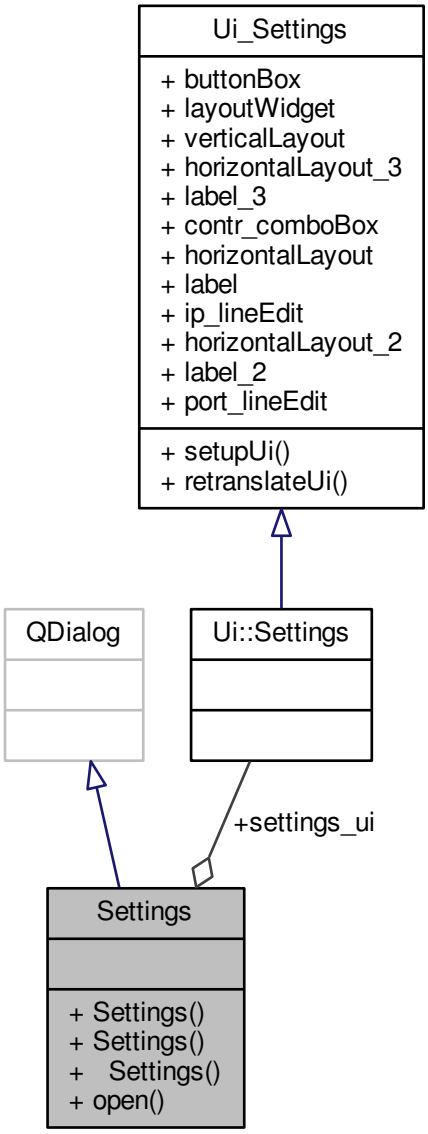
Settings is the settings class for the settings window.

```
#include <Settings.h>
```

Inheritance diagram for Settings:



Collaboration diagram for Settings:



Public Member Functions

- Settings ()

  This is the default constructor.
- Settings (QMainWindow ∗parent, QString IP, QString Port, QString Dev)

  The constructor for creating the settings window.
- ∼Settings ()

  This is the destructor.
- void open ()

  open() this function opens the settings window

Public Attributes

- Ui::Settings * settings_ui

  The settings user interface for setting data in the gui.

4.22.1 Detailed Description

Settings is the settings class for the settings window.

Definition at line 17 of file Settings.h.

4.22.2 Constructor & Destructor Documentation

4.22.2.1 Settings::Settings (QMainWindow * *parent*, QString *IP*, QString *Port*, QString *Dev*)

The constructor for creating the settings window.

Parameters

in	**parent*	- The object of the parent Window
in	*IP*	- The IP which is currently stored in the settings file and should be displayed
in	*Port*	- The Port which is currently stored in the settings file and should be displayed
in	*Dev*	- The device name of the gamepad which is currently saved in the settings file and should be set as active

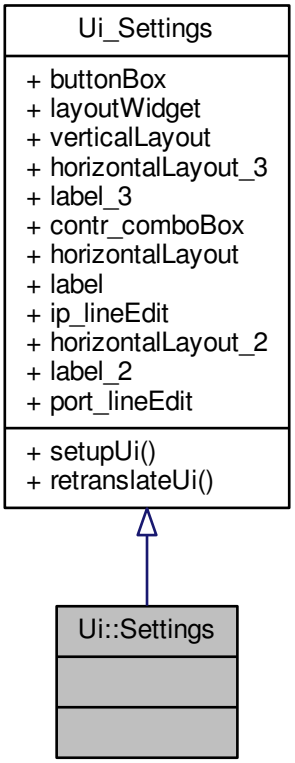
Definition at line 11 of file Settings.cpp.

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

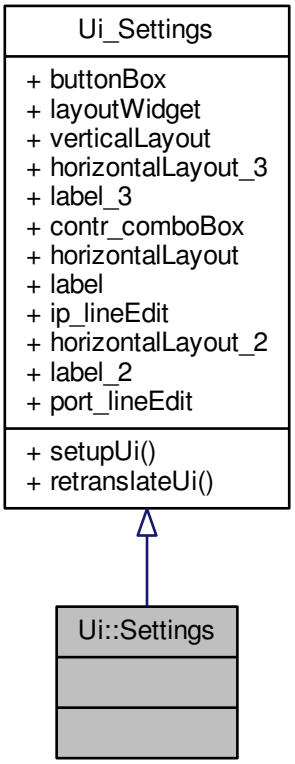
- Settings.h
- Settings.cpp

4.23 Ui::Settings Class Reference

Inheritance diagram for Ui::Settings:



Collaboration diagram for Ui::Settings:



Additional Inherited Members

4.23.1 Detailed Description

Definition at line 126 of file ui_Settings.h.

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

- ui_Settings.h

4.24 Steering Class Reference

```
#include <Steering.hpp>
```

Collaboration diagram for Steering:

Public Member Functions

- Steering ()=delete

Static Public Member Functions

- static void Init (alt_u8 max_angle)
- static void Set (alt_8 angle)

4.24.1 Detailed Description

class Steering for controlling the steering servo. No object is needed because of static functions

Definition at line 16 of file Steering.hpp.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 Steering::Steering ( ) [delete]

Delete the default constructor

4.24.3 Member Function Documentation

4.24.3.1 void Steering::Init (alt_u8 *max_angle*) [static]

Init Function for initializing the Steering with the maximum steering angle

Parameters

max_angle	This is the maximum steering angle in one direction (e.g. 50 deg). If the Set(alt_8 angle) is called with a bigger angle, it is set to max_angle_delta

Definition at line 14 of file Steering.cpp.

4.24.3.2 void Steering::Set (alt_8 *angle*) [static]

Set Function for setting given angle to the servo

Parameters

angle	This is the angle to set the servo (between -max_angle_delta and max_angle_delta)

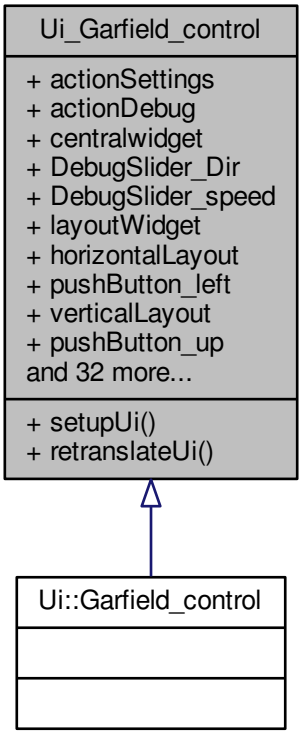
Definition at line 24 of file Steering.cpp.

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

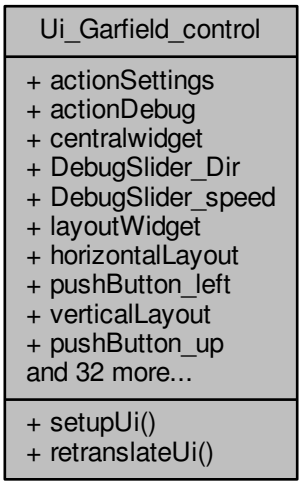
- Steering.hpp
- Steering.cpp

4.25 Ui_Garfield_control Class Reference

Inheritance diagram for Ui_Garfield_control:



Collaboration diagram for Ui_Garfield_control:



Public Member Functions

- void **setupUi** (QMainWindow ∗Garfield_control)
- void **retranslateUi** (QMainWindow ∗Garfield_control)

Public Attributes

- QAction ∗ **actionSettings**
- QAction ∗ **actionDebug**
- QWidget ∗ **centralwidget**
- QSlider ∗ **DebugSlider_Dir**
- QSlider ∗ **DebugSlider_speed**
- QWidget ∗ **layoutWidget**
- QHBoxLayout ∗ **horizontalLayout**
- QPushButton ∗ **pushButton_left**
- QVBoxLayout ∗ **verticalLayout**
- QPushButton ∗ **pushButton_up**
- QPushButton ∗ **pushButton_down**
- QPushButton ∗ **pushButton_right**
- QCheckBox ∗ **checkBox_light**
- QWidget ∗ **layoutWidget1**
- QHBoxLayout ∗ **horizontalLayout_2**
- QLabel ∗ **angle_label**
- QLineEdit ∗ **angle_lineEdit**
- QLabel ∗ **speed_label**
- QLineEdit ∗ **speed_lineEdit**
- QLabel ∗ **AccGrid_label**
- QLabel ∗ **GridPoint_label**

- QLabel * **acc_label**
- QPushButton * **connect_pushButton**
- QLabel * **connstate_label**
- QWidget * **verticalLayoutWidget**
- QVBoxLayout * **verticalLayout_2**
- QHBoxLayout * **horizontalLayout_3**
- QLabel * **Gyro_X_label**
- QLineEdit * **Gyro_X_lineEdit**
- QHBoxLayout * **horizontalLayout_4**
- QLabel * **Gyro_Y_label**
- QLineEdit * **Gyro_Y_lineEdit**
- QHBoxLayout * **horizontalLayout_5**
- QLabel * **Gyro_Z_label**
- QLineEdit * **Gyro_Z_lineEdit**
- QSpacerItem * **verticalSpacer**
- QHBoxLayout * **horizontalLayout_6**
- QLabel * **temperature_label**
- QLineEdit * **temperatur_lineEdit**
- QMenuBar * **menubar**
- QMenu * **menuConfig**
- QStatusBar * **statusBar**

4.25.1 Detailed Description

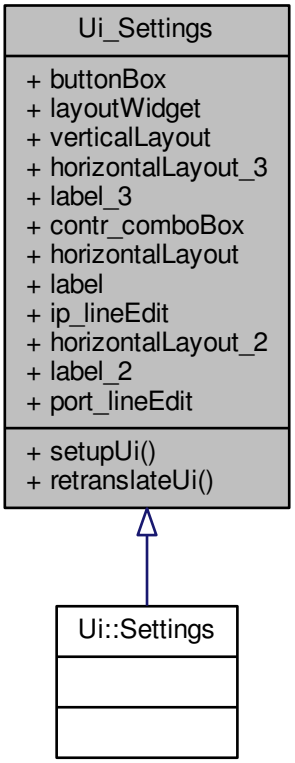
Definition at line 33 of file ui_Garfield_control.h.

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

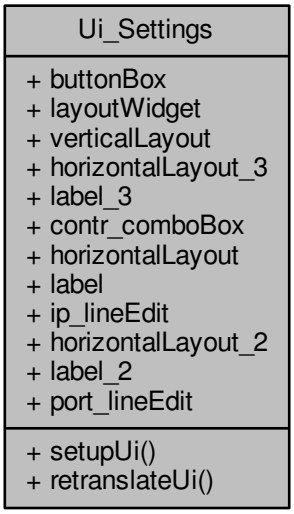
- ui_Garfield_control.h

4.26 Ui_Settings Class Reference

Inheritance diagram for Ui_Settings:



Collaboration diagram for Ui_Settings:



Public Member Functions

- void **setupUi** (QDialog ∗Settings)
- void **retranslateUi** (QDialog ∗Settings)

Public Attributes

- QDialogButtonBox ∗ **buttonBox**
- QWidget ∗ **layoutWidget**
- QVBoxLayout ∗ **verticalLayout**
- QHBoxLayout ∗ **horizontalLayout_3**
- QLabel ∗ **label_3**
- QComboBox ∗ **contr_comboBox**
- QHBoxLayout ∗ **horizontalLayout**
- QLabel ∗ **label**
- QLineEdit ∗ **ip_lineEdit**
- QHBoxLayout ∗ **horizontalLayout_2**
- QLabel ∗ **label_2**
- QLineEdit ∗ **port_lineEdit**

4.26.1 Detailed Description

Definition at line 28 of file ui_Settings.h.

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

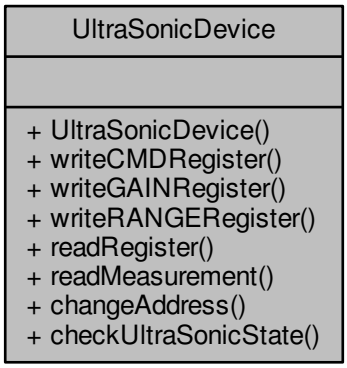
- ui_Settings.h

4.27 UltraSonicDevice Class Reference

represents a ultrasonic hardware device

`#include <ultrasonic.hpp>`

Collaboration diagram for UltraSonicDevice:



Public Member Functions

- UltraSonicDevice (const UltraSonicAddress deviceAddress)
  constructs a ultrasonic device with the given address
- alt_u8 writeCMDRegister (const UltraSonicCommands val, const bool broadcast=false) const
  function to write to the command srf08 register;
- alt_u8 writeGAINRegister (const alt_u8 val) const
  function to write to the gain srf08 register
- alt_u8 writeRANGERegister (const alt_u8 val) const
  function to write to the range srf08 register
- alt_u8 readRegister (const UltraSonicRegisterRead reg, alt_u16 &readPtr) const
  function to read from specific srf08 register (reads always high and low byte if available)
- alt_u8 readMeasurement (alt_u8 *ultrasonic_measurement, const alt_u8 length) const
  function to read one complete range measurement
- alt_u8 changeAddress (const UltraSonicAddress newAddress)
  function to change the IIC address of the ultrasonic devicer
- alt_u8 checkUltraSonicState (bool &check) const
  function to check if the device does currently a ranging

4.27.1 Detailed Description

represents a ultrasonic hardware device

Definition at line 85 of file ultrasonic.hpp.

4.27.2 Member Function Documentation

4.27.2.1 alt_u8 UltraSonicDevice::changeAddress (const UltraSonicAddress *newAddress*)

function to change the IIC address of the ultrasonic devicer

Parameters

in	*newAddress*	the new address that should be given to the device

Returns

result (status) of this operation

Definition at line 105 of file ultrasonic.cpp.

4.27.2.2 alt_u8 UltraSonicDevice::checkUltraSonicState (bool & *check*) const

function to check if the device does currently a ranging

Parameters

out	*check*	will be set to true if ranging is currently ongoing otherwise set to false

Returns

result (status) of this operation

Warning

do not use this function with the RTOS,

Definition at line 133 of file ultrasonic.cpp.

4.27.2.3 alt_u8 UltraSonicDevice::readMeasurement (alt_u8 * *ultrasonic_measurement,* const alt_u8 *length*) const

function to read one complete range measurement

Parameters

out	*ultrasonic_measurement*	buffer to store the current measurement
in	*length*	maximal length to read

Returns

result (status) of this operation

Definition at line 81 of file ultrasonic.cpp.

4.27.2.4 alt_u8 UltraSonicDevice::readRegister (const UltraSonicRegisterRead *reg*, alt_u16 & *readPtr*) const

function to read from specific srf08 register (reads always high and low byte if available)

Parameters

in	*reg*	register to read from
out	*readPtr*	stores the read value from reg

Returns

result (status) of this operation

Definition at line 51 of file ultrasonic.cpp.

4.27.2.5 alt_u8 UltraSonicDevice::writeCMDRegister (const UltraSonicCommands *val*, const bool *broadcast* = `false`) const

function to write to the command srf08 register;

Parameters

in	*val*	value which will be written to reg
in	*broadcast*	if true the command will be sent with address 0x00, which indicates a broadcast

Returns

result (status) of this operation

Definition at line 15 of file ultrasonic.cpp.

4.27.2.6 alt_u8 UltraSonicDevice::writeGAINRegister (const alt_u8 *val*) const

function to write to the gain srf08 register

Parameters

in	*val*	value which will be written to reg

Returns

result (status) of this operation

Definition at line 29 of file ultrasonic.cpp.

4.27.2.7 alt_u8 UltraSonicDevice::writeRANGERegister (const alt_u8 *val*) const

function to write to the range srf08 register

Parameters

in	*val*	value which will be written to reg

Returns

result (status) of this operation

Definition at line 40 of file ultrasonic.cpp.

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

- ultrasonic.hpp
- ultrasonic.cpp

Chapter 5

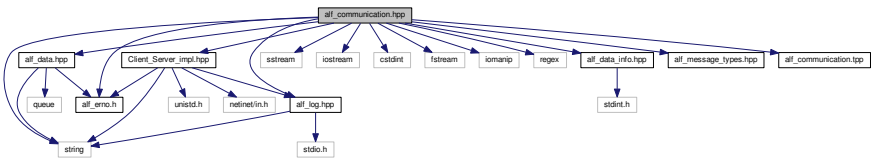
File Documentation

5.1 alf_communication.hpp File Reference

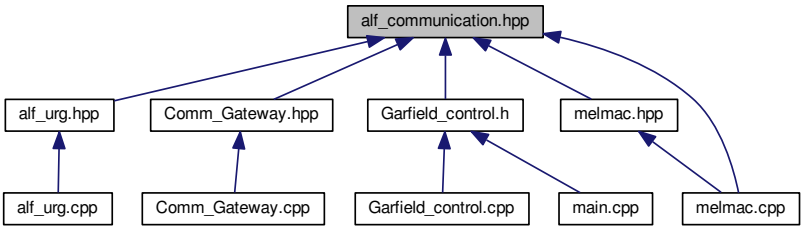
a library for handling all the communication between a client and a server. This file contains all types of communications like writing to files or socket communication over LAN

```
#include <string>
#include <sstream>
#include <iostream>
#include <cstdint>
#include <fstream>
#include <iomanip>
#include <regex>
#include "alf_data.hpp"
#include "alf_data_info.hpp"
#include "alf_log.hpp"
#include "alf_erno.h"
#include "alf_message_types.hpp"
#include "Client_Server_impl.hpp"
#include "alf_communication.tpp"
```

Include dependency graph for alf_communication.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Alf_Communication< _comType >

  CommunicationClass that handles all the communication. Possible template parameters are at the moment std::fstream, Client and Server. No other com-types are supported.

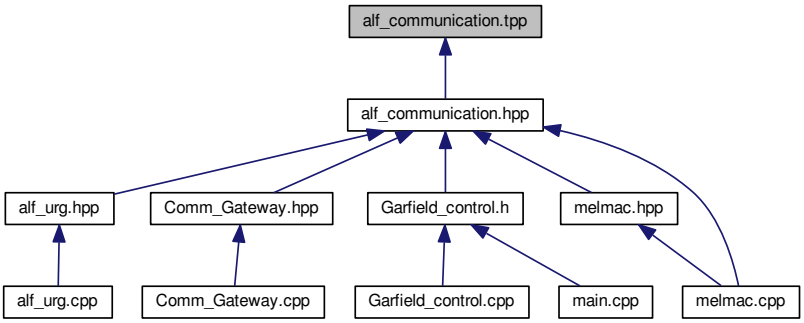
5.1.1 Detailed Description

a library for handling all the communication between a client and a server. This file contains all types of communications like writing to files or socket communication over LAN

5.2 alf_communication.tpp File Reference

contains the implementations for template functions to be outside of the hpp

This graph shows which files directly or indirectly include this file:



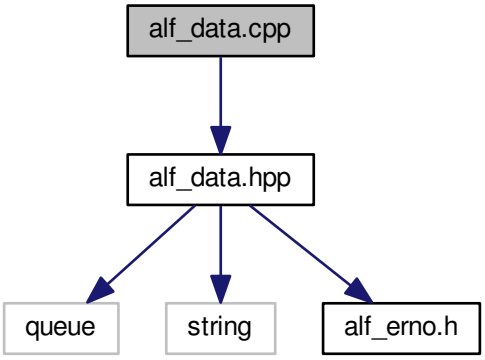
5.2.1 Detailed Description

contains the implementations for template functions to be outside of the hpp

5.3 alf_data.cpp File Reference

```
#include "alf_data.hpp"
```

Include dependency graph for alf_data.cpp:

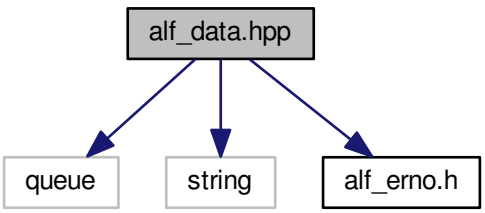


5.4 alf_data.hpp File Reference

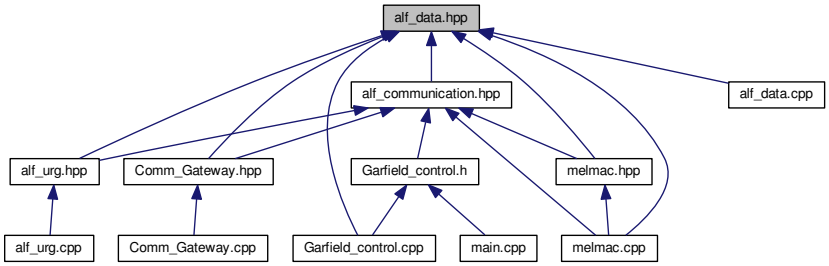
a library for collect all classes which represents any physical data

```
#include <queue>
#include <string>
#include "alf_erno.h"
```

Include dependency graph for alf_data.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Alf_Data

  contains all the data about the vehicle which could be exchanges between the vehicle and other applications so serves as interface between a controller and the hardware

- class Alf_Urg_Measurement

  *This class stands for **one** whole measurement of the laser scanner and provides additional informations It contains all measurement values, also this one, which are invalid in case of the datasheet.*

- class Alf_Urg_Measurements_Buffer

  This buffer can store a set of Alf_Urg_Measurement . It use the std::queue for storing the data and have a maximum size to determine the maximum RAM size which can be used.

Macros

- #define MAX_SIZE_OF_MEASUREMENT_BUFFER_DEFAULT 10

  the number of elements the measurement buffer can store by default.

- #define URG_NUMBER_OF_MEASUREMENT_DATA 768

  number of the measurements the urg_sensors made. These number varies from sensor to sensor, so with another sensor this value must be adjusted

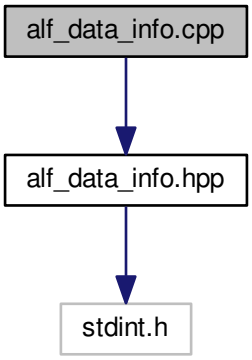
5.4.1 Detailed Description

a library for collect all classes which represents any physical data

5.5 alf_data_info.cpp File Reference

```
#include "alf_data_info.hpp"
```

Include dependency graph for alf_data_info.cpp:



Variables

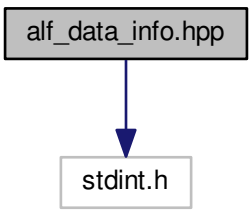
- Alf_Drive_Info global_drive_info {}

  global variables
- Alf_Drive_Command **global_drive_command** {}

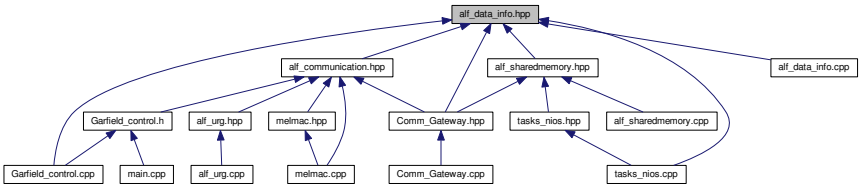
5.6 alf_data_info.hpp File Reference

```
#include "stdint.h"
```

Include dependency graph for alf_data_info.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Alf_Drive_Info

  The Alf_Drive_Info class holds the Infos for steering the Alf.
- class Alf_Drive_Command

Variables

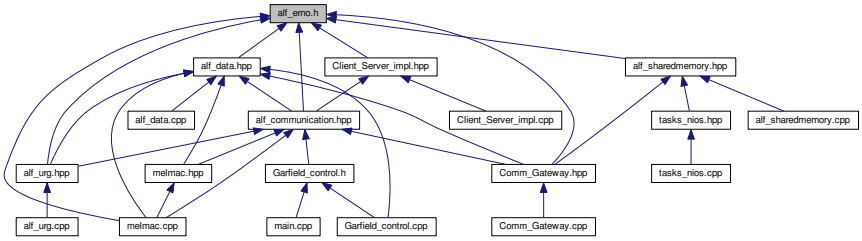
- Alf_Drive_Info global_drive_info

  global variables
- Alf_Drive_Command **global_drive_command**

5.7 alf_erno.h File Reference

contains various means for error coding

This graph shows which files directly or indirectly include this file:



Typedefs

- typedef enum ALF_ERROR_CODES alf_error

  the error codes are available within a type

Enumerations

- enum ALF_ERROR_CODES {
  ALF_BUFFER_READ_IS_WRITE = -100, **ALF_BUFFER_NOTHING_TO_READ**, **ALF_BUFFER_IS_FULL**, **ALF_NOTHING_IN_BUFFER**,
  ALF_NO_COMMUNICATION_FILE, **ALF_IO_ERROR**, **ALF_SOCKET_NOT_READY**, ALF_SOCKET_SERVER_NOT_READY,
  ALF_CANNOT_SEND_MESSAGE, **ALF_CANNOT_READ_SOCKET**, **ALF_NO_WELL_FPGABridge_MAPPING**, **ALF_LOCK_MEMORY_FAILED**,
  ALF_WRITE_SHARED_MEMORY_DISABLED, **ALF_UNKNOWN_ERROR** = -1, ALF_NO_ERROR = 1 }

  contains error codes for all errors which could occur during execution of the application and the information could be interesting for error handling

5.7.1 Detailed Description

contains various means for error coding

5.7.2 Enumeration Type Documentation

5.7.2.1 enum ALF_ERROR_CODES

contains error codes for all errors which could occur during execution of the application and the information could be interesting for error handling

Enumerator

ALF_SOCKET_SERVER_NOT_READY the serverconnection can not be opened, there are some errors in catching the port, opening the file etc.

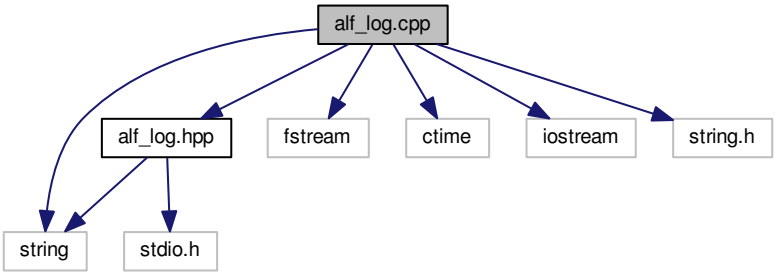
ALF_NO_ERROR alright, there was no error in the functionality

Definition at line 13 of file alf_erno.h.

5.8 alf_log.cpp File Reference

```
#include "alf_log.hpp"
#include <fstream>
#include <ctime>
#include <iostream>
#include <string.h>
#include <string>
```

Include dependency graph for alf_log.cpp:

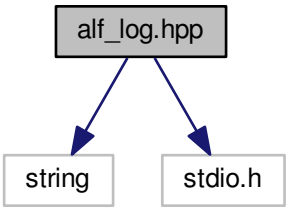


5.9 alf_log.hpp File Reference

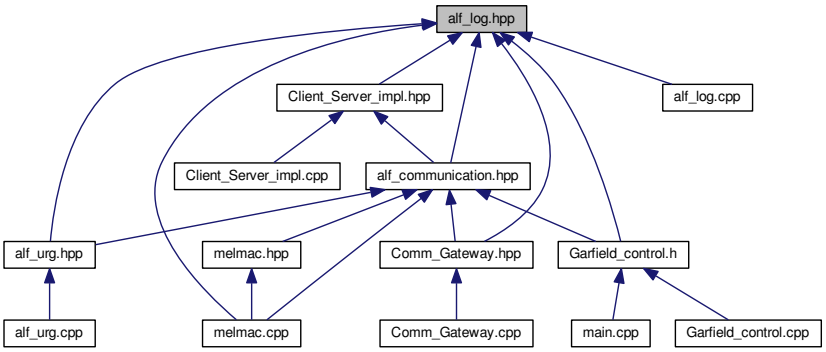
a library give access to log variants and functionality for this

```
#include <string>
#include <stdio.h>
```

Include dependency graph for alf_log.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Alf_Log

  This class handle all the log informations. There will be always a log file, additional the log can be printed to standard output.

Macros

- #define LOG_ENABLE

  LOG_ENABLE does enabling the log, with LOG_DISABLE there are no further log informations.
- #define **ALF_LOG_INIT**(args...) Alf_Log::alf_log_init(args)
- #define **ALF_LOG_WRITE**(args...) Alf_Log::alf_log_write(args)
- #define **ALF_LOG_END**() Alf_Log::alf_log_end()
- #define **ALF_LOG_SET_LEVEL**(a) ALF_Log::alf_set_loglevel(a)

Enumerations

- enum alf_log_level_e { log_error = 0, log_warning, log_info, log_debug }

  all log leves which are available
  the log levels are based on each other, which means, that every log_error is also a log_warning, log_info, log_debug, but a log_info is no log_warning but a log_debug

5.9.1 Detailed Description

a library give access to log variants and functionality for this

5.9.2 Enumeration Type Documentation

5.9.2.1 enum alf_log_level_e

all log leves which are available
the log levels are based on each other, which means, that every log_error is also a log_warning, log_info, log_debug, but a log_info is no log_warning but a log_debug

Enumerator

log_error strongest error, should be used if the desired function of the application could not be provided

log_warning a warning should be used it the execution of the application is in danger, but it is still running

log_info just for info messages, which could be later used in case of errors or warnings to see the control flow etc.

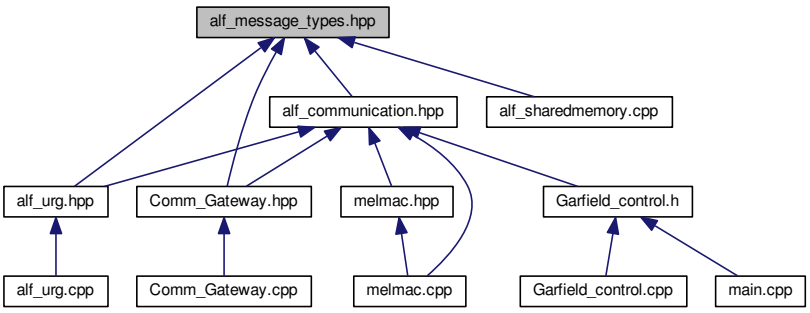
log_debug developer informations

Definition at line 31 of file alf_log.hpp.

5.10 alf_message_types.hpp File Reference

contains enumeration for easy identification of various messages

This graph shows which files directly or indirectly include this file:



Typedefs

- typedef enum ALF_MESSAGE_TYPES **alf_mess_types**

Enumerations

- enum ALF_MESSAGE_TYPES {
  ALF_INIT_ID = 2, ALF_MEASUREMENT_DATA_ID = 1, ALF_DRIVE_COMMAND_ID = 3, ALF_DRIVE_I↩
  NFO_ID = 4,
  ALF_END_ID = 255 }
  contains the IDs for all of the messages which can be sended

5.10.1 Detailed Description

contains enumeration for easy identification of various messages

5.10.2 Enumeration Type Documentation

5.10.2.1 enum ALF_MESSAGE_TYPES

contains the IDs for all of the messages which can be sended

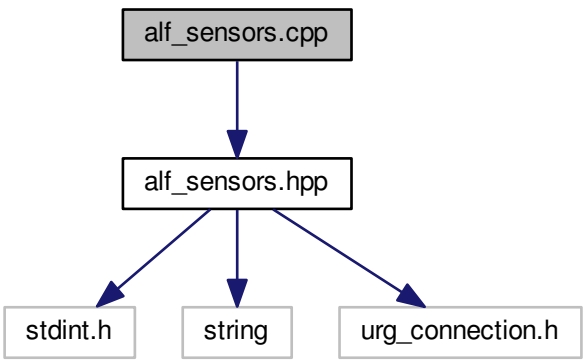
Enumerator

ALF_INIT_ID initalisation data of the laser scanner
ALF_MEASUREMENT_DATA_ID a measurement is sended
ALF_DRIVE_COMMAND_ID ALF drive command.
ALF_DRIVE_INFO_ID ALF drive info.
ALF_END_ID the communication should stop or interrupt now

Definition at line 12 of file alf_message_types.hpp.

5.11 alf_sensors.cpp File Reference

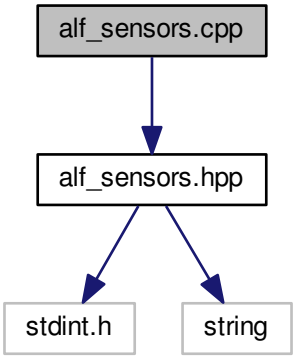
```
#include "alf_sensors.hpp"
```
Include dependency graph for Software_ARM/alf_urg/alf_sensors.cpp:



5.12 alf_sensors.cpp File Reference

```
#include "alf_sensors.hpp"
```

Include dependency graph for common/ARM_HQ/alf_sensors.cpp:

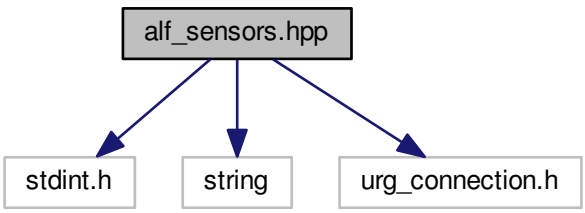


5.13 alf_sensors.hpp File Reference

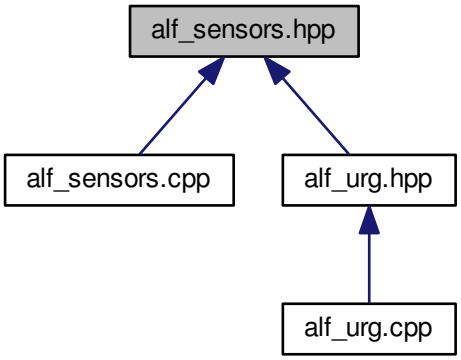
contains datatypes and functionalitys for sensors on the alf vehicle

```
#include <stdint.h>
#include <string>
#include <urg_connection.h>
```

Include dependency graph for Software_ARM/alf_urg/alf_sensors.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Alf_Urg_Sensor

  Represents the laser scanner on the alf vehicle and provide common settings etc.

5.13.1 Detailed Description

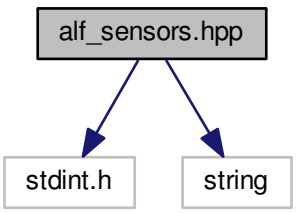
contains datatypes and functionalitys for sensors on the alf vehicle

5.14 alf_sensors.hpp File Reference

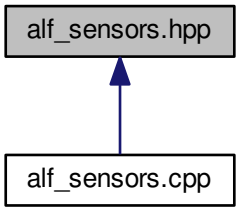
contains datatypes and functionalitys for sensors on the alf vehicle

```
#include <stdint.h>
#include <string>
```

Include dependency graph for common/ARM_HQ/alf_sensors.hpp:



This graph shows which files directly or indirectly include this file:



5.14.1 Detailed Description

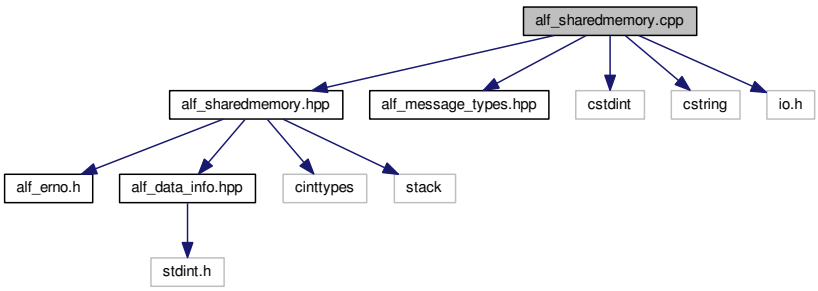
contains datatypes and functionalitys for sensors on the alf vehicle

5.15 alf_sharedmemory.cpp File Reference

Implementation of class to handle communication over hardware shared memory in the garfield fpga project. alf↩ _sharedmemory.cpp.

```
#include "alf_sharedmemory.hpp"
#include "alf_message_types.hpp"
#include <cstdint>
#include <cstring>
#include "io.h"
```

Include dependency graph for alf_sharedmemory.cpp:



Macros

- #define **RW_REGISTER**(reg) ∗(volatile uint32_t∗)(reg)
- #define RAW_NEXT_REG 0x04

  Used to calculate the next register within a 32-bit addressed system. Works only AND only on 32-bit systems!

5.15.1 Detailed Description

Implementation of class to handle communication over hardware shared memory in the garfield fpga project. alf↩ _sharedmemory.cpp.

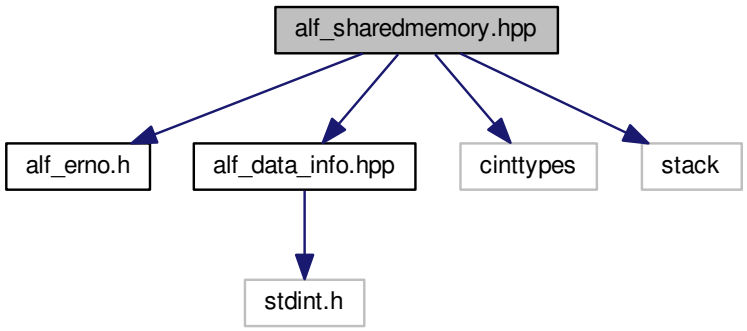
Created on: 02.03.2017 Author: florian

5.16 alf_sharedmemory.hpp File Reference

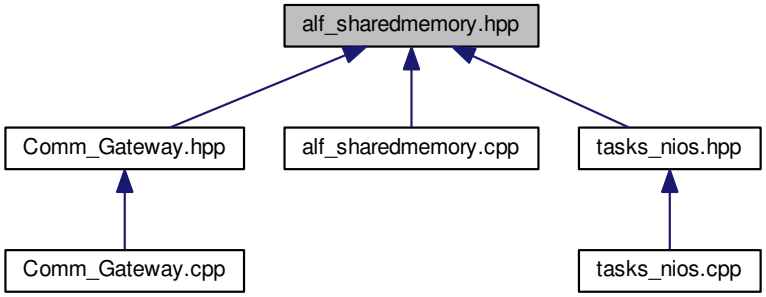
Header file of abstraction class for hardware communication on the hardware shared memory (with mutex and mailbox) in the garfield project. alf_sharedmemory.hpp.

```
#include "alf_erno.h"
#include "alf_data_info.hpp"
#include <cinttypes>
#include <stack>
```

Include dependency graph for alf_sharedmemory.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Garifield_RingBuffer< obj, size >

  Implementation of a ringbuffer with fixed size. If the queue is full, the oldest element will be overwritten.
- class Alf_SharedMemoryComm

  Implementation for communcatiing via a shared memory section on the fpga. Abstraction for the mailbox, the hardware mutex and the shared memory in both directions.

5.16.1 Detailed Description

Header file of abstraction class for hardware communication on the hardware shared memory (with mutex and mailbox) in the garfield project. alf_sharedmemory.hpp.

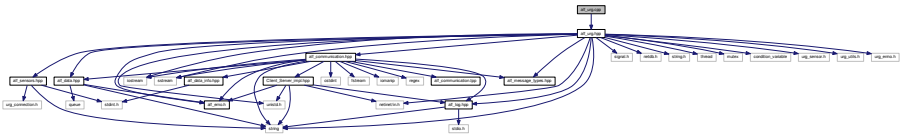
Created on: 02.03.2017 Author: florian

5.17 alf_urg.cpp File Reference

contains the main application to collect measurements from the URG Lidar and offer the collected data in a properitary format other applications

```
#include "alf_urg.hpp"
```

Include dependency graph for alf_urg.cpp:



Macros

- #define **COMMSERVICE** Server
- #define **COMMFILE** 6666
- #define **msleep**(a) usleep(a∗1000)

Functions

- void GetMeasurements ()

  function for collecting data from a urg_sensor and pushing them into a the Alf_Measurements_Buffer
- void ServerConnection ()

  function for sending collected measurement data over the socket connection
- void Stop_Program (int sig)

  dummy function which wake up the main thread from "sleep". This is needed for a clean stop of the programm with a SIGINT of the OS (typical CTRL+C)
- int main ()

  the main process of this application this does

Variables

- Alf_Urg_Measurements_Buffer Alf_Measurements_Buffer (100)
  the buffer with the Size of 100 for all measurements
- std::mutex Alf_Measurements_Buffer_Mutex
  mutex to lock the Alf_Measurements_Buffer
- urg_t urg_sensor
  struct for the ONE connected sensor
- bool Run_Measurement_Task
  control variable for the thread which collects the measurements
- bool Run_Server_Task
  control variable for the thread which handles the communication
- std::condition_variable Run_Main_Task_cond
  variable to let sleep the main thread
- std::mutex Run_Main_Task_mut
  mutex to for the main thread
- Alf_Communication< COMMSERVICE > server_communication
  the communication which shall be handled

5.17.1 Detailed Description

contains the main application to collect measurements from the URG Lidar and offer the collected data in a proper-itary format other applications

5.17.2 Function Documentation

5.17.2.1 void GetMeasurements ( ) [inline]

function for collecting data from a urg_sensor and pushing them into a the Alf_Measurements_Buffer

Attention

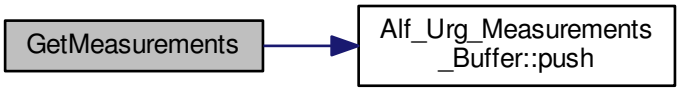
needs a initialized and running urg_sensor, given by the global variable urg_sensor

Note

normally executed as a standalone thread/task

Definition at line 41 of file alf_urg.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.17.2.2 int main ( )

the main process of this application this does

The Main function.

- initializing the urg_sensor
- initializing the server connection
- starting the two threads
- ending the application in a clean way (after CTRL+C)

Definition at line 134 of file alf_urg.cpp.

5.17.2.3 void ServerConnection ( ) [inline]

function for sending collected measurement data over the socket connection

Attention

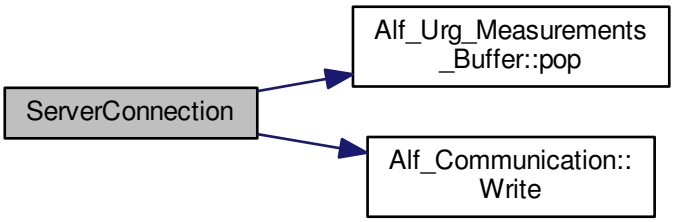
the server connection should be established before calling

Note

normally executed as an own thread

Definition at line 99 of file alf_urg.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.17.2.4 void Stop_Program (int *sig*)

dummy function which wake up the main thread from "sleep". This is needed for a clean stop of the programm with a SIGINT of the OS (typical CTRL+C)

Sig Handler for closing the socket.

Parameters

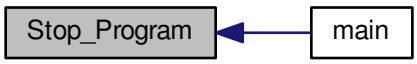
in	*sig*	- SIGINT

Returns

-

Definition at line 122 of file alf_urg.cpp.

Here is the caller graph for this function:

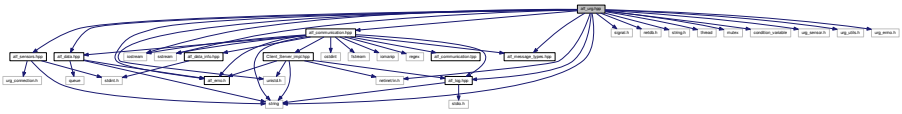


5.18 alf_urg.hpp File Reference

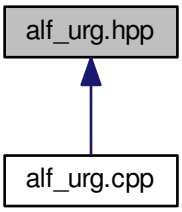
```
#include <iostream>
```

```
#include <sstream>
#include <string>
#include <signal.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
#include <string.h>
#include <thread>
#include <mutex>
#include <condition_variable>
#include "alf_log.hpp"
#include "alf_data.hpp"
#include "alf_erno.h"
#include "alf_communication.hpp"
#include "alf_message_types.hpp"
#include "alf_sensors.hpp"
#include "urg_sensor.h"
#include "urg_utils.h"
#include "urg_errno.h"
```

Include dependency graph for alf_urg.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- int main ()

  the main process of this application this does

5.18.1 Function Documentation

5.18.1.1 int main ( )

the main process of this application this does

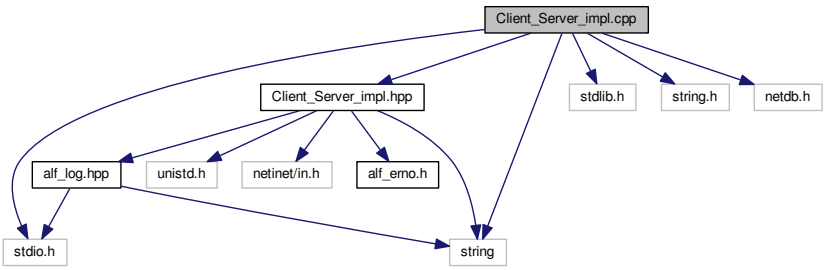
- initializing the urg_sensor
- initializing the server connection
- starting the two threads
- ending the application in a clean way (after CTRL+C)

Definition at line 134 of file alf_urg.cpp.

5.19 Client_Server_impl.cpp File Reference

```
#include "Client_Server_impl.hpp"
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
```

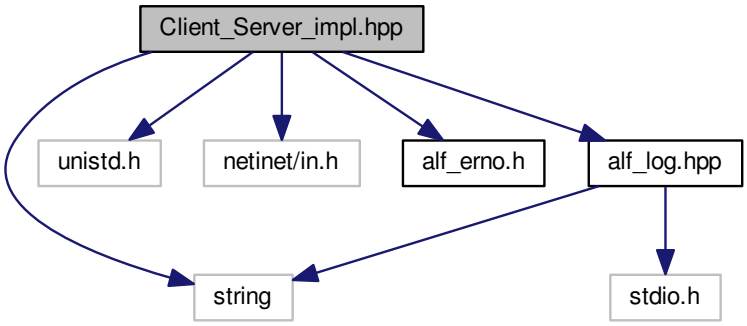
Include dependency graph for Client_Server_impl.cpp:



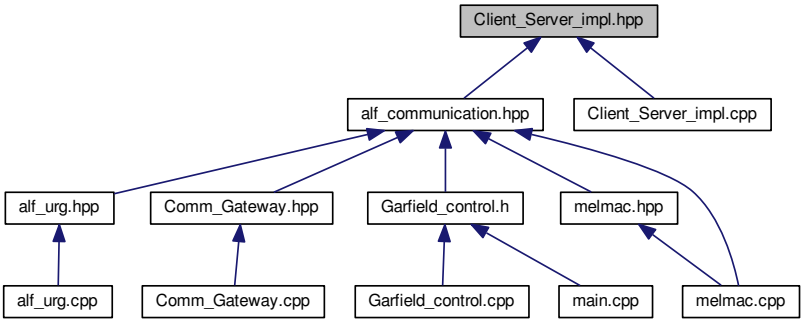
5.20 Client_Server_impl.hpp File Reference

```
#include <string>
#include <unistd.h>
#include <netinet/in.h>
#include "alf_erno.h"
#include "alf_log.hpp"
```

Include dependency graph for Client_Server_impl.hpp:



This graph shows which files directly or indirectly include this file:



Classes

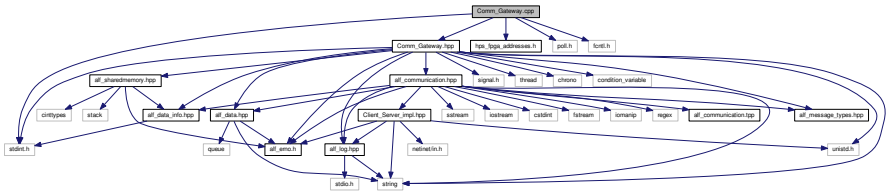
- class Client
- class Server

  Represents the serverside of an communication for the whole application.

5.21 Comm_Gateway.cpp File Reference

```
#include "Comm_Gateway.hpp"
#include "hps_fpga_addresses.h"
#include <stdint.h>
#include <poll.h>
#include <fcntl.h>
```

Include dependency graph for Comm_Gateway.cpp:



Macros

- #define COMPORT 6666

  Port on which socket is created.
- #define COMFREQ 50

  Send/Receive Frequence in Hz.

Functions

- void Stop_Program (int sig)

  Sig Handler for closing the socket.
- void HardwareReadHandler (void)

  This function is for interrupt handling in user mode. It should run in its own thread.
- void writeData (void)

  writeData() Function runs in a thread an writes cyclic the alf_drive_info data in the socket for Garfield_control to display
- void readData (void)

  readData() Function runs in a thread an reads cyclic the alf_drive_command data from the socket to send it over the Mailbox to the NIOS2
- int main ()

  the main process of this application this does

Variables

- Alf_Communication< Server > ServerComm

  Alf Communication Server object.
- std::condition_variable Run_Main_Task_cond

  variable to let sleep the main thread
- std::condition_variable **Run_ServerWrite_Task**
- std::mutex Run_Main_Task_mut

  mutex to for the main thread
- Alf_Log log

  Alf Log.
- Alf_SharedMemoryComm shared_mem

  Shared Memory Mailbox object.
- bool run_threads = true

  Run or close threads.
- bool **notify_ServerWrite_Task** = false
- int **fd**

5.21.1 Function Documentation

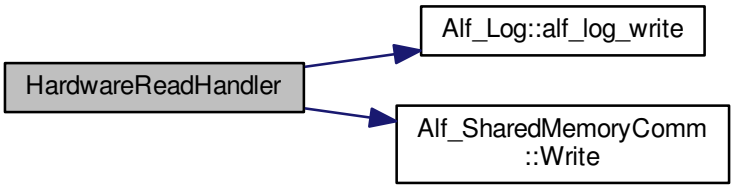
5.21.1.1 void HardwareReadHandler (void)

This function is for interrupt handling in user mode. It should run in its own thread.

This function is for interrupt handling from the hardware mailbox in user mode. It should run in its own thread.

Definition at line 47 of file Comm_Gateway.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.21.1.2 int main ( )

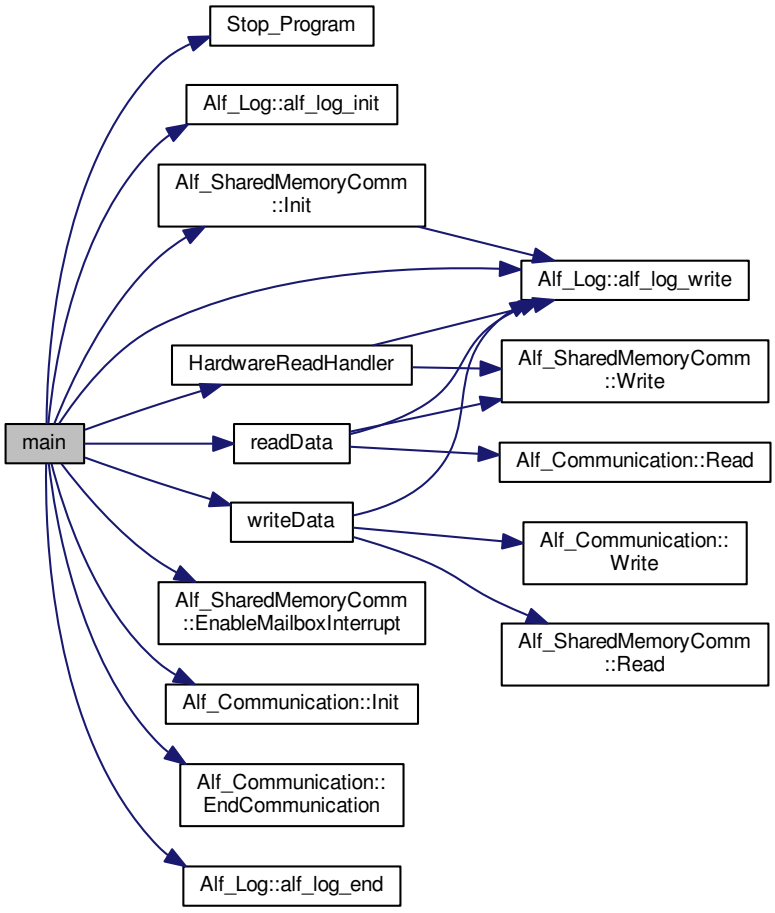
the main process of this application this does

The Main function.

- initializing the urg_sensor
- initializing the server connection
- starting the two threads
- ending the application in a clean way (after CTRL+C)

Definition at line 127 of file Comm_Gateway.cpp.

Here is the call graph for this function:



5.21.1.3 void Stop_Program (int *sig*)

Sig Handler for closing the socket.

Parameters

in	*sig*	- the signal

Sig Handler for closing the socket.

Parameters

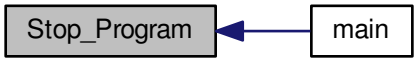
in	*sig*	- SIGINT

Returns

-

Definition at line 39 of file Comm_Gateway.cpp.

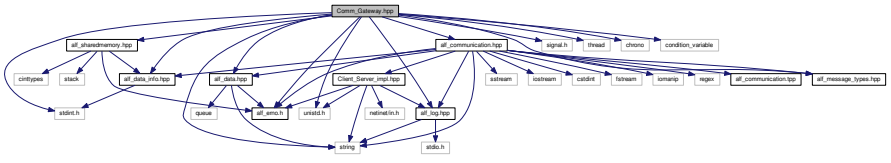
Here is the caller graph for this function:



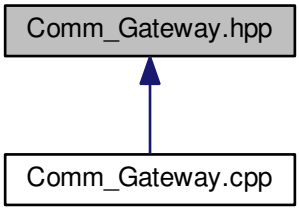
5.22 Comm_Gateway.hpp File Reference

```
#include <string>
#include <unistd.h>
#include <stdint.h>
#include <signal.h>
#include <thread>
#include <chrono>
#include <condition_variable>
#include "alf_log.hpp"
#include "alf_data.hpp"
#include "alf_data_info.hpp"
#include "alf_erno.h"
#include "alf_communication.hpp"
#include "alf_message_types.hpp"
#include "alf_sharedmemory.hpp"
```

Include dependency graph for Comm_Gateway.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- void Stop_Program (int sig)

  Sig Handler for closing the socket.
- void HardwareReadHandler (void)

  This function is for interrupt handling from the hardware mailbox in user mode. It should run in its own thread.
- void writeData (void)

  writeData() Function runs in a thread an writes cyclic the alf_drive_info data in the socket for Garfield_control to display
- void readData (void)

  readData() Function runs in a thread an reads cyclic the alf_drive_command data from the socket to send it over the Mailbox to the NIOS2
- int main ()

  The Main function.

5.22.1 Function Documentation

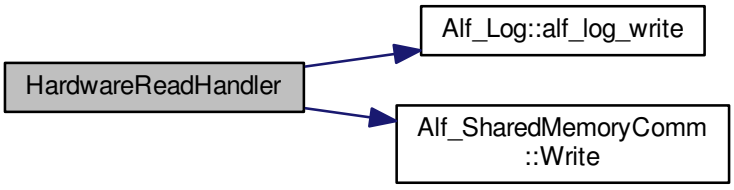
5.22.1.1 void HardwareReadHandler (void)

This function is for interrupt handling from the hardware mailbox in user mode. It should run in its own thread.

This function is for interrupt handling from the hardware mailbox in user mode. It should run in its own thread.

Definition at line 47 of file Comm_Gateway.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.22.1.2 int main ( )

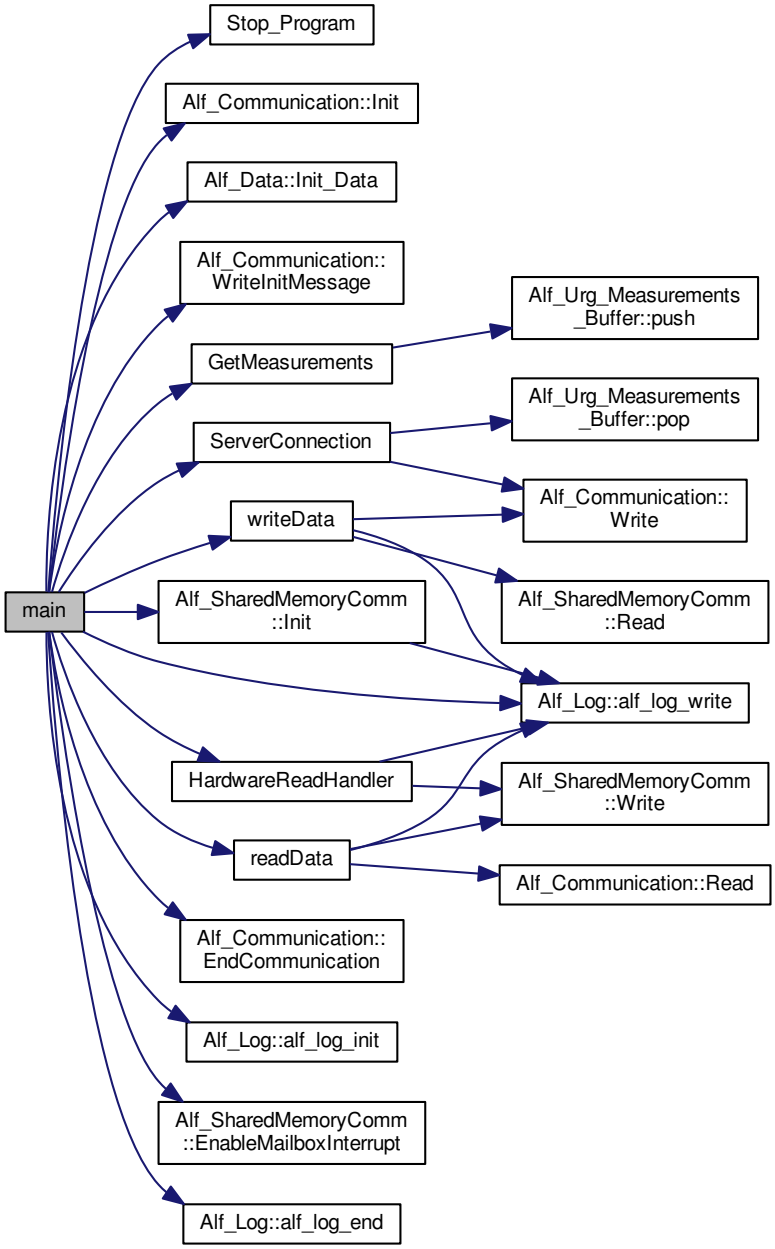
The Main function.

The Main function.

- initializing the urg_sensor
- initializing the server connection
- starting the two threads
- ending the application in a clean way (after CTRL+C)

Definition at line 134 of file alf_urg.cpp.

Here is the call graph for this function:



5.22.1.3 void Stop_Program (int *sig*)

Sig Handler for closing the socket.

Parameters

in	*sig*	- the signal

Sig Handler for closing the socket.

Parameters

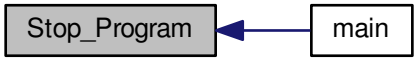
in	*sig*	- SIGINT

Returns

-

Definition at line 122 of file alf_urg.cpp.

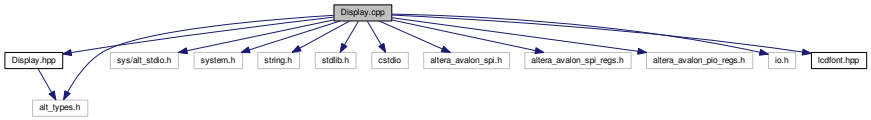
Here is the caller graph for this function:



5.23 Display.cpp File Reference

```
#include "Display.hpp"
#include "alt_types.h"
#include "sys/alt_stdio.h"
#include "system.h"
#include <string.h>
#include <stdlib.h>
#include <cstdio>
#include "altera_avalon_spi.h"
#include "altera_avalon_spi_regs.h"
#include "altera_avalon_pio_regs.h"
#include "io.h"
#include "lcdfont.hpp"
```

Include dependency graph for Display.cpp:



Functions

- void set_tft_dc (bool bit)
- void spi_write_byte (alt_u8 byte)
- void delay_ms (alt_u8 ms)
- void **reversestr** (char s[ ])
- void **itochptr** (int n, char s[ ])

5.23.1 Function Documentation

5.23.1.1 void delay_ms (alt_u8 *ms*)

Delay Function. WARNING: This function is not accurate and needs to be updated

Definition at line 39 of file Display.cpp.

Here is the caller graph for this function:

5.23.1.2 void set_tft_dc (bool *bit*)

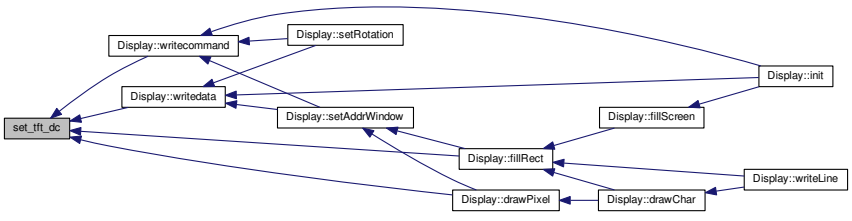
Function for setting the TFT-DC Pin for writing data or commands

Parameters

bit	If set false, DC Bit is set low, for sending a command. If set true, DC Bit is set high, for sending data

Definition at line 23 of file Display.cpp.

Here is the caller graph for this function:



5.23.1.3 void spi_write_byte (alt_u8 *byte*)

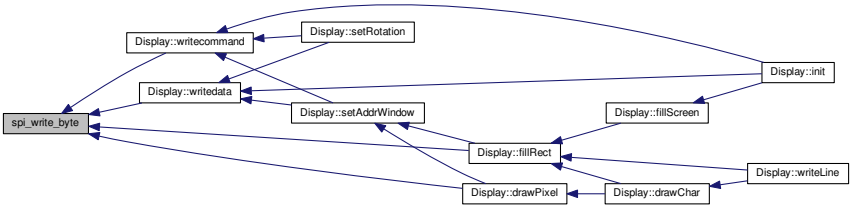
Function for writing 1 byte to the SPI TFT Slave Actually abstracting the generated SPI Function

Parameters

byte	to be sent

Definition at line 35 of file Display.cpp.

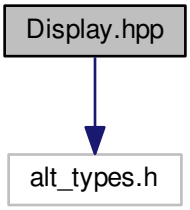
Here is the caller graph for this function:



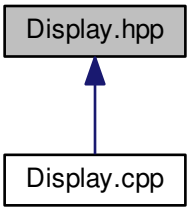
5.24 Display.hpp File Reference

```
#include "alt_types.h"
```

Include dependency graph for Display.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Display

Macros

- #define **ILI9341_TFTWIDTH** 240
- #define **ILI9341_TFTHEIGHT** 320
- #define **ILI9341_NOP** 0x00
- #define **ILI9341_SWRESET** 0x01
- #define **ILI9341_RDDID** 0x04
- #define **ILI9341_RDDST** 0x09
- #define **ILI9341_SLPIN** 0x10
- #define **ILI9341_SLPOUT** 0x11
- #define **ILI9341_PTLON** 0x12
- #define **ILI9341_NORON** 0x13
- #define **ILI9341_RDMODE** 0x0A
- #define **ILI9341_RDMADCTL** 0x0B
- #define **ILI9341_RDPIXFMT** 0x0C
- #define **ILI9341_RDIMGFMT** 0x0D
- #define **ILI9341_RDSELFDIAG** 0x0F
- #define **ILI9341_INVOFF** 0x20
- #define **ILI9341_INVON** 0x21
- #define **ILI9341_GAMMASET** 0x26
- #define **ILI9341_DISPOFF** 0x28
- #define **ILI9341_DISPON** 0x29
- #define **ILI9341_CASET** 0x2A
- #define **ILI9341_PASET** 0x2B
- #define **ILI9341_RAMWR** 0x2C
- #define **ILI9341_RAMRD** 0x2E
- #define **ILI9341_PTLAR** 0x30
- #define **ILI9341_MADCTL** 0x36
- #define **ILI9341_PIXFMT** 0x3A
- #define **ILI9341_FRMCTR1** 0xB1
- #define **ILI9341_FRMCTR2** 0xB2
- #define **ILI9341_FRMCTR3** 0xB3
- #define **ILI9341_INVCTR** 0xB4
- #define **ILI9341_DFUNCTR** 0xB6
- #define **ILI9341_PWCTR1** 0xC0
- #define **ILI9341_PWCTR2** 0xC1
- #define **ILI9341_PWCTR3** 0xC2
- #define **ILI9341_PWCTR4** 0xC3
- #define **ILI9341_PWCTR5** 0xC4
- #define **ILI9341_VMCTR1** 0xC5
- #define **ILI9341_VMCTR2** 0xC7
- #define **ILI9341_RDID1** 0xDA
- #define **ILI9341_RDID2** 0xDB
- #define **ILI9341_RDID3** 0xDC
- #define **ILI9341_RDID4** 0xDD
- #define **ILI9341_GMCTRP1** 0xE0
- #define **ILI9341_GMCTRN1** 0xE1
- #define **ILI9341_BLACK** 0x0000 /∗ 0, 0, 0 ∗/
- #define **ILI9341_NAVY** 0x000F /∗ 0, 0, 128 ∗/
- #define **ILI9341_DARKGREEN** 0x03E0 /∗ 0, 128, 0 ∗/
- #define **ILI9341_DARKCYAN** 0x03EF /∗ 0, 128, 128 ∗/
- #define **ILI9341_MAROON** 0x7800 /∗ 128, 0, 0 ∗/
- #define **ILI9341_PURPLE** 0x780F /∗ 128, 0, 128 ∗/
- #define **ILI9341_OLIVE** 0x7BE0 /∗ 128, 128, 0 ∗/
- #define **ILI9341_LIGHTGREY** 0xC618 /∗ 192, 192, 192 ∗/

- #define **ILI9341_DARKGREY** 0x7BEF /∗ 128, 128, 128 ∗/
- #define **ILI9341_BLUE** 0x001F /∗ 0, 0, 255 ∗/
- #define **ILI9341_GREEN** 0x07E0 /∗ 0, 255, 0 ∗/
- #define **ILI9341_CYAN** 0x07FF /∗ 0, 255, 255 ∗/
- #define **ILI9341_RED** 0xF800 /∗ 255, 0, 0 ∗/
- #define **ILI9341_MAGENTA** 0xF81F /∗ 255, 0, 255 ∗/
- #define **ILI9341_YELLOW** 0xFFE0 /∗ 255, 255, 0 ∗/
- #define **ILI9341_WHITE** 0xFFFF /∗ 255, 255, 255 ∗/
- #define **ILI9341_ORANGE** 0xFD20 /∗ 255, 165, 0 ∗/
- #define **ILI9341_GREENYELLOW** 0xAFE5 /∗ 173, 255, 47 ∗/
- #define **ILI9341_PINK** 0xF81F
- #define **MADCTL_MY** 0x80
- #define **MADCTL_MX** 0x40
- #define **MADCTL_MV** 0x20
- #define **MADCTL_ML** 0x10
- #define **MADCTL_RGB** 0x00
- #define **MADCTL_BGR** 0x08
- #define **MADCTL_MH** 0x04
- #define **CHARSIZE_HEIGHT** 8
- #define **CHARSIZE_WIDTH** 6

Functions

- void set_tft_dc (bool bit)
- void spi_write_byte (alt_u8 byte)
- void delay_ms (alt_u8 ms)
- void **reversestr** (char s[ ])
- void **itochptr** (int n, char s[ ])

5.24.1 Function Documentation

5.24.1.1 void delay_ms (alt_u8 *ms*)

Delay Function. WARNING: This function is not accurate and needs to be updated

Definition at line 39 of file Display.cpp.

Here is the caller graph for this function:

5.24.1.2 void set_tft_dc (bool *bit*)

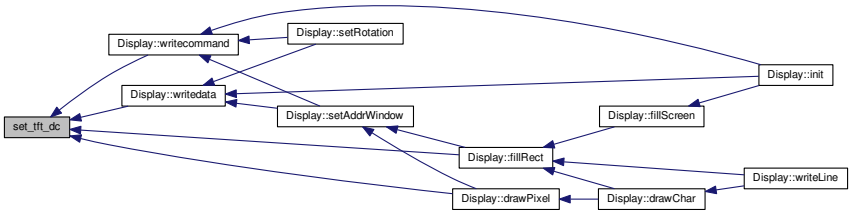
Function for setting the TFT-DC Pin for writing data or commands

Parameters

bit	If set false, DC Bit is set low, for sending a command. If set true, DC Bit is set high, for sending data

Definition at line 23 of file Display.cpp.

Here is the caller graph for this function:



5.24.1.3 void spi_write_byte (alt_u8 *byte*)

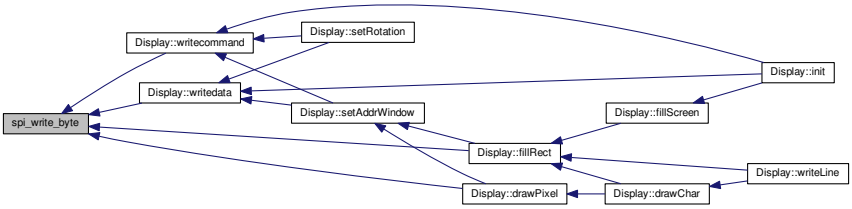
Function for writing 1 byte to the SPI TFT Slave Actually abstracting the generated SPI Function

Parameters

byte	to be sent

Definition at line 35 of file Display.cpp.

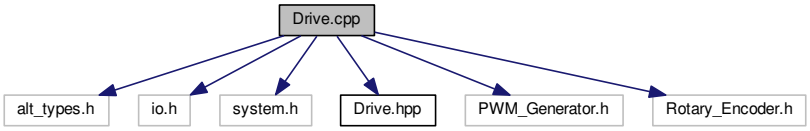
Here is the caller graph for this function:



5.25 Drive.cpp File Reference

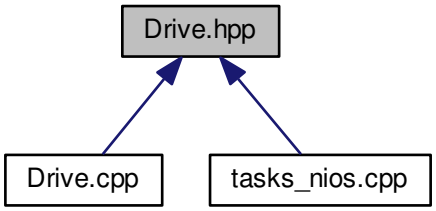
```
#include "alt_types.h"
#include "io.h"
#include "system.h"
#include "Drive.hpp"
#include "PWM_Generator.h"
#include "Rotary_Encoder.h"
```

Include dependency graph for Drive.cpp:



5.26 Drive.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Drive

5.27 Garfield_control.cpp File Reference

```
#include "QSettings"
#include "alf_data.hpp"
#include "alf_data_info.hpp"
#include "joystick.h"
#include "Garfield_control.h"
#include "Settings.h"
#include "ui_Garfield_control.h"
#include "ui_Settings.h"
#include <QtConcurrent>
#include <QThread>
#include <QTimer>
#include <QDebug>
```

Include dependency graph for Garfield_control.cpp:

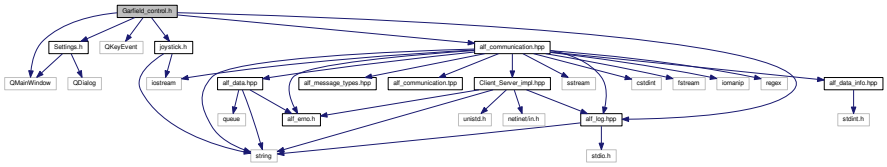
Macros

- #define **ANGLE_MAX_VAL** 90
- #define **ANGLE_MIN_VAL** -90
- #define **SPEED_MAX_VAL** 255
- #define **SPEED_MIN_VAL** 0
- #define **ACC_MAX_VAL** 2.0
- #define **ACC_MIN_VAL** -2.0
- #define **POLLING_GAMEPAD_INTERVAL_MS** 1
- #define **ACC_MAP_UPDATE_MS** 20
- #define **SEND_REC_INTERVAL_MS** 20

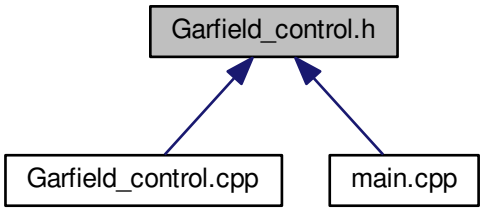
5.28 Garfield_control.h File Reference

```
#include <QMainWindow>
#include "QKeyEvent"
#include "joystick.h"
#include "Settings.h"
#include "alf_communication.hpp"
#include "alf_log.hpp"
```

Include dependency graph for Garfield_control.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Garfield_control

  Garfield_control is the main class that provides all functionalities for the Garfield control program.

Macros

- #define GAMEPAD_BUTTON_TRIANGLE 12

  ID of the gamepad triangle button.
- #define GAMEPAD_BUTTON_CIRCLE 13

  ID of the gamepad circle button.
- #define GAMEPAD_BUTTON_CROSS 14

  ID of the gamepad cross button.
- #define GAMEPAD_BUTTON_SQUARE 15

  ID of the gamepad square button.
- #define GAMEPAD_BUTTON_L1 10

  ID of the gamepad L1 button.
- #define GAMEPAD_BUTTON_R1 11

  ID of the gamepad R1 button.
- #define GAMEPAD_BUTTON_DPAD_UP 4

  ID of the gamepad direction pad up button.
- #define GAMEPAD_BUTTON_DPAD_RIGHT 5

  ID of the gamepad direction pad right button.
- #define GAMEPAD_BUTTON_DPAD_DOWN 6

  ID of the gamepad direction pad down button.
- #define GAMEPAD_BUTTON_DPAD_LEFT 7

  ID of the gamepad direction pad left button.
- #define GAMEPAD_BUTTON_SELECT 0

  ID of the gamepad select button.
- #define GAMEPAD_BUTTON_START 3

  ID of the gamepad start button.
- #define GAMEPAD_BUTTON_ANALOG_LEFT 1

  ID of the gamepad analog left button.
- #define GAMEPAD_BUTTON_ANALOG_RIGHT 2

  ID of the gamepad analog right button.
- #define GAMEPAD_AXIS_ANALOG_LEFT_LR 0

  ID of the gamepad analog left axis from left to right.
- #define GAMEPAD_AXIS_ANALOG_LEFT_UD 1

  ID of the gamepad analog left axis from up to down.
- #define GAMEPAD_AXIS_ANALOG_RIGHT_LR 2

  ID of the gamepad analog right axis from left to right.
- #define GAMEPAD_AXIS_ANALOG_RIGHT_UD 3

  ID of the gamepad analog right axis from up to down.
- #define GAMEPAD_AXIS_L2 12

  ID of the gamepad analog left axis L2.
- #define GAMEPAD_AXIS_R2 13

  ID of the gamepad analog left axis R2.
- #define GAMEPAD_BUTTON_DOWN 1

  Gamepad value button down.
- #define GAMEPAD_BUTTON_UP 0

  Gamepad value button up.
- #define GAMEPAD_AXIS_DOWN 32767

  Gamepad value axis down.
- #define GAMEPAD_AXIS_UP -32768

  Gamepad value axis up.

Functions

- template<typename T >
  T norm_value (T in_min, T in_max, T out_min, T out_max, T value)

  Function normalizes values from given intervall to given intervall.

5.28.1 Function Documentation

5.28.1.1 template<typename T > T norm_value (T *in_min*, T *in_max*, T *out_min*, T *out_max*, T *value*)

Function normalizes values from given intervall to given intervall.

Parameters

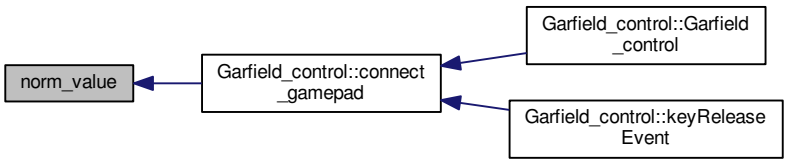
in	*in_min*	- This is the minimal value of the originally intervall
in	*in_max*	- This is the maximal value of the originally intervall
in	*out_min*	- This is the minimal value of the destination intervall
in	*value*	- This is the value to convert to the destination intervall

Returns

the converted value is returned

Definition at line 77 of file Garfield_control.h.

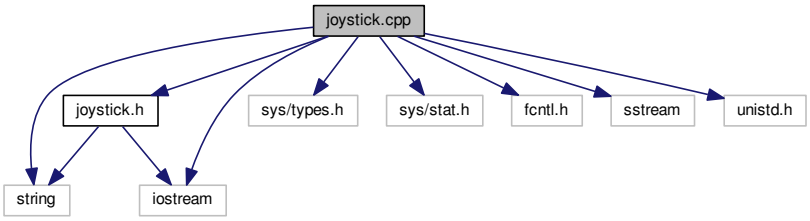
Here is the caller graph for this function:



5.29 joystick.cpp File Reference

```
#include "joystick.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <iostream>
#include <string>
#include <sstream>
#include "unistd.h"
```

Include dependency graph for joystick.cpp:



Functions

- std::ostream & operator<< (std::ostream &os, const JoystickEvent &e)

5.29.1 Function Documentation

5.29.1.1 std::ostream& operator<< (std::ostream & *os,* const JoystickEvent & *e*)

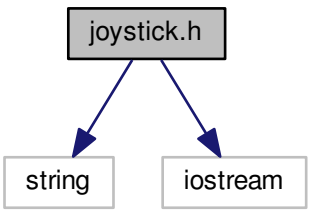
The ostream inserter needs to be a friend so it can access the internal data structures.

Definition at line 78 of file joystick.cpp.

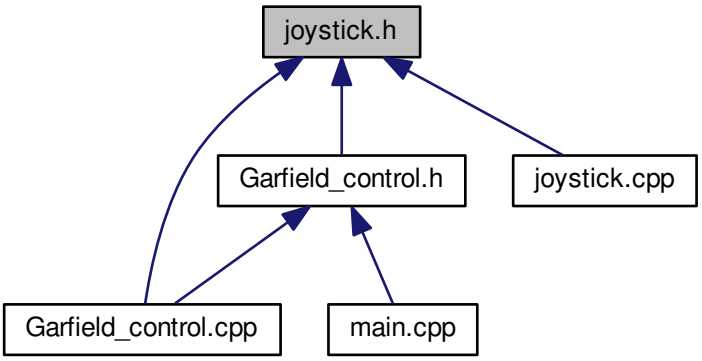
5.30 joystick.h File Reference

```
#include <string>
#include <iostream>
```

Include dependency graph for joystick.h:



This graph shows which files directly or indirectly include this file:



Classes

- class JoystickEvent
- class Joystick

Macros

- #define **JS_EVENT_BUTTON** 0x01
- #define **JS_EVENT_AXIS** 0x02
- #define **JS_EVENT_INIT** 0x80

Functions

- std::ostream & operator<< (std::ostream &os, const JoystickEvent &e)

5.30.1 Function Documentation

5.30.1.1 std::ostream& operator<< (std::ostream & *os,* const JoystickEvent & *e*)

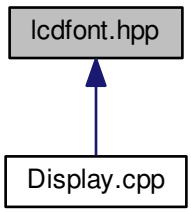
Stream insertion function so you can do this: cout << event << endl;

The ostream inserter needs to be a friend so it can access the internal data structures.

Definition at line 78 of file joystick.cpp.

5.31 lcdfont.hpp File Reference

This graph shows which files directly or indirectly include this file:

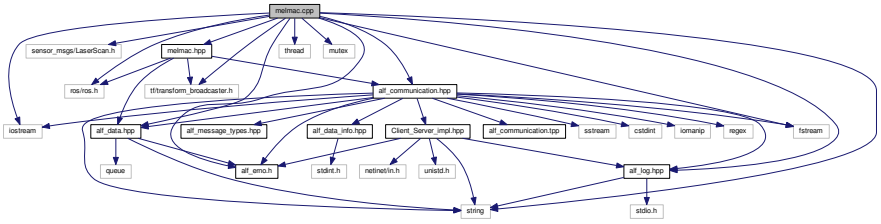


5.32 melmac.cpp File Reference

contains the main application for wrapping data which are collected with the alf_urg application and sended to this client

```
#include <ros/ros.h>
#include <sensor_msgs/LaserScan.h>
#include <tf/transform_broadcaster.h>
#include <iostream>
#include <string>
#include <fstream>
#include <thread>
#include <mutex>
#include "melmac.hpp"
#include "alf_erno.h"
#include "alf_data.hpp"
#include "alf_log.hpp"
#include "alf_communication.hpp"
```

Include dependency graph for melmac.cpp:



Macros

- #define BUF_SIZE 1322
- #define LIDAR_FREQ 10
- #define ANGLE_INC 0.006136
- #define TIME_INC 0.000098

Functions

- void rvizWrapper (ros::NodeHandle ∗n, ros::Publisher ∗scan_pub, tf::TransformBroadcaster ∗broadcaster, ros::Rate ∗r)

  This function represents the sendThread.
- void readStreamingData (void)

  function for reading the measurement data from the socket connection. If and end message was read the function returns and the user can end or reopen the communication
- int main (int argc, char ∗∗argv)

  Main function of rviz_wrapper.

5.32.1 Detailed Description

contains the main application for wrapping data which are collected with the alf_urg application and sended to this client

Attention

can only be build within a working ROS environment

5.32.2 Macro Definition Documentation

5.32.2.1 #define ANGLE_INC 0.006136

Better working ANGLE_INC which works better than the commented calculation

Definition at line 29 of file melmac.cpp.

5.32.2.2 #define BUF_SIZE 1322

This defines the size of AlfMeasBuffer

Definition at line 25 of file melmac.cpp.

5.32.2.3 #define LIDAR_FREQ 10

The frequence of the Lidar. It is needed for the ros loop and scan_time

Definition at line 27 of file melmac.cpp.

5.32.2.4 #define TIME_INC 0.000098

Better working TIME_INC which works better than the commented calculation

Definition at line 30 of file melmac.cpp.

5.32.3 Function Documentation

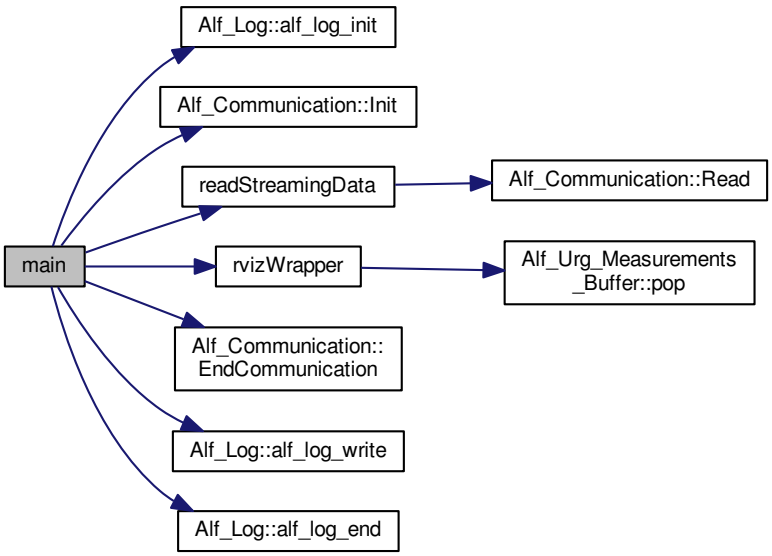
5.32.3.1 int main (int *argc,* char ** *argv*)

Main function of rviz_wrapper.

It opens the socket communication, starts the two threads (readThread and sendThread) etc.

Definition at line 125 of file melmac.cpp.

Here is the call graph for this function:



5.32.3.2 void readStreamingData (void)

function for reading the measurement data from the socket connection. If and end message was read the function returns and the user can end or reopen the communication

Parameters

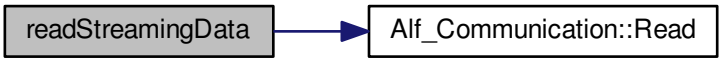
in	-	

Returns

-

Definition at line 93 of file melmac.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.32.3.3 void rvizWrapper (ros::NodeHandle * *n*, ros::Publisher * *scan_pub*, tf::TransformBroadcaster * *broadcaster*, ros::Rate * *r*)

This function represents the sendThread.

It takes all data from Alf Measurement Buffer and maps the data to the ros data structure

Parameters

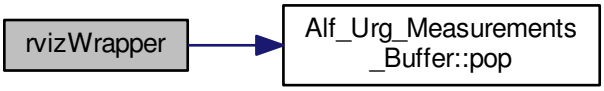
in	*n*	is the nodehandler which checks the status
in	*scan_pub*	is the Scan Publisher which sends all data to rviz
in	*broadcaster*	is the broadcaster to send tf messages to rviz
in	*r*	is necessary for creating a ros loop with the frequence of the lidar (here: 10 Hz)

Returns

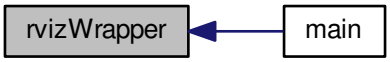
void

Definition at line 42 of file melmac.cpp.

Here is the call graph for this function:



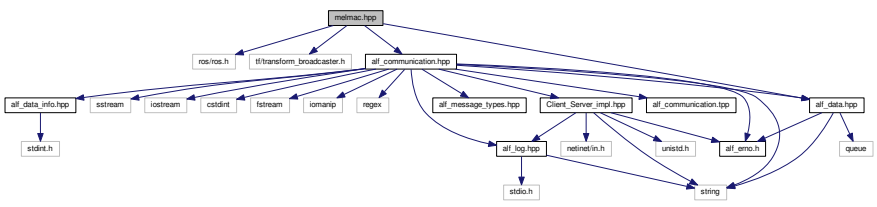
Here is the caller graph for this function:



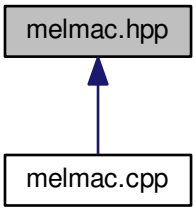
5.33 melmac.hpp File Reference

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
#include "alf_data.hpp"
#include "alf_communication.hpp"
```

Include dependency graph for melmac.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- void rvizWrapper (ros::NodeHandle ∗n, ros::Publisher ∗scan_pub, tf::TransformBroadcaster ∗broadcaster, ros::Rate ∗r)

  This function represents the sendThread.
- void readStreamingData (void)

  function for reading the measurement data from the socket connection. If and end message was read the function returns and the user can end or reopen the communication
- int main (int argc, char ∗∗argv)

  Main function of rviz_wrapper.

5.33.1 Detailed Description

All global variables, defines and the two functions which represents the threads are declared here

5.33.2 Function Documentation

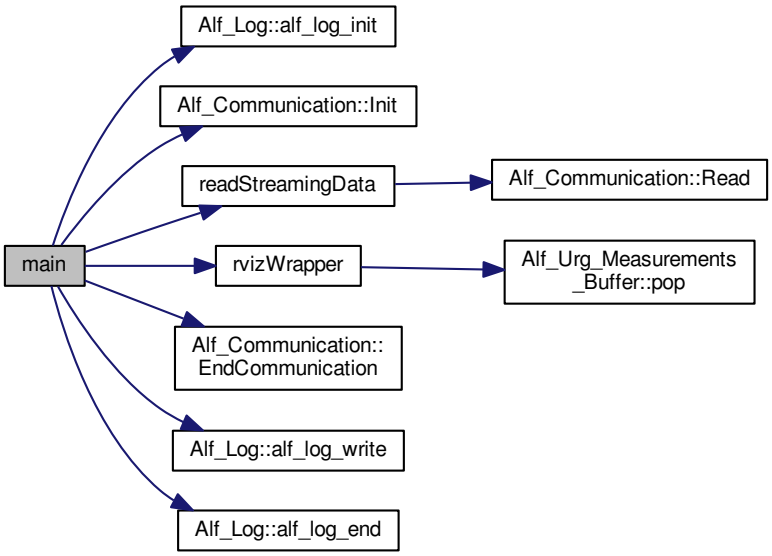
5.33.2.1 int main (int *argc,* char ∗∗ *argv*)

Main function of rviz_wrapper.

It opens the socket communication, starts the two threads (readThread and sendThread) etc.

Definition at line 125 of file melmac.cpp.

Here is the call graph for this function:



5.33.2.2 void readStreamingData (void)

function for reading the measurement data from the socket connection. If and end message was read the function returns and the user can end or reopen the communication

Parameters

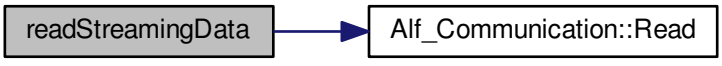
in	-	

Returns

-

Definition at line 93 of file melmac.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

5.33.2.3 void rvizWrapper (ros::NodeHandle * *n*, ros::Publisher * *scan_pub*, tf::TransformBroadcaster * *broadcaster*, ros::Rate * *r*)

This function represents the sendThread.

It takes all data from Alf Measurement Buffer and maps the data to the ros data structure

Parameters

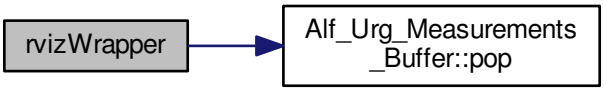
in	*n*	is the nodehandler which checks the status
in	*scan_pub*	is the Scan Publisher which sends all data to rviz
in	*broadcaster*	is the broadcaster to send tf messages to rviz
in	*r*	is necessary for creating a ros loop with the frequence of the lidar (here: 10 Hz)

Returns

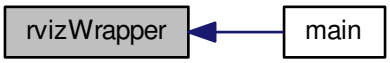
void

Definition at line 42 of file melmac.cpp.

Here is the call graph for this function:



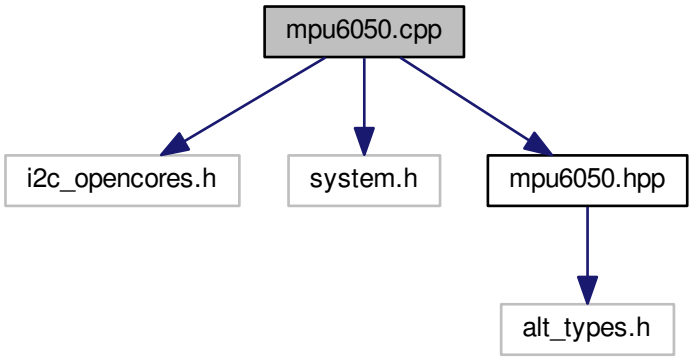
Here is the caller graph for this function:



5.34 mpu6050.cpp File Reference

```
#include "i2c_opencores.h"
#include "system.h"
#include "mpu6050.hpp"
```

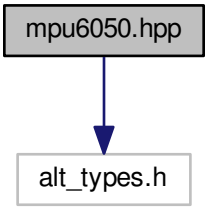
Include dependency graph for mpu6050.cpp:



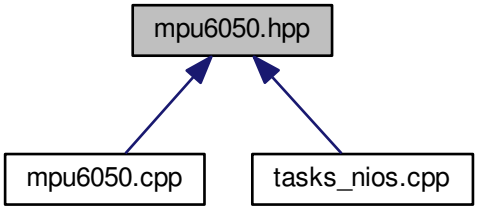
5.35 mpu6050.hpp File Reference

```
#include "alt_types.h"
```

Include dependency graph for mpu6050.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class mpu6050

  represents the mpu6050 hardware device
- struct mpu6050::AccelerometerData

  AccelerometerData.
- struct mpu6050::GyroscopeData

  GyroscopeData.

Enumerations

- enum MPU6050_Register : alt_u8 {
  SMPRT_DIV = 0x19, **CONFIG** = 0x1A, **GYRO_CONFIG** = 0x1B, **ACCEL_CONFIG** = 0x1C,
  INT_PIN_CFG = 0x37, **INT_ENABLE** = 0x38, **INT_STATUS** = 0x3A, **ACCEL_XOUT_H** = 0x3B,
  ACCEL_XOUT_L = 0x3C, **ACCEL_YOUT_H** = 0x3D, **ACCEL_YOUT_L** = 0x3E, **ACCEL_ZOUT_H** = 0x3F,
  ACCEL_ZOUT_L = 0x40, **TEMP_OUT_H** = 0x41, **TEMP_OUT_L** = 0x42, **GYRO_XOUT_H** = 0x43,
  GYRO_XOUT_L = 0x44, **GYRO_YOUT_H** = 0x45, **GYRO_YOUT_L** = 0x46, **GYRO_ZOUT_H** = 0x47,
  GYRO_ZOUT_L = 0x48, **SIGNAL_PATH_RESET** = 0x68, **USER_CTRL** = 0x6A, **PWR_MGMT_1** = 0x6B,
  PWR_MGMT_2 = 0x6C, **FIFO_COUNT_H** = 0x72, **FIFO_COUNT_L** = 0x73, **FIFO_R_W** = 0x74,
  WHO_AM_I = 0x75 }

defines all possible mpu6050 registers reset values are 0x00, except PWR_MGMT_1 = 0x40 and WHO_AM_I = 0x68

- enum MPU6050_Addresses : alt_u8 { **DEVICE_0** = 0xD0, **DEVICE_1** = 0xD2 }

  defines the two possible default i2c addresses (hardware setting)

- enum AccelerometerSettings : alt_u8 { **RANGE_2G** = 0x00, **RANGE_4G** = 0x08, **RANGE_8G** = 0x10, **RANGE_16G** = 0x18 }

  defines the possible accelerometer register settings

- enum GyroscopeSettings : alt_u8 { **RANGE_250_DEG** = 0x00, **RANGE_500_DEG** = 0x08, **RANGE_1000_DEG** = 0x10, **RANGE_2000_DEG** = 0x18 }

  defines the possible gyroscope register settings

5.36 Settings.cpp File Reference

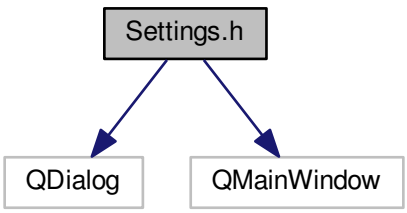
```
#include "Settings.h"
#include "ui_Settings.h"
#include <QProcess>
#include <QDebug>
```

Include dependency graph for Settings.cpp:

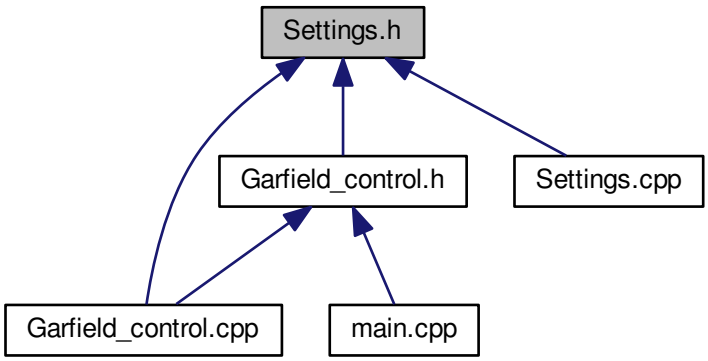
5.37 Settings.h File Reference

```
#include <QDialog>
#include <QMainWindow>
```

Include dependency graph for Settings.h:



This graph shows which files directly or indirectly include this file:



Classes

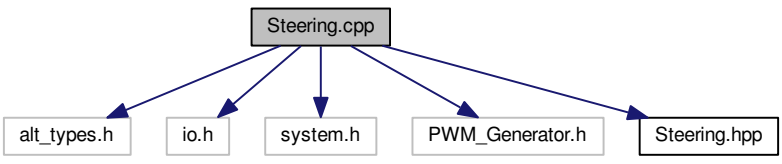
- class Settings

  Settings is the settings class for the settings window.

5.38 Steering.cpp File Reference

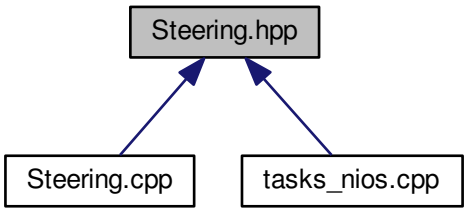
```
#include "alt_types.h"
#include "io.h"
#include "system.h"
#include "PWM_Generator.h"
#include "Steering.hpp"
```

Include dependency graph for Steering.cpp:



5.39 Steering.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Steering

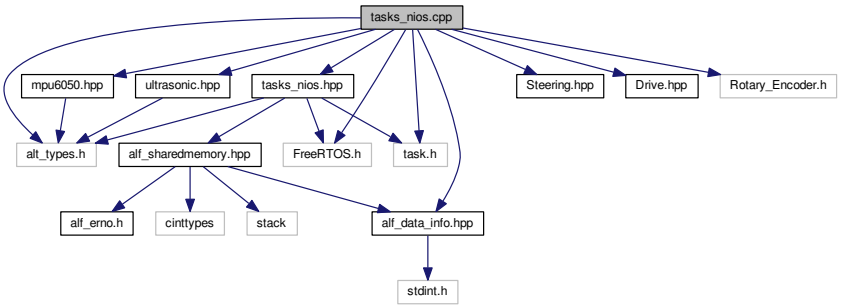
Macros

- #define **MAX_STEERING_ANGLE** 60
- #define **NEUTRAL_POS_VALUE** 51

5.40 tasks_nios.cpp File Reference

```
#include "tasks_nios.hpp"
#include "alt_types.h"
#include "mpu6050.hpp"
#include "ultrasonic.hpp"
#include "Steering.hpp"
#include "Drive.hpp"
#include "alf_data_info.hpp"
#include "Rotary_Encoder.h"
#include "FreeRTOS.h"
#include "task.h"
```

Include dependency graph for tasks_nios.cpp:



Functions

- void **readMPU** (void *p)
- void **readUltraSonic** (void *p)
- void **readRotary** (void *p)
- void **setMotor_and_Steering** (void *p)
- void **setDriveInfo** (void *p)
- void **Mailbox_isr** (void *ptr, alt_u32 a)

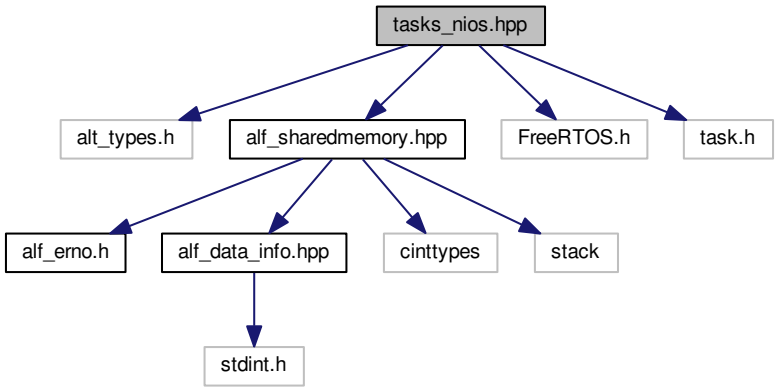
Variables

- Alf_SharedMemoryComm **sharedMem** {}
- TaskHandle_t **writeTask**

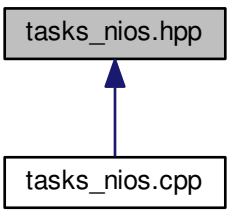
5.41 tasks_nios.hpp File Reference

```
#include "alt_types.h"
#include "alf_sharedmemory.hpp"
#include "FreeRTOS.h"
#include "task.h"
```

Include dependency graph for tasks_nios.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- void **readUltraSonic** (void *p)
- void **readMPU** (void *p)
- void **readRotary** (void *p)
- void **setMotor_and_Steering** (void *p)
- void **setDriveInfo** (void *p)
- void **Mailbox_isr** (void *ptr, alt_u32 a)

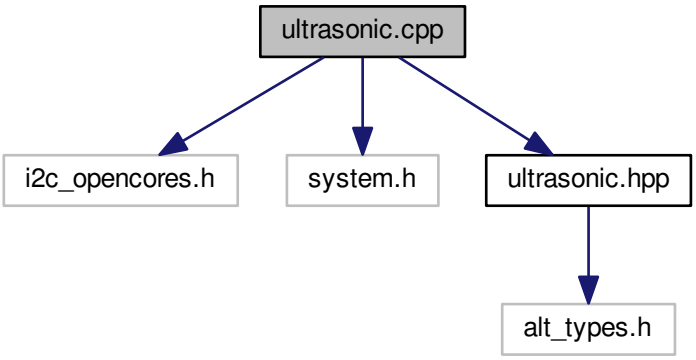
Variables

- Alf_SharedMemoryComm **sharedMem**
- TaskHandle_t **writeTask**

5.42 ultrasonic.cpp File Reference

```
#include "i2c_opencores.h"
#include "system.h"
#include "ultrasonic.hpp"
```

Include dependency graph for ultrasonic.cpp:

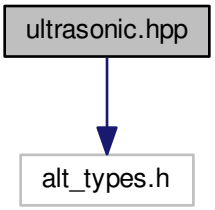


5.43 ultrasonic.hpp File Reference

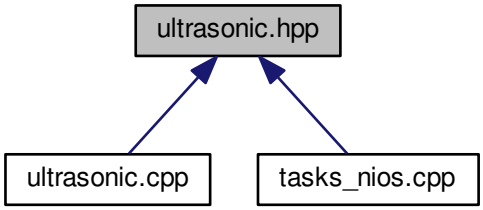
this file contains the definition of all ultrasonic specific modules (currently only UltraSonicDevice)

```
#include "alt_types.h"
```

Include dependency graph for ultrasonic.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class UltraSonicDevice

  represents a ultrasonic hardware device

Enumerations

- enum UltraSonicAddress : alt_u8 {
  DEVICE_00 = 0xE0, **DEVICE_01** = 0xE2, **DEVICE_02** = 0xE4, **DEVICE_03** = 0xE6,
  DEVICE_04 = 0xE8, **DEVICE_05** = 0xEA, **DEVICE_06** = 0xEC, **DEVICE_07** = 0xEE,
  DEVICE_08 = 0xF0, **DEVICE_09** = 0xF2, **DEVICE_10** = 0xF4, **DEVICE_11** = 0xF6,
  DEVICE_12 = 0xF8, **DEVICE_13** = 0xFA, **DEVICE_14** = 0xFC, **DEVICE_15** = 0xFE }

  defines all possible IIC addresses for the SRF08 ultra sonic range finder
- enum UltraSonicRegistersWrite : alt_u8 { **COMMAND** = 0x00, **MAX_GAIN** = 0x01, **RANGE** = 0x02 }

  defines all possible write registers
- enum UltraSonicRegisterRead : alt_u8 {
  SW_REVISION = 0x00, **LIGHT_SENSOR** = 0x01, **ECHO_0x01** = 0x02, **ECHO_0x02** = 0x04,
  ECHO_0x03 = 0x06, **ECHO_0x04** = 0x08, **ECHO_0x05** = 0x0A, **ECHO_0x06** = 0x0C,
  ECHO_0x07 = 0x0E, **ECHO_0x08** = 0x10, **ECHO_0x09** = 0x12, **ECHO_0x0A** = 0x14,
  ECHO_0x0B = 0x16, **ECHO_0x0C** = 0x18, **ECHO_0x0D** = 0x1A, **ECHO_0x0E** = 0x1C,
  ECHO_0x0F = 0x1E, **ECHO_0x10** = 0x20, **ECHO_0x11** = 0x22 }

defines all possible read registers

- enum UltraSonicCommands : alt_u8 {
  START_MEAS_INCHES = 0x50, **START_MEAS_CM** = 0x51, **START_MEAS_TIME_MICROSEC** = 0x52, **START_MEAS_INCHES_ANN** = 0x53,
  START_MEAS_CM_ANN = 0x54, **START_MEAS_TIME_MICROSEC_ANN** = 0x55, **CHANGE_ADDRES↩ S_COMMAND_1** = 0xA0, **CHANGE_ADDRESS_COMMAND_2** = 0xAA,
  CHANGE_ADDRESS_COMMAND_3 = 0xA5 }

  defines all possible commands for the ultrasonic sensor

5.43.1 Detailed Description

this file contains the definition of all ultrasonic specific modules (currently only UltraSonicDevice)

5.44 using_shared_memory_example.cpp File Reference

Functions

- void **this_is_my_interruptroutine** (void ∗mess)
- int main ()

  the main process of this application this does

Variables

- Alf_SharedMemoryComm communication

  This is a example how to use the Alf_SharedMemoryComm in a proper way. This example is not compileable!

5.44.1 Function Documentation

5.44.1.1 int main ( )

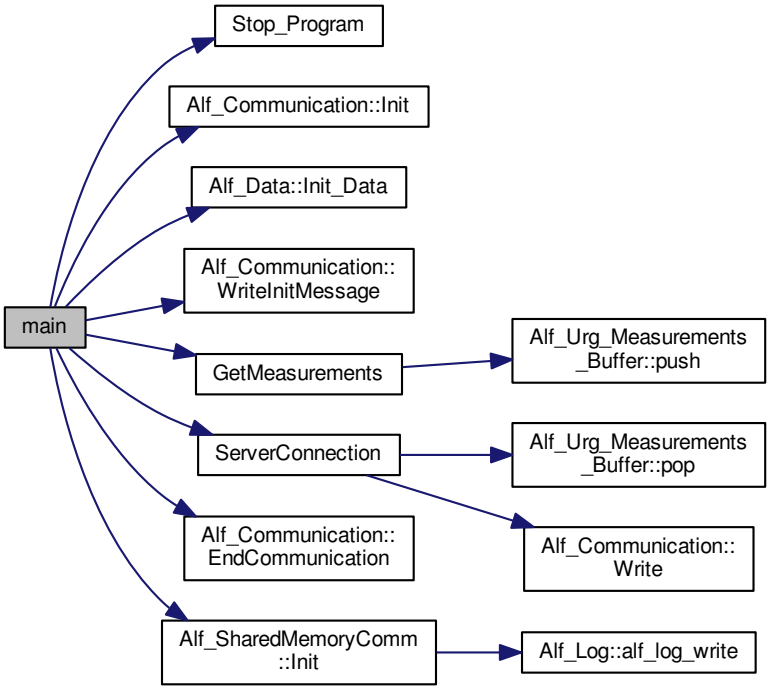
the main process of this application this does

The Main function.

- initializing the urg_sensor
- initializing the server connection
- starting the two threads
- ending the application in a clean way (after CTRL+C)

Definition at line 23 of file using_shared_memory_example.cpp.

Here is the call graph for this function:



5.44.2 Variable Documentation

5.44.2.1 Alf_SharedMemoryComm communication

This is a example how to use the Alf_SharedMemoryComm in a proper way. This example is not compileable!

: florian : 2017-03-12T12:31:18+01:00 modified by: florian modified time: 2017-03-12T12:31:18+01:00

Definition at line 13 of file using_shared_memory_example.cpp.

Chapter 6

Example Documentation

6.1 using_shared_memory_example.cpp

The routine which should be called within the interrupt routine for receiving messages. It saves the pointer and command register from the read mailbox and saves that information. Later in a programm context, you can read a object from the shared memory with #Read This is an example how to proper use the class and in special this function!