A Distributed, Time-Triggered Architecture For an Attitude Control System

Jan Bredereke, Sercan Catalkaya, Eike Diekmann, Henrik Gießel, Arthur Guz, Daniel Kunde, Tim Niebuhr, Kai Nortmann, Hans-Martin Pfennig, Marvin Pöperny, André Sarich, Olga Tschernobai, Hermann Wafo, René Wanzow, and Christian Zöller

City University of Applied Sciences Bremen
Flughafenallee 10, D-28199 Bremen

`http://homepages.hs-bremen.de/~jbredereke`

Rico Thiele

Airbus Defence & Space
Airbus-Allee 1, D-28199 Bremen

Mar. 2018

The following pages were generated from the project wiki hosted by Airbus.

CFGv2.1 - A Distributed, Time-Triggered Architecture For an Attitude Control System

Rico Thiele

Creation Date:	29.09.2017 09:39
Modification Date:	11.12.2017 15:56
Version:	21

Table of Contents

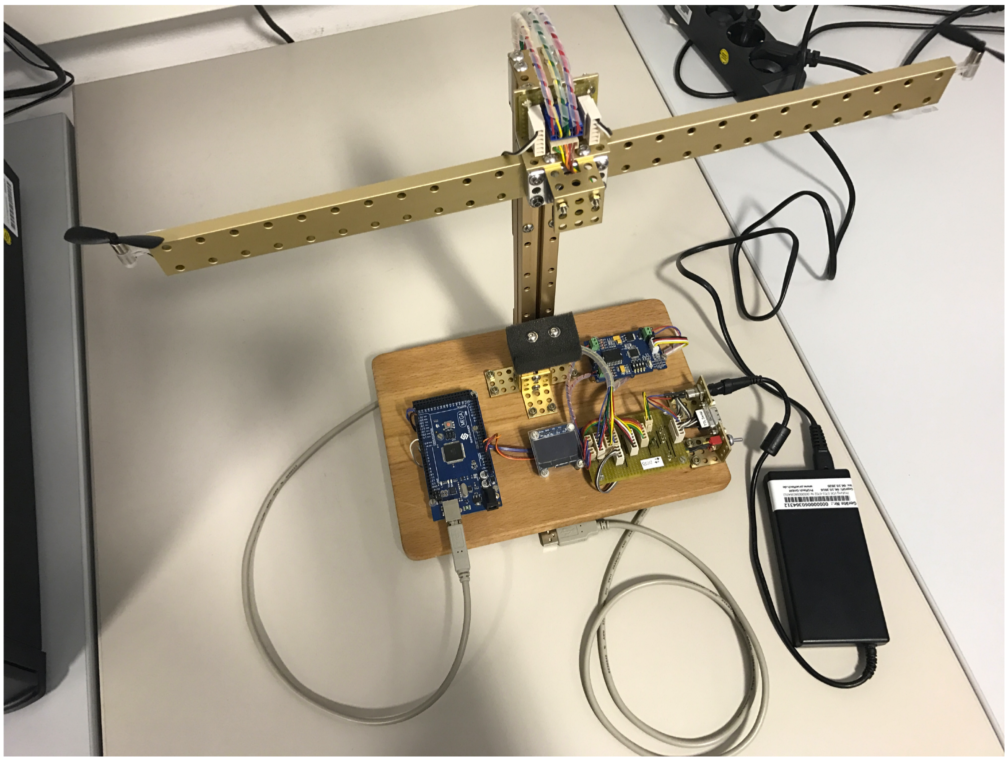
1 Introduction

Prof. Dr. Jan Bredereke, City University of Applied Sciences Bremen

This project demonstrates how to implement a time-triggered architecture for a real-time and distributed embedded system. The application is a simple attitude control system. The project was carried out as part of the course "Embedded Systems" at the City University of Applied Sciences Bremen, during the winter term 2017/18. It was supervised by Prof. Dr. Jan Bredereke. The chapters after the introduction were written by the students named there. The project was conducted in cooperation with Airbus Bremen.

1.1 The Attitude Control System Demonstrator

The demonstrator consists of an arm with a single degree of freedom, see figure. The arm is kept in a defined attitude by two propellers. The control algorithm for this is executed on a STM32 microcontroller running the FreeRTOS operating system. The actuators and the sensor are controlled by an Arduino-compatible microcontroller (Sunfounder Mega). Both microcontrollers communicate over an RS-232 serial connection using a time-triggered communication protocol.



1.2 The Time-Triggered Architecture

1.2.1 Time-Triggered Processing for Hard Real-Time Systems

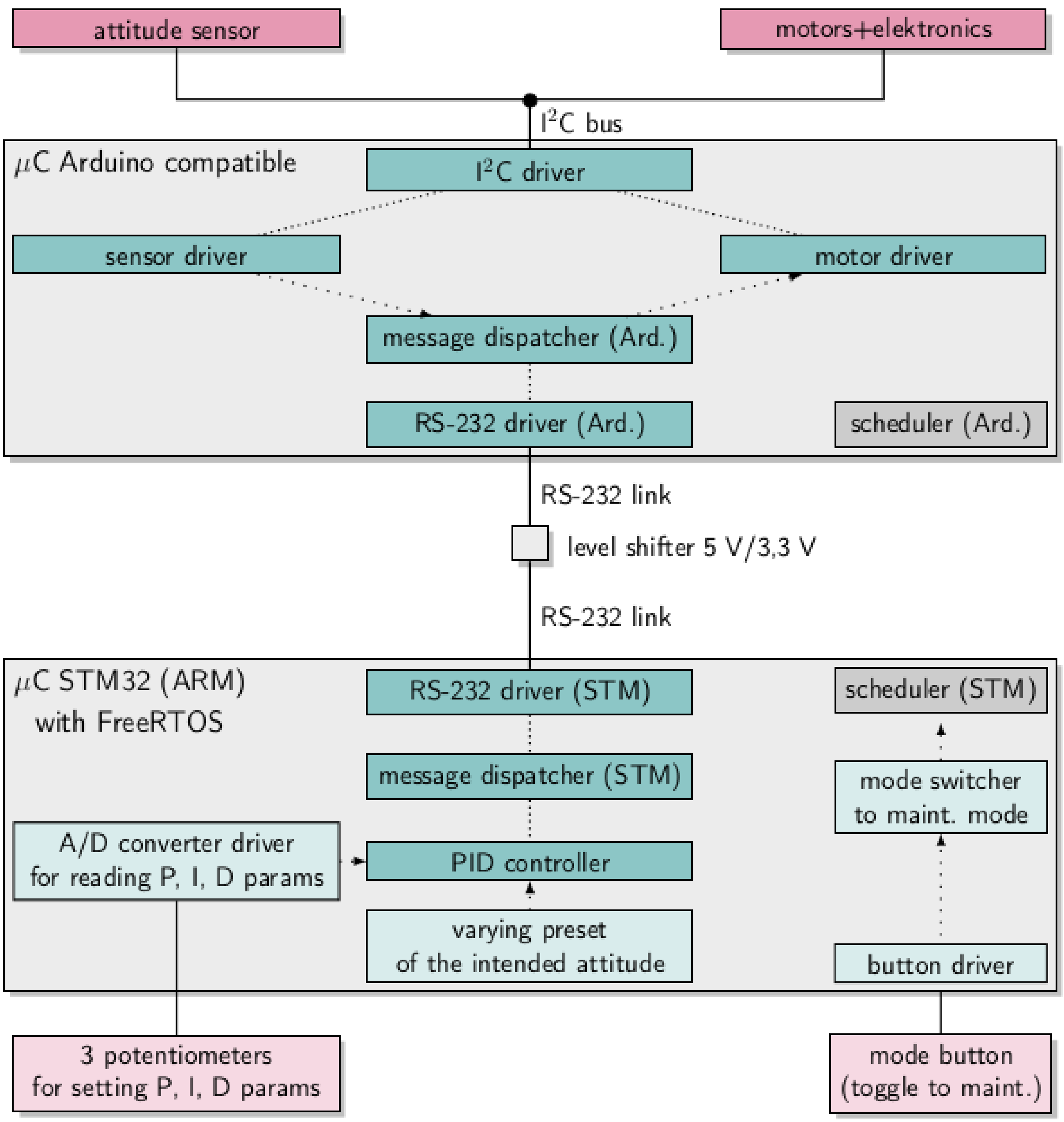
Using a time-triggered processing scheme helps to meet hard real-time constraints. Every task has a fixed time slot in the schedule, with a fixed period and a fixed length. This allows to prove that a system constructed in this way will meet its timing requirements under all circumstances. Such a proof can be done by checking that the worst-case execution time of every task in isolation does not exceed the length of its alotted time slot. Such a system does not use any interrupt mechanism. Each task runs to completion. The scheduling therefore is cooperative. If the worst-case execution times of all tasks have been proven to not exceed their deadlines, such a design is highly reliable.

An alternative scheme would be event-triggered processing. Such a scheme uses interrupts and priorities for tasks. At any time, a higher-priority task may interrupt the execution of a lower-priority task. This allows for a quick response to an urgent issue. Also, the average response time of tasks in such a scheme often is considerably shorter than when using a time-triggered scheme. However, an interrupting task may be interrupted itself, and interrupts may be postponed for a long time or even lost due to other high-priority processing. Therefore, usually it is practically impossible to provide a proof that a specific task will meet a specific deadline in the worst case. Consequently, an event-triggered processing scheme usually is not suitable for a hard real-time system. Hard real-time system here means that guarantees for its reliability must be provided.

1.2.2 The Time-Triggered Architecture for the Attitude Control System Demonstrator

We use a distributed time-triggered architecture for our attitude control system demonstrator. There are two processing nodes, and they synchronize und communicate using an RS-232 serial interface connection. The STM32 microcontroller is the master. It generates the time ticks which determine the schedule of all tasks on all nodes. The Arduino-compatible microcontroller is a slave and follows the time ticks. The communication schedule is defined by these regular time ticks, too. We defined a simple time-triggered communication protocol for this. The details follow in the sections below.

The following figure shows the system structure. Light blue and light red boxes denote optional components, to be realized only if time permits. One optional component is a varying preset of the intended attitude. It is intended to make the behaviour of the controlled arm look more interesting. Furthermore, optionally the parameters of the PID controller shall be controllable by three potentiometers. This can help to find suitable values for these parameters much faster than by recompiling the software for every iteration. In order to be able to find out the current values of these parameters, optionally there is also a button that toggles the STM32 node into a maintenance mode. In the maintenance mode, the PID controller does not run. Instead, the parameter values are sent by a maintenance program to a PC on a second RS-232 link. (This mode and the second link are not shown in the figure.)



1.3 Original Project Description (in German)

projektbeschreibung-2017-09-29.pdf

2 FreeRTOS

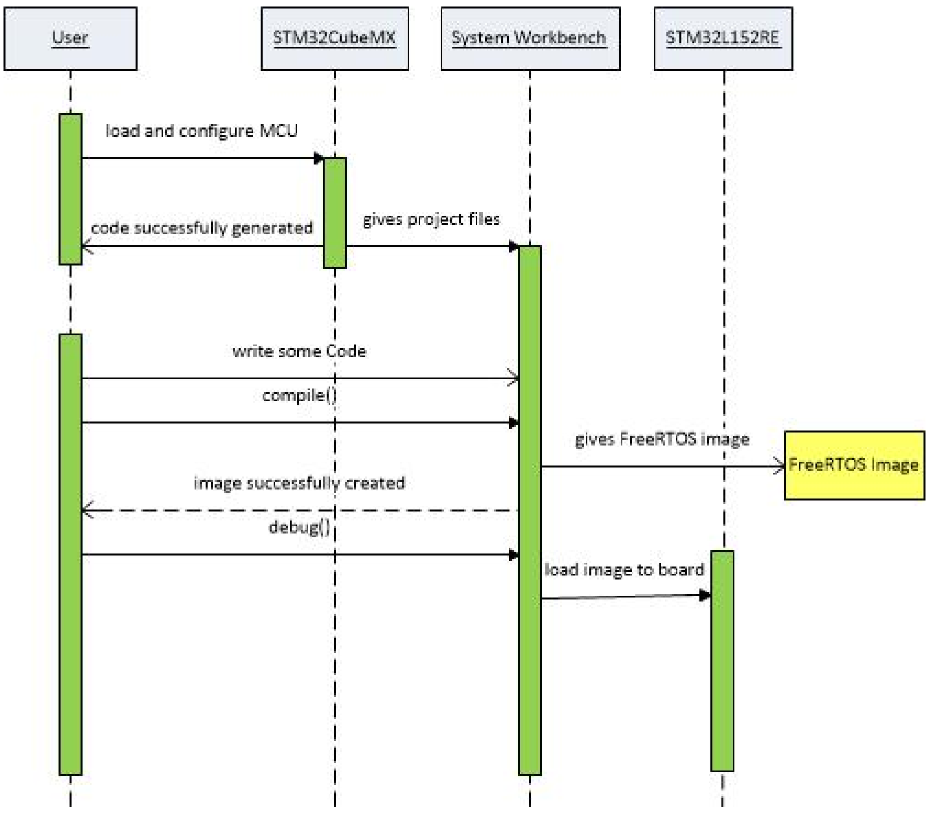
Authors: Kai Nortmann, Henrik Gießel, Arthur Guz, René Wanzow

This page describes FreeRTOS which is the real-time operating system we use in our project on the STM32L1 board.

2.1 Contents

2.2 Toolchain

Authors: Henrik Gießel



The sequencediagram shows the workflow of the creation of a FreeRTOS image. In the first step the user has to create project files for the needed microctrontroller. The Framewok STM32CubeMX will create the project files after the user finished the needed configuration.The system workbench can import this project files and the user can start writing his code. Everytime the user successfull compile the project a FreeRTOS ".bin"-image will be created in the workspace. The image can be loaded onto the microcontroller when he click "debug" in the system workbench IDE.

2.3 Getting Started with System Workbench for STM32

Authors: Kai Nortmann, Henrik Gießel

This sections descripes how to get start with the System Workbench for STM32.

The System Workbench toolchain,called SW4STM32, is a free, multi-OS software develepemont enviroment based on Eclipse, which supports the full range of STM32 microcontrollers and associated boards.It has been built by AC6, a service company providing training and consultancy on embedded systems. For the latest information on the specification, refer to the third party’s website: www.ac6.fr.

2.3.1 Getting the Software

Authors: Kai Nortmann, Henrik Gießel

To get the software you need to register first at: http://www.openstm32.org/tiki-register.php.Fillout this exhausting list of personal information and confirm to get a verification e-mail to finish the registration.
Without this step you wont be able to get software from the Openstm32 community and to do further steps.

After the registration go following ressoucre on http://www.openstm32.org to get the needed installer: Documentation » System Workbench for STM32 » Installing System Workbench for STM32 » Installing System Workbench for STM32 with installer » Downloading the System Workbench for STM32 installer.

For Windows7 and newer (64 Bit) download: "install_sw4stm32_win_64bits-v2.3.exe" for the 32 Bit edition the install_sw4stm32_win_32bits-v2.3.exe

For Linux based systems (64 Bit) download: "install_sw4stm32_linux_64bits-latest.run". The 32 Bit Version will be deptricated in the future and will not be documented in this dokumentation.
Download possible with:

```
wget http://www.ac6-tools.com/downloads/SW4STM32/install_sw4stm32_linux_64bits-v2.3.run
```

2.3.2 Installation

Windows Installation Setup

Authors: Kai Nortmann, Henrik Gießel

Start the installation process by running the install_sw4stm32_win_64bits-v2.3.exe. Specify the installation location and accept the licence agreements. Other setting can be left default.
In the Instalaltion-process following ressources will be installed:

1. System Workbench files
2. Needed drivers (STMicroelectronics (WinUSB) STLinkWinUSB, (STMicroelectronics (usbser))
3. Uninstaller in the installation location

In the finish dialog of the setup an installation-script can automaticaly be generated to repeat the installation on other computer. If additional installations are necessary this point
should be considered.

Linux Installation Setup

Authors: Kai Nortmann, Henrik Gießel

```
# Download Dependecies for OpenSTM32
sudo apt-get install libc6:i386
sudo apt-get install lib32ncurses5

# Download dependecies for the GUI Installer of OpenSTM32
sudo apt-get install gksu

# Change to download directory
cd <$DOWNLOAD_DIRECTORY>

# Granting rights to run installer
sudo chmod a+x install_sw4stm32_linux_64bits-latest.run

# Run GUI Installer in interactive mode on the current user
bash install_sw4stm32_linux_64bits-latest.run

# Run GUI Installer in interactive mode as root for mutli-user systems
sudo bash install_sw4stm32_linux_64bits-latest.run -f

# Run Installer with answer-file
bash install_sw4stm32_linux_64bits-latest.run <$PATH_TO_ANSWER_FILE.XML>
```

The GUI-Installer will run in following steps:

1. The first page describes the product features. Click on the “Next” button.
2. Please read and accept the license agreement to continue the installation. Click on the “Next” button.
3. Choose the installation path (default: /home/user/Ac6/SystemWorkbench). Click on the “Next” button.
4. A warning message is displayed :
    a. If the directory does not exist, it proposes to create the installation directory.
    b. If the directory exists, the installer will suppress the installation folder.
5. The next page shows the list of packs that will be installed.
6. The following page shows on the installation settings. Click on the “Next” button to proceed installation then wait until the install process is done.
7. The next page will automatically unpack the tool and configure the tool.
8. When the unpack is finished, click on the “Next” button to display to Finish page.

2.3.3 Configuration of the System Workbench

Authors: Kai Nortmann, Henrik Gießel

There is no configuration needed in the default settings. Just open the project, generated by STM32CubeMX (described in the section below)

2.4 Installation of STM32CubeMX

2.4.1 Getting the Installation Files

Authors: Kai Nortmann, Henrik Gießel

STM32CubeMX is freeware package for Windows, Mac OS X and Linux that is a graphical software configuration tool that allows generating C initialization code using graphical wizards. The installation files can be found on http://www.st.com/en/development-tools/stm32cubemx.html in the "Get Software" section. Read and Accept licence agreement and download the zip-archive which contains all installation programms for WIndows, Mac OS X and Linux.

2.4.2 Installation on Windows

Authors: Kai Nortmann, Henrik Gießel

General Steps:

1. Switch to download directory and unzip file to desired directory
2. Run "SetupSTM32CubeMX-4.23.0.exe"
3. Accept licence agreement
4. Choose installation location
5. Choose of you want shortcuts on desktop and startmenu and finish installation
6. Run "<$INSTALLDIR>\STMicroelectronics\STM32Cube\STM32CubeMX\STM32CubeMX.exe" to start STM32CubeMX

No further instructions are needed.To uninstall STM32CubeMX run "<$INSTALLDIR>\STMicroelectronics\STM32Cube\STM32CubeMX\Uninstaller\startuninstall.exe"

2.4.3 Installation on Linux

Author: Henrik Gießel

Refer to Section "Getting the Software" to get the needed installation files. You should habe an ZIP-Archive like "en.stm32cubemx.zip". Following script can be used to install STM32CubeMX.

```
# Extract Dependecies
sudo apt-get install gcj-jre
sudo apt-get install default-jre

# Extract ZIP Archive for a single user deployment
unzip en.stm32cubemx.zip -d /home/user/bin/stm32cubemx

# Extract ZIP Archive for a multi user deployment
sudo unzip en.stm32cubemx.zip -d /usr/local/sbin/stm32cubemx
sudo chown -R root:root /usr/local/sbin/stm32cubemx
sudo chmod -R 755 /usr/local/sbin/stm32cubemx

# Install STM32CubeMX
./SetupSTM32CubeMX-4.23.0.linux (graphical-Installer)
./SetupSTM32CubeMX-4.23.0.linux -console (CLI-Installer)
```

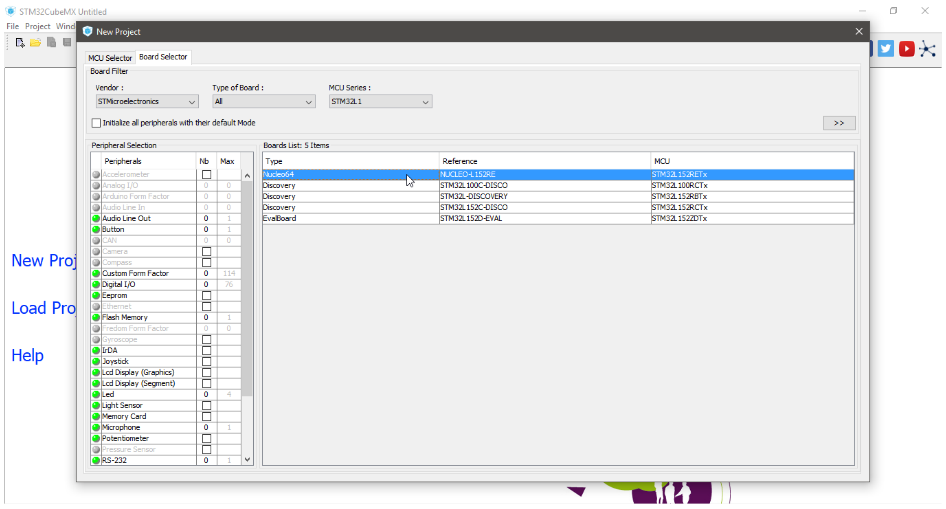
1. Accept licence agreement
2. Choose installation location
3. Choose of you want shortcuts on desktop and startmenu and finish installation
4. Run "<$INSTALLDIR>\STM32CubeMX" to start STM32CubeMX in Linux

2.4.4 Create/Configurate a Project for Board STM32L152RE with STM32CubeMX

author: René Wanzow

General steps:

1. Start STM32CubeMX with "<$INSTALLDIR>\STMicroelectronics\STM32Cube\STM32CubeMX\STM32CubeMX.exe"
2. Click on "File" "New Project"
3. Switch to "Board selector"-Tab in the "New Project"-Window
4. Double-click on "NUCLEO-L152RE" (see left screenshot below)



Board Filter

Vendor : STMicroelectronics
Type of Board : All
MCU Series : STM32L1

☐ Initialize all peripherals with their default Mode

Peripheral Selection

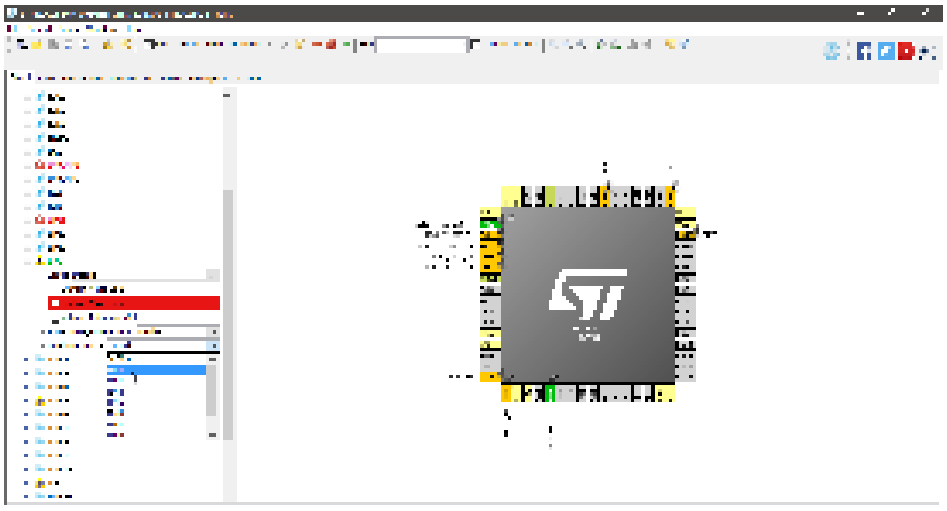
Peripherals	Nb	Max
Accelerometer	☐	
Analog I/O	0	0
Arduino Form Factor	0	0
Audio Line In	0	0
Audio Line Out	0	1
Button	0	1
CAN	0	0
Camera	☐	
Compass	☐	
Custom Form Factor	0	114
Digital I/O	0	76
Eeprom	☐	
Ethernet	☐	
Flash Memory	0	1
Fredom Form Factor	0	0
Gyroscope	☐	
IrDA	☐	
Joystick	☐	
Lcd Display (Graphics)	☐	
Lcd Display (Segment)	☐	
Led	0	4
Light Sensor	☐	
Memory Card	☐	
Microphone	0	1
Potentiometer	☐	
Pressure Sensor	☐	
RS-232	0	1

Boards List: 5 Items

Type	Reference	MCU
Nucleo64	NUCLEO-L152RE	STM32L152RETx
Discovery	STM32L100C-DISCO	STM32L100RCTx
Discovery	STM32L-DISCOVERY	STM32L152RBTx
Discovery	STM32L152C-DISCO	STM32L152RCTx
EvalBoard	STM32L152D-EVAL	STM32L152ZDTx

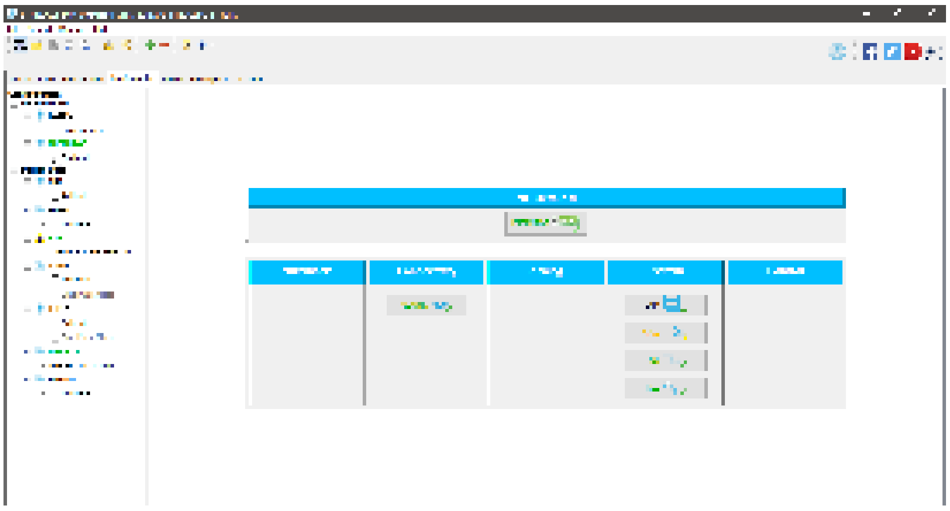
Board selection

5. After few seconds the "pinout"-window will appear
6. Expand the "SYS"-node and change the "timebase source" to "TIM2" (see right screenshot below)



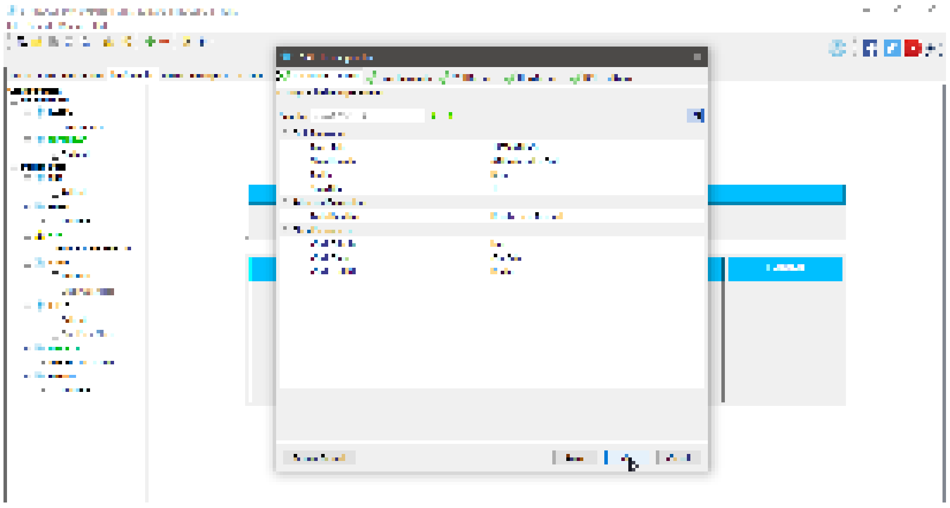
Board configuration

7. Expand the "USART1"-node and set the mode to "synchronous"
8. Switch to the "configuration"-tab and enable "FREERTOS" und the "middleware"-node (see screenshot below)



Main configuration

9. If needed the configuration of the USART-interface could be changed (baudrate, word length, parity, etc.; see screenshot below)



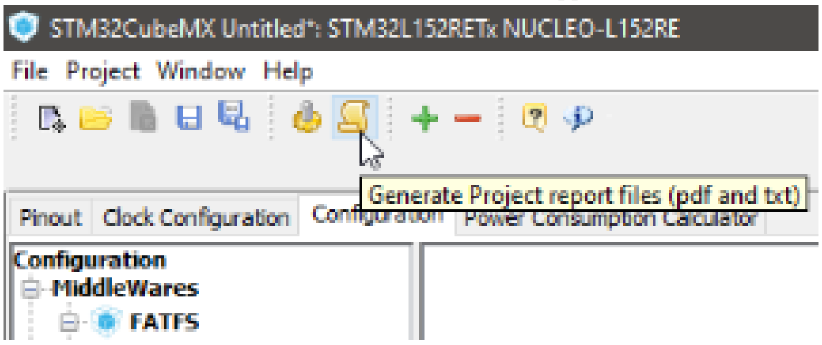
USART configuration

10. The basic configuration is now complete

2.5 Generate FreeRTOS Project with STM32CubeMX

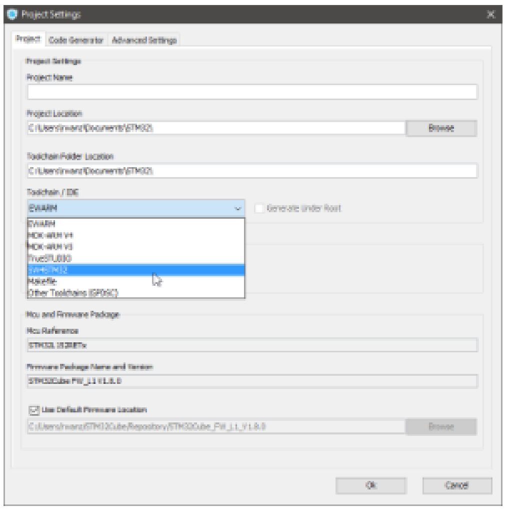
author: René Wanzow

1. After the creation and configuration of a project with STM32CubeMX, the project can be generated for the IDE
2. Generate the project for "System Workbench for STM32" by clicking on the srcoll-symbol on the button-toolbar of the main-window in "STM32CubeMX" application (see screenshot below)


STM32CubeMX Untitled*: STM32L152RETx NUCLEO-L152RE
File Project Window Help
Generate Project report files (pdf and txt)
Pinout
Clock Configuration
Configuration
Power Consumption Calculator
Configuration
MiddleWares
FATFS

3. Confirm the next window with "yes"
4. In the next window you can configurate the genrate-process
5. Set a destination folder for the generated project

6. Also select the toolchain/IDE to "SW4STM32" to be able to open it later with the workbench



Project Settings

Project Name

Project Location: C:\Users\rivanz\Documents\STM32\

Toolchain Folder Location: C:\Users\rivanz\Documents\STM32\

Toolchain / IDE: EWARM

EWARM
MDK-ARM V4
MDK-ARM V5
TrueSTUDIO
SW4STM32
Makefile
Other Toolchains (GPDSC)

Mcu and Firmware Package

Mcu Reference: STM32L152RETx

Firmware Package Name and Version: STM32Cube FW_L1 V1.8.0

Use Default Firmware Location: C:/Users/rivanz/STM32Cube/Repository/STM32Cube_FW_L1_V1.8.0

7. Click on "Ok" and the project should be generated. After the generate process is complete, the project can be found in the previous selected folder.
8. Finish, the project is ready to open with the "Workbench for STM32"

Several example projects can be found in the Git-Repository under "acd\ST Nucleo-L152RE-demos\"-folder.

2.5.1 Build project for other IDE / Toolchain

For building for other tools, just repeat the instruction above. At step 6 just choose another Toolchain / IDE to built your project for the given tools.

2.6 Difference between HAL and LL

2.6.1 HAL drivers

Authors: Kai Nortmann, Arthur Guz

"The HAL drivers were designed to offer a rich set of APIs and to interact easily with the application upper layers. Each driver consists of a set of functions covering the most common peripheral features. The development of each driver is driven by a common API which standardizes the driver structure, the functions and the parameter names. The HAL drivers include a set of driver modules, each module being linked to a standalone peripheral. However, in some cases, the module is linked to a peripheral functional mode. As an example, several modules exist for the USART peripheral: UART driver module, USART driver module, SMARTCARD driver module and IRDA driver module. The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
  - Fully reentrant APIs
  - Systematic usage of timeouts in polling mode.

- Support of peripheral multi-instance allowing concurrent API calls for multiple instances of a given peripheral (USART1, USART2...)
- All HAL APIs implement user-callback functions mechanism:
    - Peripheral Init/DeInit HAL APIs can call user-callback functions to perform peripheral system level Initialization/De-Initialization (clock, GPIOs, interrupt,DMA)
    - Peripherals interrupt events
    - Error events.
- Object locking mechanism: safe hardware access to prevent multiple spurious accesses to shared resources.
- Timeout used for all blocking processes: the timeout can be a simple counter or a timebase."(UM1816 User manual, section 2)

2.6.2 LL drivers

Authors: Kai Nortmann, Arthur Guz

"The Low Layer (LL) drivers are designed to offer a fast lightweight expert oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, LL APIs are not provided for peripherals where optimized access is not a key feature, or those requiring heavy software configuration and/or complex upper-level stack (such as FSMC, USB or
SDMMC).
The LL drivers feature:

- A set of functions to initialize peripheral main features according to the parameters specified in data structures
- A set of functions used to fill initialization data structures with the reset values of each field
- Functions to perform peripheral de-initialization (peripheral registers restored to their default values)
- A set of inline functions for direct and atomic register access
- Full independence from HAL since LL drivers can be used either in standalone mode (without HAL drivers) or in mixed mode (with HAL drivers)
- Full coverage of the supported peripheral features.

The Low Layer drivers provide hardware services based on the available features of the STM32 peripherals. These services reflect exactly the hardware capabilities and provide one-shot operations that must be called following the programming model described in the microcontroller line reference manual. As a result, the LL services do not implement any processing and do not require any additional memory resources to save their states, counter or data pointers: all the operations are performed by changing the associated peripheral registers content."(UM1816 User manual, section 3)

2.6.3 Tablewise comparison of HAL and LL drivers

author: René Wanzow

Refering to the quotations above, this table compares the HAL and LL drivers in a single, compact table.

	HAL-driver	LL-driver
Level	Uses APIs as interfaces for the hardware, easy access to upper application layer, driver module being linked to a standalone peripheral	Close to hardware (but uses APIs too), expert-oriented, direct and register level based operations, LL APIs are not provided for peripherals, provide hardware services: these services reflect exactly the hardware capabilities
Memory	Uses more memory in comparison to LL-driver	fast light-weight, better optimization
Configuration	Initialization functions in API, offers Initialization structures	Initialization functions in API, The LL drivers offer three sets of initialization functions
Functionality coverage	common peripheral features (extendable through extended APIs)	full coverage
Special functionality	Three API programming models: polling, interrupt and DMA	LL services do not implement any processing and …
	Support of peripheral multi-instance	No additional memory is required to save the states, counter or data pointers (operations performed by changing the associate register)
	All HAL APIs implement user-callback functions mechanism	The Low Layer is designed to be used in standalone mode or combined with the HAL
	Object locking mechanism	
	Timeout used for all blocking processes	
	Both drivers can be used in mixed mode	

2.7 Serial communication

source: http://www.st.com/content/ccc/resource/technical/document/reference_manual/cc/f9/93/b2/f0/82/42/57/CD00240193.pdf/files/CD00240193.pdf/jcr:content/translations/en.CD00240193.pdf

2.7.1 UART

An universal asynchronous receiver-transmitter (UART) is a computer hardware device for asynchronous serial communication. It takes bytes of data and transmitts the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes.

Asynchronous means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.

The two UARTS of the STM32L152RE-MCU provide the following operation modes:

- asynchronous mode
- multibuffer communication (DMA)
- Multiprocessor communication
- Half-duplex(single-wire mode)
- IrDA
- LIN

2.7.2 USART

An universal synchronous and asynchronous receiver-transmitter (USART) is a type of serial interface device for synchronous or asynchronous serial communication. For the asynchronous capabillities of an USART reffer to the UART article.

Synchronous serial transmission requires that the sender and receiver share a clock with one another, or in other words, the sender provide a clock or other timing signal so that the receiver knows when to “read” the next bit of the data.

The three USARTs of the STM32L152RE-MCU provides the following operation modes:

- Asynchronous mode
- Hardware flow control
- Multibuffer communication (DMA)
- Synchronous
- Smartcard
- Multiprocessor communication
- Half-duplex(single-wire mode)
- IrDA
- LIN

2.8 Understanding Polling-, Interrupt- and DMA-Mode of HAL-driver

Aurthor: Arthur Guz

Source: “Description of STM32L1 HAL and low-layer drivers”, Chapter 2.12.3

"The HAL functions with internal data processing like transmit, receive, write and read are generally provided with three data processing modes as follows:"

- Polling mode
- Interrupt mode
- DMA mode

2.8.1 Polling mode

"In Polling mode, the HAL functions return the process status when the data processing in blocking mode is complete. The operation is considered complete when the function returns the HAL_OK status, otherwise an error status is returned. The user can get more information through the HAL_PPP_GetState() function. The data processing is handled internally in a loop. A timeout (expressed in ms) is used to prevent process hanging. "

2.8.2 Interrupt mode

"In Interrupt mode, the HAL function returns the process status after starting the data processing and enabling the appropriate interruption. The end of the operation is indicated by a callback declared as a weak function. It can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the HAL_PPP_GetState() function. "

2.8.3 DMA mode

"In DMA mode, the HAL function returns the process status after starting the data processing through the DMA and after enabling the appropriate DMA interruption. The end of the operation is indicated by a callback declared as a weak function and can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the HAL_PPP_GetState() function. "

2.9 Timing

Author: Kai Nortmann

2.9.1 The SysTick-Timer (HAL)

- "Systick-timer is used by default as source of time base, but user can eventually implement his proper time base source (a general purpose timer for example orother time source), keeping in mind that Time base duration should be kept 1ms since PPP_TIMEOUT_VALUEs are defined and handled in milliseconds basis.
- Time base configuration function (HAL_InitTick ()) is called automatically at the beginning of the program after reset by HAL_Init() or at any time when clock is configured, by HAL_RCC_ClockConfig().
- Source of time base is configured to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, the Tick interrupt line must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked.
- functions affecting time base configurations are declared as __Weak to make override possible in case of other implementations in user file." (UM1816 User manual, section 5.1.2)

Since the time base duration is also used to handle the timeouts of certain tasks (for example sending bytes on the UART bus), the value should not be changed. When changin the value, it should be kept in mind, that all timeout values are affected by that change. If a certain task is given a timeout value of 1000, it would be one second if the time base duration is set to 1 ms. Changing the time base duration to 10ms, all timeout values are the ten times bigger than before!

2.9.2 The RTOS Tick

"When sleeping, an RTOS task will specify a time after which it requires 'waking'. When blocking, an RTOS task can specify a maximum time it wishes to wait.

The FreeRTOS real time kernel measures time using a **tick** count variable. A timer interrupt (the RTOS **tick interrupt**) increments the tick count with strict temporal accuracy - allowing the real time kernel to measure time to a resolution of the chosen timer interrupt frequency.

Each time the tick count is incremented the real time kernel must check to see if it is now time to unblock or wake a task. It is possible that a task woken or unblocked during the tick ISR will have a priority higher than that of the interrupted task. If this is the case the tick ISR should return to the newly woken/unblocked task - effectively interrupting one task but returning to another." (RTOS Homepage - "How FreeRTOS works")

The standard value of each **tick** is 1 ms. The tick value is directly coupled to the "SysTick-Timer" of the STM32 HAL driver (so changing the tick value will set the time base duration of the STM32 to that exact value). The tick value can be used in this project to determine the duration of each timeframe. It should be kept in mind that all timeout values are affected by that change! (see above).

2.9.3 Built-in Scheduler in freeRTOS

Author: René Wanzow

freeRTOS comes with a built-in scheduler which comes with basic features for scheduling. It is possible to configure the scheduler to operate preemptive or cooperative.

In the "FreeRTOSConfig.h" iit is possible to configure that mode of operation:

```
FreeRTOSCOnfig.h

[...]


#define configUSE_PREEMPTION                     0


[...]
```

If this define is set to "0", the scheduler operates in cooperative mode. If that is the case you need to insert a "osThreadYield();" at the end of each task to signal the scheduler, that there is not more caöculation time needed and other tasks in the queue can be called (see the code block below).

```
main.c

void startInitTask(void const * argument)
{
    struct Data data;
    for(;;)
    {
        transmitDebug("init task running \r\n", 10);

        data = GetEmptyData();
        osThreadSetPriority(initTaskHandle, osPriorityLow);
        osThreadYield();
    }
}
```

To determine for the scheduler which task should be called next, there is a queue of tasks with a priority (see in code block upwards the line with "osThreadSetPriority"). If you want to transfer data to other tasks you can do that by the method "osMessagePut" (see code block below).

```
main.c

//put received data onto Rx-buffer
osMessagePut(serialBufRxHandle,(uint32_t) &tmp, 10);
osThreadYield();
```

2.9.4 Example Projects for the STM32

Author: René Wanzow

LED-Blink via button

Example project where the LED (LD2) toggles if the blue button (B1) was pressed. There are two varaiants in the Git-Repository, one with interrupt support and one with polling on the pin of the button instead of interrupt.

```
Project with interrupt

 git@acd\ST Nucleo-L152RE-demos\"LED_BLINK_BUTTON_EXAMPLE_withInterrupt.zip"
```

```
Project with polling

git@acd\ST Nucleo-L152RE-demos\"LED_BLINK_BUTTON_EXAMPLE_withoutInterrupt.zip"
```

LED-Blink via Sys-timer

In this example, LD2 toggles on each timer tick (TIM2).

```
Project with systick

git@acd\ST Nucleo-L152RE-demos\"LED_BLINK_SYSTICK_EXAMPLE.zip"
```

2.10 Appendix

2.10.1 Automated Installation File for Systemworkbench

2.10.2 Automated Installation File for STM32CubeMX

2.10.3 STM32CubeMX Configuration for STM32L152RE

DefaultProjectConfig.ioc

3 Angle Controller (PID)

Authors: Olga Tschernobai, Marvin Pöperny, Hans Martin Pfennig

3.1 Content

The following content is about angle control with a PID controller and its basic functionality. For demonstration purposes is the ACD (Angle Control Demonstrator) provided by Airbus used.

3.2 The control loop

Remark

Basically the following paragraph "Structure" is copied from the previous Documentation but modified to be more readable.
Remark just before delivery: Most paragraphs are completely rewritten and expanded with more information.

3.2.1 Control loops in general

Basics

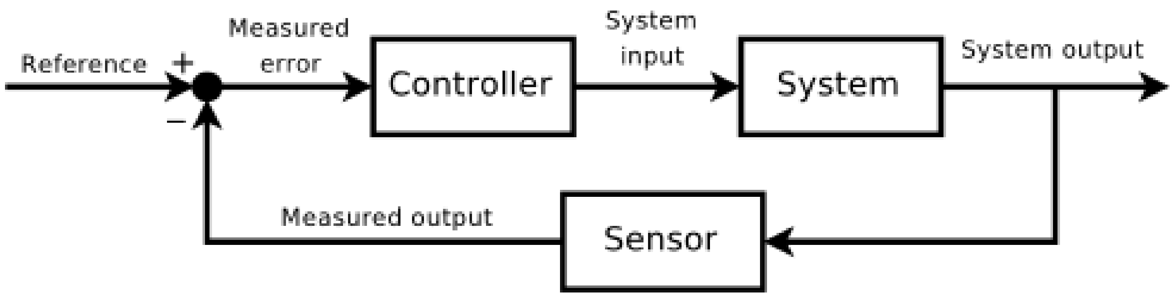
A control loop is a mechanism that allows a system to be controlled dynamically. By means of a target/actual comparison, it makes it possible to adjust the current value of a system and to take errors into account. Picture 1 shows the systematic structure and context of the components which are explained in more detail below.

Controller

This is the part of the control loop that takes corrective action to correct the system deviation, taking into account the dynamic properties of the controlled system.

System

The System is the actual system which has to be controlled.



Picture 1: Standard control loop

Sensor and Measured output / Process variable (PV)

The sensor measures the current value and passes it on in a suitable form (measured output).

Reference / Setpoint (SP)

Is the target value to be reached by the control loop. It is given by an extern source.

Measured error

$$e(t) = SP - PV(t)$$

The measured error is the deviation of the reference value and the measured value. It is the value by which the system still has to be changed at the current point in time to reach the target value.

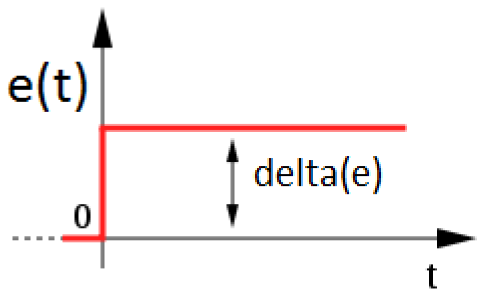
System input / Controller output (CO)

A calculated value by the controller for the system to set the actors correctly.

System output

The real value by the system to control the periphere.

The following four images show jump responses. This means that they only show the behaviour of the controller, but do not affect the system.



Picture 2: set error for step response

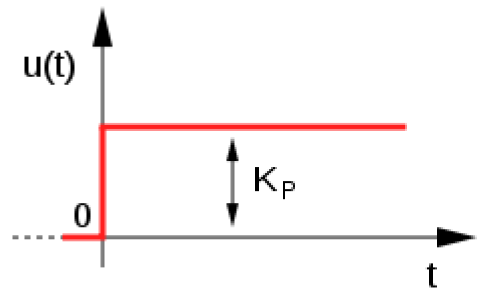
3.2.2 Proportional term (P)

$$P = K_P e(t)$$

The proportional term **P** produces an output that is proportional to the current error value, thus increasing the controllers output with increasing error. It makes up the main part of the control algorithm. The controller multiplies the error e(t) by the proportional gain Kp to get the controller output. The step response of the proportional term is shown in picture 2.

Advantage: The proportional term tries to reach the setpoint as fast as possible.

Disadvantage: If the proportional gain is too large, the process variable will begin to oscillate. If K_c is increased further, the oscillations will become larger and the system will become unstable and may even oscillate out of control.



Source: https://de.wikipedia.org/wiki/Regler
Picture 3: proportional term, step response graph

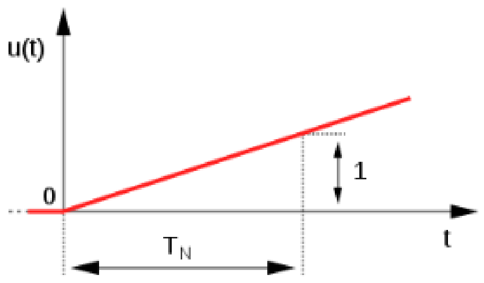
3.2.3 Integral term (I)

$$I = K_I \int_0^t e(x)\,\mathrm{d}x$$

The integral term **I** grows with both the magnitude of the error and its duration, as shown in picture 3.

Advantage: This term can accelerate the movement of the process towards the setpoint if an error is not corrected over longer time periods.

Disadvantage: Because it accumulates the errors of the past it can result in overshooting the setpoint value. To mitigate this effect for certain circumstances windup protection can be used.



Source: https://de.wikipedia.org/wiki/Regler
Picture 4: integral term, step response graph

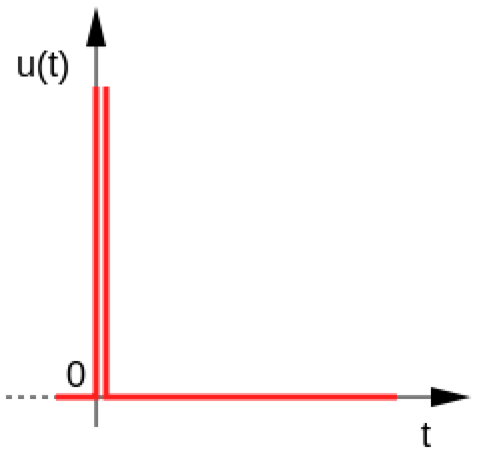
3.2.4 Derivative term (D)

$$D = K_D \frac{\mathrm{d}e(t)}{\mathrm{d}t}$$

The derivative of the error is used to determine the slope of the error over time. This can be used to predict system behavior and improves settling time as well as stability of the system.

Advantage: Increasing the *derivative time (Kd)* parameter will cause the control system to react more strongly to changes in the error term and will increase the speed of the overall control system response.

Disadvantage: The derivative doesn't consider if e(t) is postive, negative or how much time has passed, just how fast e (t) is changing.



Source: https://de.wikipedia.org/wiki/Regler
Picture 5: derivative term, step response graph

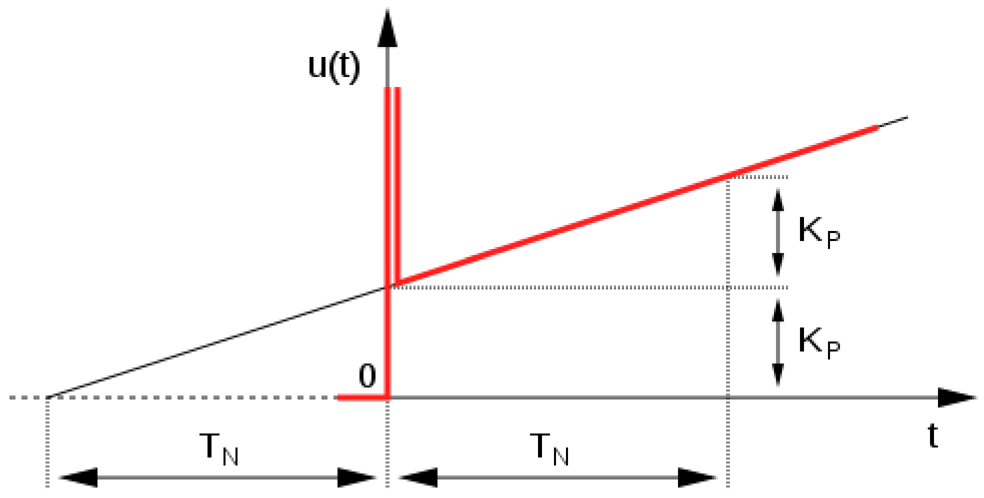
3.2.5 Combination of all terms: PID controller

If all three previously described terms are combined a PID controller is build.

Demo: http://www.rentanadviser.com/en/pid-fuzzy-logic/pid-fuzzy-logic.aspx

The proportional term considers how far the current process variable is from the setpoint and changes the CO to reach the setpoint as fast as possible. The derivatie term acts against the oscillations caused by the proportional term, because it considers how fast the e(t) is changing. The integral term considers how long and how far PV has been from SP by continually summing e(t) over time, so it acts like a moving bias that eliminates the offset.

$$u(t) \quad = K_P e(t) + K_I \int_0^t e(x)\,\mathrm{d}x + K_D \frac{\mathrm{d}e(t)}{\mathrm{d}t}$$



Picture 6: PID step response graph

3.3 Algorithm

Most of the code presented in this section was originally written for arduino, thus the type double has the same meaning as float.

Basically the same algorithm is used as in the previous Documentation. A really simple implementation of an PID controller is the following code:

```
/*working variables*/
unsigned long lastTime;
double Input, Output, Setpoint;
double errSum, lastErr;
double kp, ki, kd; // set this to the appropriate values
void Compute()
{
   /*How long since we last calculated*/
```

```
    unsigned long now = millis();
    double timeChange = (double)(now - lastTime);

    /*Compute all the working error variables*/
    double error = Setpoint - Input;
    errSum += (error * timeChange);
    double dErr = (error - lastErr) / timeChange;

    /*Compute PID Output*/
    Output = kp * error + ki * errSum + kd * dErr;

    /*Remember some variables for next time*/
    lastErr = error;
    lastTime = now;
 }
```

Source: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

To be more robust some improvements are done:

3.3.1 Derivative kick problem

The formula for the derivative term is the derivation of the error value.

$$\frac{de(t)}{dt}$$

Since the error value is defined by:

$$e(t) = SP - PV(t)$$

it means the following for the derivative:

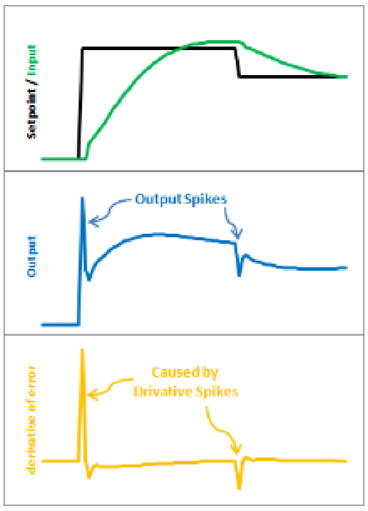
$$\frac{de(t)}{dt} = \frac{d(SP - PV(t))}{dt}$$

When it comes to a setpoint change, the error increases rapidly and results in a spike of the deriative, as shown in picture 6. It is called the derivative kick.

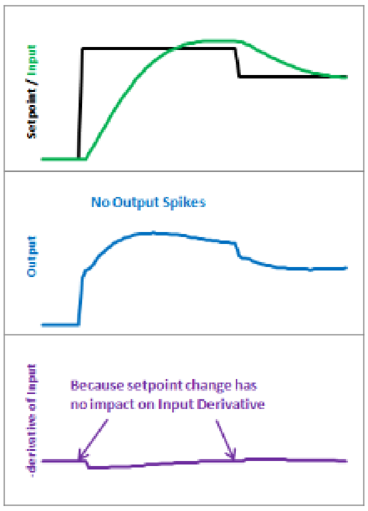
Due to the fact, that the derivative part only needs to consider the change of the process variable, the derivative kick can be prevented by eliminating SP from the formula.

$$SP = 0 \rightarrow \frac{de(t)}{dt} = \frac{d(-PV(t))}{dt} = \frac{dPV}{dt}$$

That means that the derivative depends only on the last input. This is called "Derivative on Measurement". In picture 7 you can see that the setpoint change doesn't cause spikes anymore.



Picture 7: derivative kick



Picture 8: Correction of derivative kick

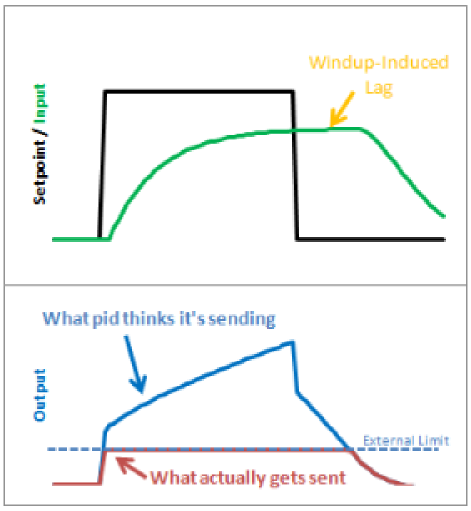
In our code it looks like:

```
Derivation kick prevention

/*working variables*/
double errSum, lastInput;
[...]
void Compute()
{
    [...]
    /*Compute all the working error variables*/
    double error = Setpoint - Input;
    errSum += error;
    double dInput = (Input - lastInput);

    /*Compute PID Output*/
    Output = kp * error + ki * errSum - kd * dInput;

    /*Remember some variables for next time*/
    lastInput = Input;
    [...]
}
[...]
```

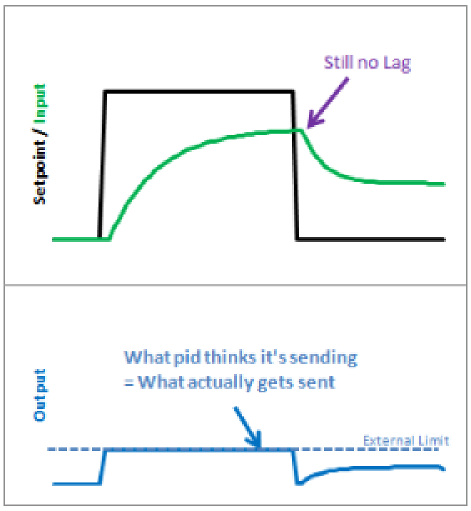
3.3.2 Windup reduction

One possible problem during controlling is the windup.

If the setpoint is not reachable by the CO (e.g. due a restriction in the engine controller), after a while the integral adds up to a very large number. Because of that, the CO grows to exceed the restriction and the desired value isn't reachable by the controller hardware. This is called windup. As a consequence, the controller can't regulate the process until the error changes sign and the integral term shrinks sufficiently so that the CO value is in the range again. Therefore the controller reacts much later on a setpoint change, so it leads to a windup-induced lag as shown in picture 8.



Picture 9: windup lag



Picture 10: Correction of windup lag

To prevent the windup it is necessary to let the controller know about the restriction in output. This can be in a different unit as the actual restriction due conversions in the system output loop (e.g. in our system the limit is the motor voltage but for the control loop it can be a imaginary restriction as the output is formatted, recalculated to both engines, send to speed controller and then it is the actual voltage).

Implementation is simply clamp the calculation to the restriction as suggested:

```
Windup reduction

[...]
double outMin, outMax;
void Compute()
{
  [...]
  ITerm+= (ki * error);

  //************* HERE *************
  if(ITerm> outMax) ITerm= outMax;
  else if(ITerm< outMin) ITerm= outMin;
  double dInput = (Input - lastInput);

  /*Compute PID Output*/
  Output = kp * error + ITerm - kd * dInput;

  //************* HERE *************
  if(Output > outMax) Output = outMax;
  else if(Output < outMin) Output = outMin;
  [...]
}
[...]

void SetOutputLimits(double Min, double Max)
{
   if(Min > Max) return;
   outMin = Min;
   outMax = Max;

   //************* HERE *************
   if(Output > outMax) Output = outMax;
   else if(Output < outMin) Output = outMin;

   if(ITerm> outMax) ITerm= outMax;
   else if(ITerm< outMin) ITerm= outMin;
}
```

3.3.3 Sample time awareness

The presented code is designed to be called in a regular interval. If the environment can't guarantee this the implementation should be aware of the sample time and handle it accordingly. This is necessary as the I and D part relies on the time. The Blog suggests to handle interval generation by the controller itself:

```
Sample time awareness

[...]
int SampleTime = 1000; //1 sec
void Compute()
{
    unsigned long now = millis();
    int timeChange = (now - lastTime);
    if(timeChange>=SampleTime)
    {
          [...] Compute [...]
    }
}
void SetTunings(double Kp, double Ki, double Kd)
{
  [...]
    ki = Ki * SampleTimeInSec;
    kd = Kd / SampleTimeInSec;
}
[...]
```

In our time triggered environment this method isn't applicable. If the call of the controller is too irregular a possible implementation like the suggested is to calculate the exact Ki value (as in line 14-15) in each loop based on the time since the last call. But this way isn't preferred - we should do anything to archive a regular call time.

3.4 Adjustable PID parameters during runtime

One improvement mentioned in the previous Documentation is to adjust the control parameters on the fly. In our view it is necessary to archive this to be able to find appropriate values for the ACD.

The first idea was to use an additional serial port on the STM32 as control port. But this may cause timing issues and adds unknown amount of complexity. An additional approach is to use some Hardware to change the parameters on the fly. An other solution is to add a potentiometer and use the digitalized value to select a value from a range of possible values. This approach is better controllable and fits better into the time triggered architecture with time slots. To gain optimal flexibility we add three potentiometers to the microcontroller and assign them to each parameter. To get the actual values a button is added which toggles between to system states: Running and Maintenance. The switch between the two modes is done by switching two main loops. This is done by FreeRTOS Group. In the Maintenance Loop we will print the actual PID Parameters to the control serial port.

3.4.1 Setup

The used STM32 microcontroller board has an additional button for general purposes. This button is labeled as "B1 (USER)", has the color Blue on the Board and is available as PC13 on the STM32 (see Pictures adjacent). The final implementation should toggle the mode between Running and Maintenance with each button press.

For the Running mode is characteristic that every task has to meet the time slot requirements. In the Maintenance mode this constrain does not apply. Therefore the "slow" output of the parameters is only there possible.

Scaling is done by following formula (assuming R is the input value and P the parameter):

$$P(R) := \frac{(P_{End} - P_{Start}) \cdot R}{1024} + P_{Start}$$

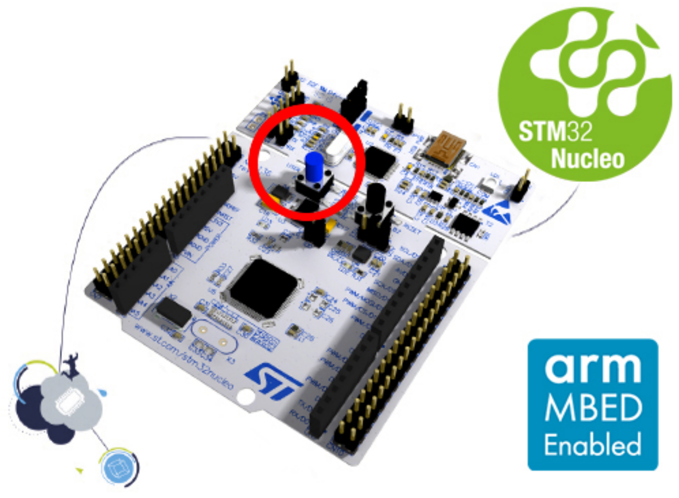
This is done for the P, I and D parameter.

For simplicity the start and end values are hardcoded - as improvement it could be done in maintenance mode through command console.

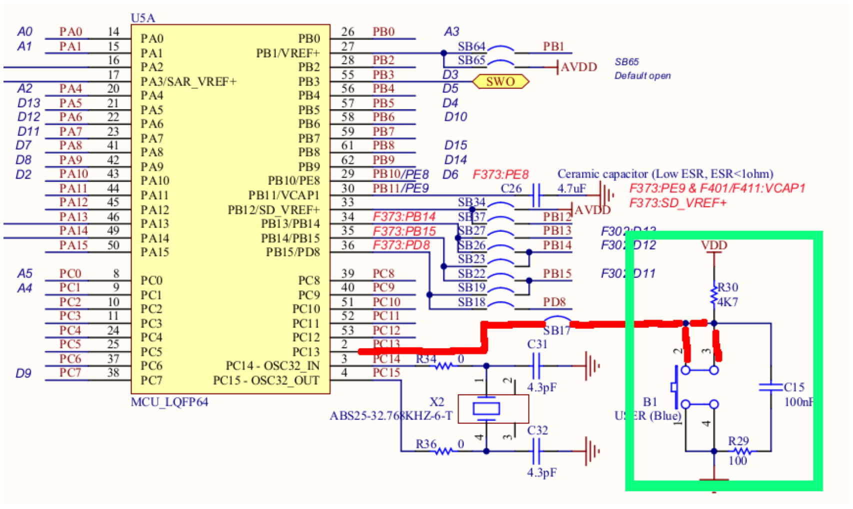
3.4.2 Parameter tuning

To adapt the PID controller to the hardware system, the parameters used in the controller functions must be adapted to the system. One way of doing this is to add hardware to the system, which makes it possible to adjust the parameters during runtime. For this purpose potentiometers were used (Picture 14 and Picture 15) which influence the parameters Kp, Ki and Kd of the used control loop.

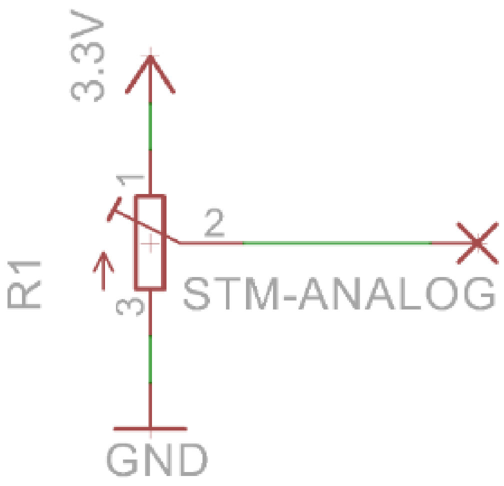
The potentiometers are connected as voltage divider. Each tap is connected to one analog input. The voltage is divided in the range [0,..,3.3V]. After digitalizing the software sees the 12 bit representation [0,1,...,4096]. This value is used to scale the parameters in a defined range.



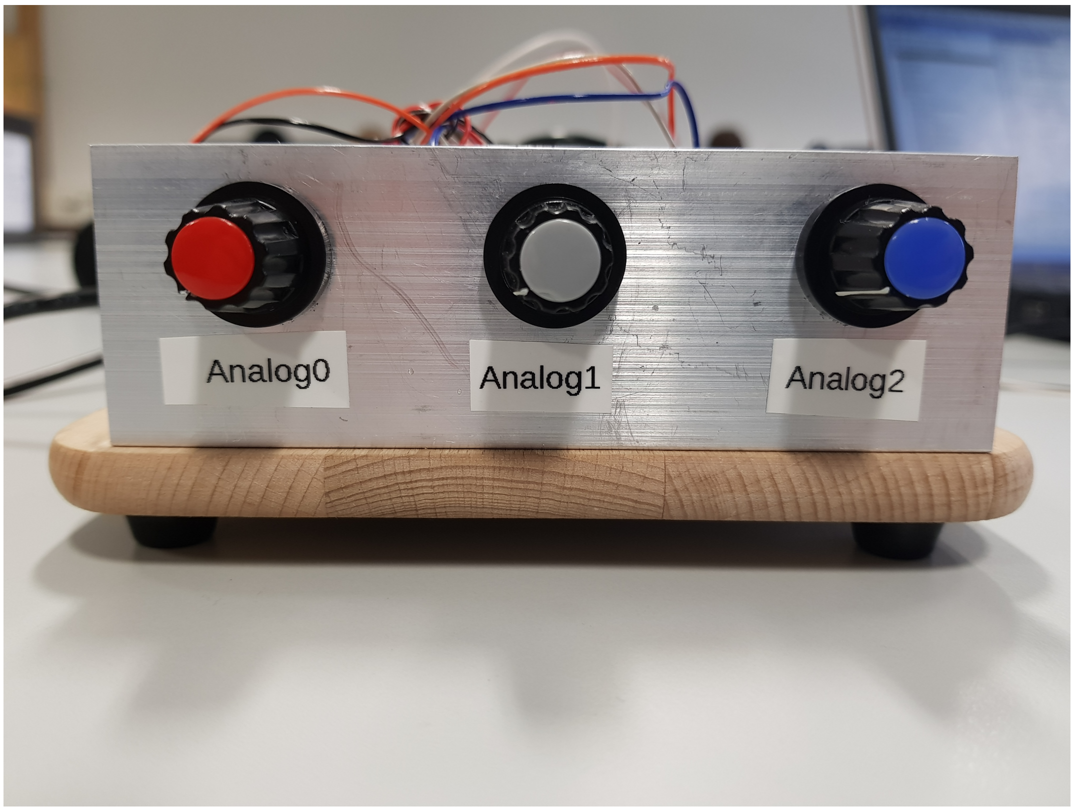
Picture 11: NUCLEO STM32-L152RE Board with USER Button marked



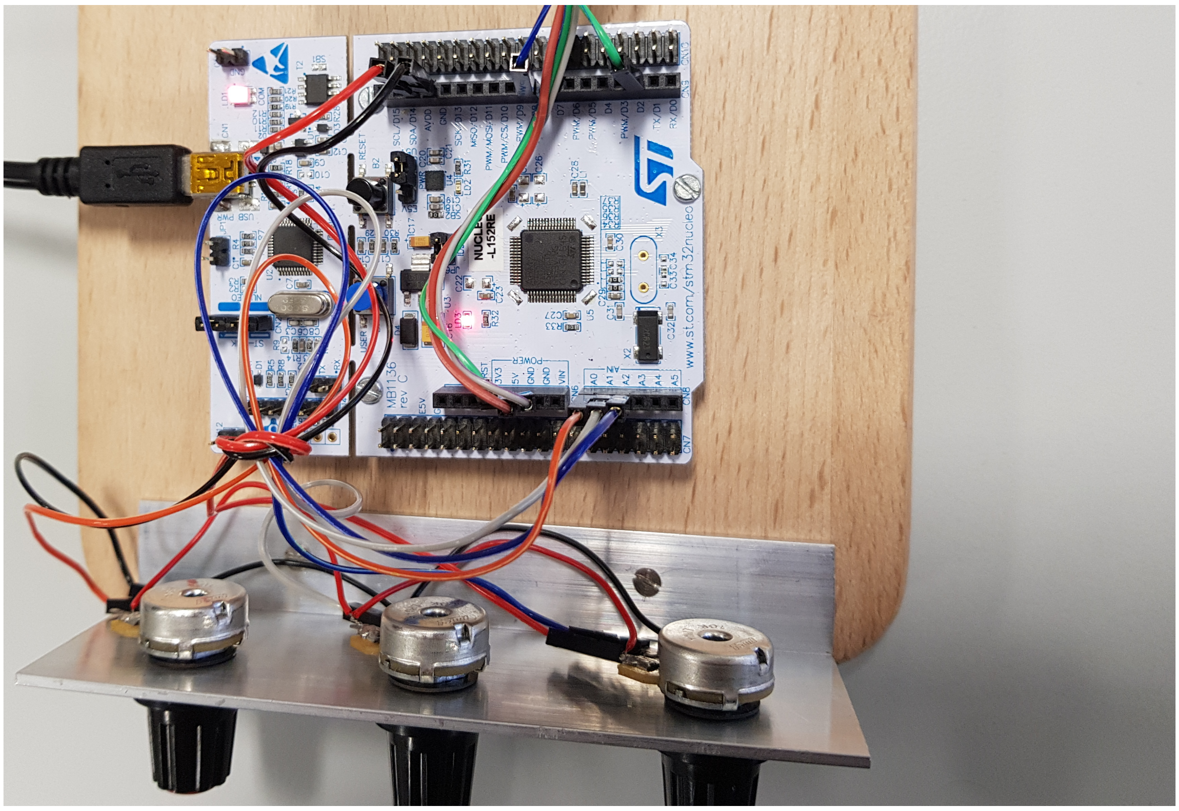
Picture 12: Schematic of B1



Picture 13: Connection of potentiometer



Picture 14: Potentiometer for analog pin 0 to 2


Picture 15: Potentiometer connected to the system

3.4.3 Problems changing parameters during run time

Using the basic code presented above and changing parameters during run time results in spikes in the output. This is due the change of multiplier in calculation formula. For example changing the I parameter to half will cause the errSum be halved.

```
Output = kp * error + ki * errSum + kd * dErr;
```

To prevent this error the errSum should already contain the scaled value.

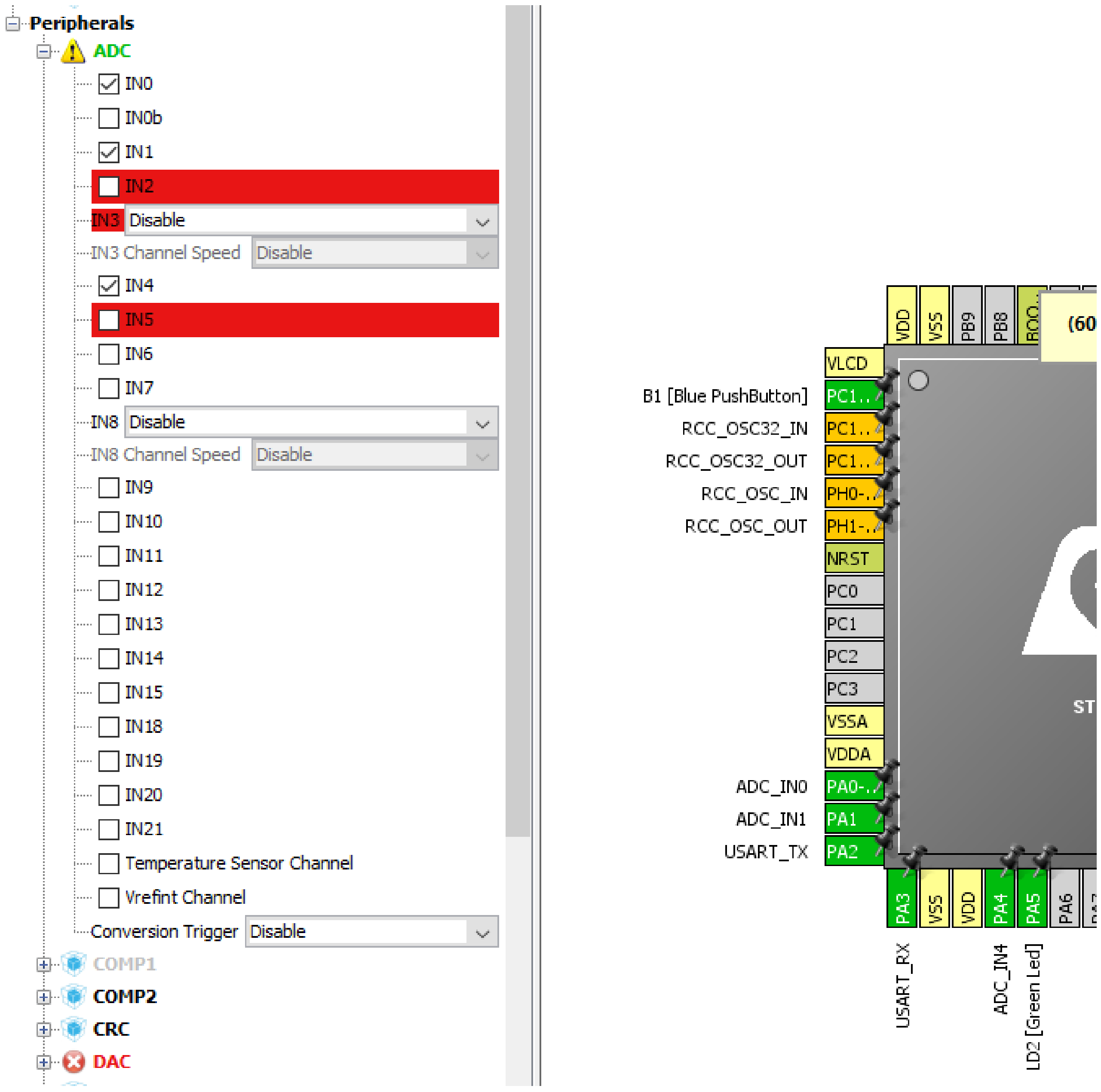
```
ITerm += (ki * error);
Output = kp * error + ITerm - kd * dInput;
```

Now changing the I Parameter will slowly take effect and will not cause a jump in output.

3.5 Implementation

3.5.1 Read analog values from ADC

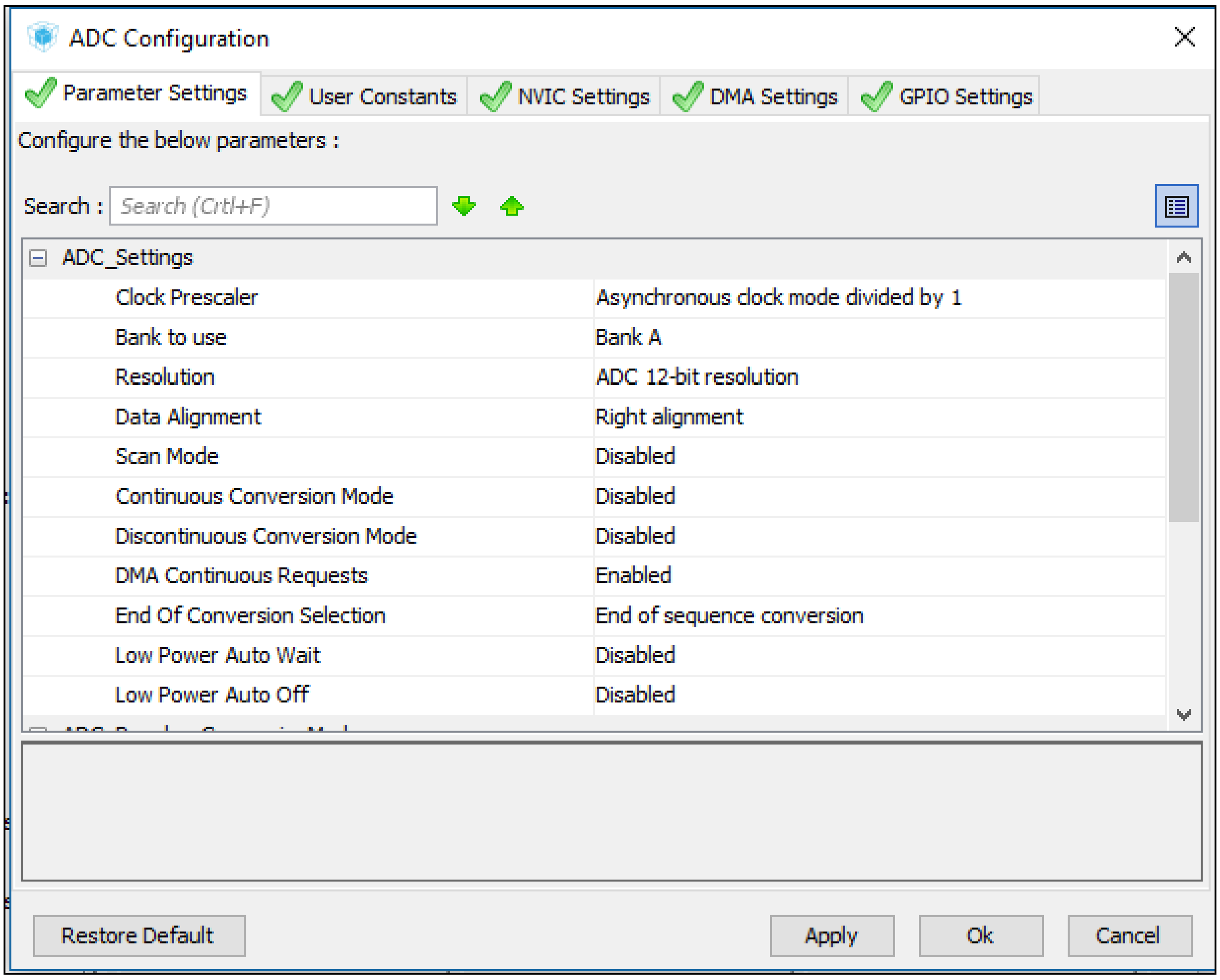
To be able to read analog values, the analog inputs must first be activated in the project. For this purpose, as shown in Picture 16, in the STM32CubeMX project under the tab "Pinout" under "Peripherals","ADC" the analog inputs have to be activated, from which the readout should be possible. If it's done correctly, the selected pins should be marked as green and pinned. In this case "IN0", "IN1" and "IN4".



- Peripherals
  - ADC
    - [✓] IN0
    - [ ] IN0b
    - [✓] IN1
    - [ ] IN2
    - IN3: Disable
    - IN3 Channel Speed: Disable
    - [✓] IN4
    - [ ] IN5
    - [ ] IN6
    - [ ] IN7
    - IN8: Disable
    - IN8 Channel Speed: Disable
    - [ ] IN9
    - [ ] IN10
    - [ ] IN11
    - [ ] IN12
    - [ ] IN13
    - [ ] IN14
    - [ ] IN15
    - [ ] IN18
    - [ ] IN19
    - [ ] IN20
    - [ ] IN21
    - [ ] Temperature Sensor Channel
    - [ ] Vrefint Channel
    - Conversion Trigger: Disable
  - COMP1
  - COMP2
  - CRC
  - DAC

Picture 16: Configure analoge pinouts

If this has been done, you can make some settings under the tab "Configuration" in the field "Analog" - "ADC", which are needed for the readout procedure used here. Picture 17 and Picture 18 show the settings made.

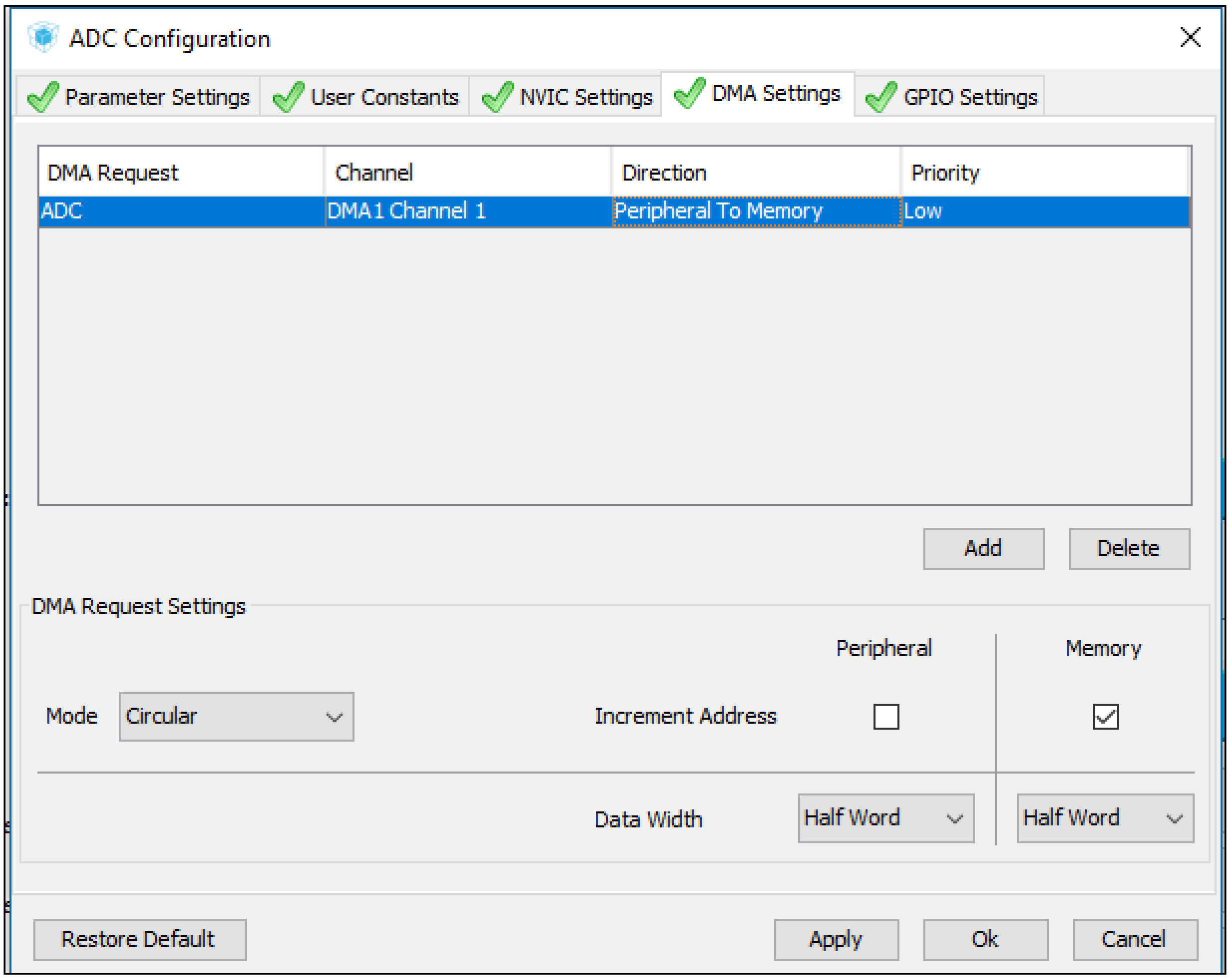


Configure the below parameters :

ADC_Settings

Clock Prescaler	Asynchronous clock mode divided by 1
Bank to use	Bank A
Resolution	ADC 12-bit resolution
Data Alignment	Right alignment
Scan Mode	Disabled
Continuous Conversion Mode	Disabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Enabled
End Of Conversion Selection	End of sequence conversion
Low Power Auto Wait	Disabled
Low Power Auto Off	Disabled

Picture 17: ADC-Configuration parameter setting


ADC Configuration

Parameter Settings | User Constants | NVIC Settings | DMA Settings | GPIO Settings

DMA Request	Channel	Direction	Priority
ADC	DMA1 Channel 1	Peripheral To Memory	Low

DMA Request Settings

		Peripheral	Memory
Mode: Circular	Increment Address	☐	☑
	Data Width	Half Word	Half Word

Picture 18: ADC-Configuration DMA-Settings

A method readAnalogPotis (Codeblock "Read analog values") was created to read out the analog values. This method has a return value as a structure in which the three values are stored.
First, the respective ADC pin is selected alternately with a function, followed by the readout of the available value and writing into an array by means of HAL_ADC_Start_DMA.

Read analog values

```
[...]
potiReadings_t readAnalogPotis() {
    potiReadings_t returnValues;
    short ADC1convertedValues[3];
    SelectAdc0(hadc);
    if(HAL_ADC_Start_DMA(&hadc, (short*)ADC1convertedValues, 3) != HAL_OK){
        returnValues.poti1 = 0;
        returnValues.poti2 = 0;
        returnValues.poti3 = 0;
    }
    SelectAdc1(hadc);
    if(HAL_ADC_Start_DMA(&hadc, (short*)ADC1convertedValues, 3) != HAL_OK){
        returnValues.poti1 = 0;
        returnValues.poti2 = 0;
        returnValues.poti3 = 0;
    }
    SelectAdc4(hadc);
    if(HAL_ADC_Start_DMA(&hadc, (short*)ADC1convertedValues, 3) != HAL_OK){
        returnValues.poti1 = 0;
        returnValues.poti2 = 0;
        returnValues.poti3 = 0;
    }
    returnValues.poti1 = ADC1convertedValues[0];
    returnValues.poti2 = ADC1convertedValues[1];
    returnValues.poti3 = ADC1convertedValues[2];

    return returnValues;
}
```

```
Select an analog port

[...]
void SelectAdc0() {
    ADC_ChannelConfTypeDef sConfig0;
    sConfig0.Channel = ADC_CHANNEL_0;
    sConfig0.Rank = 1;
    sConfig0.SamplingTime = ADC_SAMPLETIME_4CYCLES;
    if (HAL_ADC_ConfigChannel(&hadc, &sConfig0) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }
}
[...]
```

3.5.2 Structural overview of final code

The code is divided into three different main "regions" (each one in a single file). To help others to understand this construct we will introduce every region shortly here:

- "pid_controller_internal": This file contains the main logic of PID controller. The function "Compute(..)" contains the part of code described earlier - but adjusted to match the requirements on the loop in this project.
- "pid_controller": This file is the "control and interface" file for the PID controller. Here are clamps calculated, analog values read and the underlying functions are called. If anyone needs a PID controller he could use these two files and implement it in his design.
- "pid_wrapper.c": This file is a wrapper around pid_controller to meet special requirements for ACL 2.1. It serves as swtiching layer between the communication with global variables and the concept of passing all values used in the implemented PID controller. It is the interface for the scheduler (the function "void RunGlobal()" reads and sets the global values) and to the message dispatcher.

3.5.3 Values Required and provided by PID controller

The controller needs in this implementation some values to work properly (values in [] are internal stored values and not passed from outside):

- X - Acceleration (2 Bytes)

- Z - Acceleration (2 Bytes)
- Max - Engine value (2 Bytes)
- [current engine 1 (2 Bytes)]
- [current engine 2 (2 Bytes)]

As return values the controller provides:

- Engine 1 (2 Bytes)
- Engine 2 (2 Bytes)

3.6 Useful Links

- Used Board: NUCLEO-L152RE
  - http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-nucleo/nucleo-l152re.html
- Used Microcontroller: STM32-L152RE
  - http://www.st.com/en/microcontrollers/stm32l152re.html
- Basic data types on ARM processors
  - http://www.keil.com/support/man/docs/armcc/armcc_chr1359125009502.htm

3.7 Sources

[Pont2001] : http://www.safetty.net/download/pont_pttes_2001.pdf

[Wikipedia] : https://en.wikipedia.org/wiki/PID_controller

Picture 1: https://en.wikipedia.org/wiki/Control_theory#/media/File:Feedback_loop_with_descriptions.svg, Last Visit: 18.12.17

Picture 3: https://de.wikipedia.org/wiki/Regler#/media/File:Idealer_P_Sprungantwort.svg, Last Visit: 18.12.17

Picture 4: https://de.wikipedia.org/wiki/Regler#/media/File:Idealer_I_Sprungantwort.svg, Last Visit: 18.12.17

Picture 5: https://de.wikipedia.org/wiki/Regler#/media/File:Idealer_D_Sprungantwort.svg, Last Visit: 18.12.17

Picture 6: https://www.researchgate.net/figure/289531077_fig22_Figure-24-Step-Response-of-a-Proportional-Integral-Derivative-PID-Controller, Last Visit: 18.12.17

Picture 7: http://brettbeauregard.com/blog/2011/04/improving-the-beginner%E2%80%99s-pid-derivative-kick/, Last Visit: 18.12.17

Picture 8: http://brettbeauregard.com/blog/2011/04/improving-the-beginner%E2%80%99s-pid-derivative-kick/, Last Visit: 18.12.17

Picture 9: http://brettbeauregard.com/blog/2011/04/improving-the-beginner%e2%80%99s-pid-reset-windup/, Last Visit: 18.12.17

Picture 10: http://brettbeauregard.com/blog/2011/04/improving-the-beginner%e2%80%99s-pid-reset-windup/, Last Visit: 18.12.17

Picture 11: http://www.st.com/content/ccc/fragment/product_related/rpn_information/board_photo/9e/75/14/86/ee/4a/43/78/nucleo-Lx.jpg/files/nucleo-Lx.jpg/_jcr_content/translations/en.nucleo-Lx.jpg, Last Visit: 11.12.2017

Picture 12: http://www.st.com/resource/en/schematic_pack/nucleo_64pins_sch.zip, Last Visit: 11.12.2017

4 Driver for Arduino-compatible Microcontroller

In this section we describe the arduino drivers for the two motors "Crazyflie Nano Quadcopter", the attitude sensor "MPU-6050" and the RS232 Driver.

4.1 Contents

4.2 Attitude Sensor MPU-6050

Authors: Christian Zöller, Eike Diekmann, Hermann Wafo

4.2.1 Configuration

We use the following registers of the I2C device as MPU-6050 Configuration.

Register 0x6B/107 – Power Management 1

Bits	Name	Description	Configuration	Justification
7	DEVICE_RESET	When set to 1, this bit resets all internal	1	1 to reset the sensor initially.

Bits	Name	Description	Configuration	Justification
		registers to their default values.		
6	SLEEP	When set to 1, this bit puts the MPU-60X0 into sleep mode.	0	0 since we want to receive sensor data repetitively.
5	CYCLE	When this bit is set to 1 and SLEEP is disabled, the MPU-60X0 will cycle between sleep mode and waking up to take a single sample of data from active sensors at a rate determined by LP_WAKE_CTRL (register 108).	0	0 since we want to receive sensor data repetitively and control the timing on our own.
4	-	-	-	-
3	TEMP_DIS	When set to 1, this bit disables the temperature sensor.	1	1 since we don't need the temperature.
2,1,0	CLKSEL[2:0]	3-bit unsigned value. Specifies the clock source of the device.	0	0 for "Internal 8MHz oscillator".

Register 0x1C/28 – Accelerometer Configuration

Bits	Name	Description	Configuration	Justification
7	XA_ST	When set to 1, the X- Axis accelerometer performs self test.	0	No self test.
6	YA_ST	When set to 1, the Y- Axis accelerometer performs self test.	0	No self test.
5	ZA_ST	When set to 1, the Z- Axis accelerometer performs self test.	0	No self test.
4,3	AFS_SEL[1:0]	2-bit unsigned value. Selects the full scale range of accelerometers.	0 for ± 2g	Not specified yet
2,1,0	-	-	-	-

Register 0x1B/27 – Gyroscope Configuration

Bits	Name	Description	Configuration	Justification
7	XG_ST	Setting this bit causes the X axis gyroscope to perform self test.	0	No self test.
6	YG_ST	Setting this bit causes the Y axis gyroscope to perform self test.	0	No self test.
5	ZG_ST	Setting this bit causes the Z axis gyroscope to perform self test.	0	No self test.
4,3	FS_SEL[1:0]	2-bit unsigned value. Selects the full scale range of gyroscopes.	± 250 °/s	Not specified yet
2,1,0	-	-	-	-

4.2.2 Interfaces for Sensor Operation

Code

```
...
class SensorDriver {
    private:
    /*
    ....
    */

    public:
    SensorDriver(I2C &i2c, const uint8_t timeout);
    /**
    * initializes Sensor
    */
    void init();
    /**
    * Get X-axis acceleration reading
    * @return X-axis acceleration measurement in 16 bit 2's complement format.
    */
    int16_t getAccelerationX();
    /**
    * Get Y-axis acceleration reading
    * @return Y-axis acceleration measurement in 16 bit 2's complement format.
    */
    int16_t getAccelerationY();
    /**
    * Get Z-axis acceleration reading
    * @return Z-axis acceleration measurement in 16 bit 2's complement format.
    */
    int16_t getAccelerationZ();
    /**
    * Update sensordata.
    */
    void update();
    /**
    * Get return status
    * @return return status, 0 = success
    */
    uint8_t getReturnStatus();
};
```

Usage

This class is optimized for using within a scheduler.
For instantiation there will be needed the I2C and a maximum time (timeout) this task is allowed to get sensor data.
The scheduler first calls ***update()*** to get current sensor values.
After updating, sensor values can received by calling ***getAccelerationX()*** and/or ***getAccelerationY()*** and/or ***getAccelerationZ()***.
Calling ***getReturnStatus()*** reveals whether there occured an error or not. 0 = no Error.

Timeout

The scheduler runs the sensor driver and is awaiting an answer within a certain time. If the sensor driver can't receive an answer it must not block the software.
Therefore there is a Timeout. The Timeout is defined by the Scheduler and commited to the Sensor Driver Constructor.
If the defined time is over, the Sensor Driver stops the attempt receiving any further data.
If a timeout or an error occured can be read by calling the method *getReturnStatus()*.

4.2.3 Example Program for Reading Sensor Values

To figure out which values are relevant to distinguish the direction of the tilt a simple program was created. It prints the read acceleration values of X, Y, Z and converted values to angles.

Code

```
/*Begining of Auto generated code by Atmel studio */
#include <Arduino.h>
/*End of auto generated code by Atmel studio */

#include "I2C/I2C.h"
#include "SensorDriver/SensorDriver.h"

// Declared in I2C-Library globally
extern I2C I2c;

SensorDriver sensorDriver = SensorDriver(I2c, 100);

void setup()
{
    Serial.begin(115200);
}
```

```
void loop()
{
    Serial.print("Start");
    sensorDriver.update();

    int16_t accelX = sensorDriver.getAccelerationX();
    Serial.print(" | X = "); Serial.print(accelX);
    int16_t accelY = sensorDriver.getAccelerationY();
    Serial.print(" | Y = "); Serial.print(accelY);
    int16_t accelZ = sensorDriver.getAccelerationZ();
    Serial.print(" | Z = "); Serial.print(accelZ);
    int16_t returnStatus = sensorDriver.getReturnStatus();
    Serial.print(" | returnStatus = "); Serial.print(returnStatus);

    Serial.print(" | Grad X = "); Serial.print((RAD_TO_DEG * (atan2(-accelY, -accelZ) + PI ))
-180);
    Serial.print(" | Grad Y = "); Serial.print((RAD_TO_DEG * (atan2(-accelX, -accelZ) + PI ))
-180);
    Serial.print(" | Grad Z = "); Serial.println((RAD_TO_DEG * (atan2(-accelY, -accelX) + PI
))  -180);

    delay(200);
}
```

Example

This example is located at acd/Arduino/examples/ExampleSensorData.

4.3 Grove - I2C Motor Driver V1.3

Authors: Christian Zöller, Eike Diekmann, Hermann Wafo

4.3.1 Re-engineering of existing Motor Driver

Since the existing motor driver doesn't meet realtime systems requirements, it has been necessary to think about using appropriate I2C library with functionalities that have got optimized for our purpose. The selected I2C library needs to fit the structure and functionalities of the existing motor driver, so it brings us to the point of writing needed functions within it. Then we rewrote the existing motor driver using fitted I2C library.

The resulting structure of the motor driver differs just a little bit from the existing one, but the most important functionalities are well-covered.

4.3.2 Interfaces for Motor Operation

```
...
class MotorDriver {
    private:
/*...*/

    public:
            MotorDriver(I2C &i2c, const uint32_t timeout);
            void initMotors(void);
            // Initialize I2C with an I2C address you set on Grove - I2C Motor Driver v1.3
            // default i2c address: 0x0f
            void begin(unsigned char i2c_add);
            // Set the direction of both motors
            // _direction: BothClockWise, BothAntiClockWise
            uint8_t direction(unsigned char _direction);
            //Sets speed for Motors.
            uint8_t setSpeed(uint8_t speedLeft, uint8_t speedRight);
            //Update speed of Motor with set speed
            void update();
            //get returnStatus: returns ErrorCode: 0 -> No Error, >0 Error
            uint8_t getReturnStatus();
            // Set the frequence of PWM(cycle length = 510, system clock = 16MHz)
            // F_3921Hz is default
            // _frequence: F_31372Hz, F_3921Hz, F_490Hz, F_122Hz, F_30Hz
            uint8_t frequence(unsigned char _frequence);
            // Stop motors
            void stop();
};
```

Usage

This class is optimized for using within a scheduler.
For instantiation there will be needed the I2C and a maximum time (timeout) this task is allowed to get sensor data.
The scheduler first calls ***update()*** to get current sensor values.
After updateing, sensor values can set by calling ***setSpeed(uint8_t speedLeft, uint8_t speedRight)*** where speedLeft and speedRight is speciefied as the speed of correspondig motor (0-255).
Calling ***getReturnStatus()*** reveals whether an error occured or not. 0 = no Error.

Timeout

The scheduler runs the sensor driver and is awaiting an answer within a certain time. If the sensor driver can't receive an answer it must not block the software.
Therefore there is a timeout. The timeout is defined by the scheduler and passed to the Motor Driver Constructor.
If the defined time is exceeded, the Motor Driver stops the attempt sending any further data.
If a timeout or an error occured it can be read by calling the method ***getReturnStatus()***.

4.3.3 Mapping of the motor values

The speed range of the two motors(Motor Crazyflie Nano Quadcopter) is the same 0 to 255. Both motors speeds can be accessed from outside and setted directly using the speed function, that returns a status 0 when the function successful completed without timeout occured. Statuses 1 to 7 are also returned when waiting for completion/ACK/NACK while adressing slave, sending/receiving data to/from slave. It's also possible to get other statuses 8 to 0xFF which are well-documented in the datasheet of the used 8-bit Mikrocontroller ATmega2560.

As illustration, we can take a look at the following simple example for testing of the rewritten motor driver:

Example of setting motors speeds using the motor driver

```
#include <Arduino.h>

#include "I2C/I2C.h"
#include "MotorDriver/MotorDriver.h"

// Declared in I2C-Library globally
extern I2C I2c;

MotorDriver motorDriver = MotorDriver(I2c, 200);  // bind the scheduler timeout delay to
motorDriver instance

void setup()
{
    Serial.begin(115200);
    motorDriver.initMotors();
}

void loop()
{
    Serial.write("Status returned by update function: ");
    motorDriver.setSpeed(63, 127); // map motor left with 25 percent duty cycle and motor right
with 50 percent duty cycle
    uint8_t returnStatus = motorDriver.update(); // update motor left and right now, and load
the returned status to a local variable
    Serial.println(returnStatus); // print for debugging this local variable to the serial
monitor
    delay(1000);
}
```

Path for motor driver example code

A much more complete example is located at acd/Arduino/ExampleMotorDriver/ExampleMotorDriver /ExampleMotorDriverData/ExampleMotorDriverData.cpp

4.3.4 Error Handling

To allow proper error handling there is a getReturnStatus() method in the SensorDriver and the MotorDriver as well. Both return to following status codes:

Code	Type	Description
0	Success	Function executed with no errors
1	Timeout	Waiting for successful completion of a Start bit
2	Timeout	Waiting for ACK/NACK while addressing slave in transmit mode (MT)

Code	Type	Description
3	Timeout	Waiting for ACK/NACK while sending data to the slave
4	Timeout	Waiting for successful completion of a Repeated Start
5	Timeout	Waiting for ACK/NACK while addressing slave in receiver mode (MR)
6	Timeout	Waiting for ACK/NACK while receiving data from the slave
7	Timeout	Waiting for successful completion of the Stop bit
8 - 0xFF	Other	See datasheet of microcontroller for exact meaning

4.4 RS232 Driver

Authors: Sercan Catalkaya, Daniel Kunde

In this section of the project, you can find information about the Arduino Serial library and whether an interrupt is required outside of the Serial.begin() and Serial.end() functions.

Arduino Serial library

The Serial library is used for communication between Arduino Board and FreeRTOS device with the interface RS-232.

https://www.arduino.cc/reference/en/language/functions/communication/serial/

These following functions are important for the RS-232 Driver:

Serial.begin(): Sets the transmission rate in baud and opens the serial port.

https://www.arduino.cc/en/Serial/Begin

Serial.end(): Close the serial port.

https://www.arduino.cc/en/Serial/end

Serial.setTimout(time): Set the maximum wait time for serial data before the connection closed.

https://www.arduino.cc/en/Serial/SetTimeout

Serial.read(): Reads received serial data.

https://www.arduino.cc/en/Serial/read

Serial.print(): Prints ASCII text to serial port.

https://www.arduino.cc/en/Serial/print

Interrupts:

We presume that there will be no additional interrupts used outside of the Serial.begin() and Serial.end() functions.

https://www.aulis.hs-bremen.de/goto.php?target=file_853355_download&client_id=hsbremen

4.4.1 Test of Serial Connection

RS-232 was tested via USB and the Serial1 interface on ports RX1 and TX1. Both were tested with baudrate 9600 using the following c-code:

```
SerialTest

void setup() {
    Serial1.begin(9600);
    Serial.begin(9600);
}

void loop() {
    Serial1.println("Hello");
    Serial.println("Hello");
    delay(200);
}
```

Sending data

To test the communication between the Arduino ATmega2560 and the computer, we first had to connect them using the RS232 cable.

Using the INCT_PC03 computer, we failed to receive any meaningful data from the Arduino. We used the LogicPort USB Logic Analyzer to verify the cause for this issue.

We tested the connection between the Arduino-LogicPort and the PC-LogicPort. In the first step we tested the PC-LogicPort connection. So we connected the RS232 interface of the PC with the LogicPort.

We used Putty to send data from the PC to the LogicPort and we were able to receive the data in Keil Uvision.

To test the Arduino-LogicPort conenction we connected the cables of the LogicPort with the pin 18 (Tx1) and pin 19 (Rx1) of the Arduino. The SerialTest scetch above was running on the Arduino during this test.

The Keil Uvision settings were not changed for the Arduino-LogicPort and it was not possible to receive any meaningful data.

The reason for this is, that the PC is using negative logic and the Arduino is using positive logic.

After adjusting the settings in Keil Uvison for the RS232 interpreter, by changing it to positive logic, we were able to receive the data from the SerailTest scetch. We reported our findings to the STM microntroller group, so that they could verify which logic their microcontroller uses.

4.4.2 STM32L152 Nucleo-64

Author: Kai Nortmann, Arthur Guz

To establish a communication between the STM32 and the Arduino board, a serial RS232 connection will be used.

To verify, how the STM32 communicates over RS232, we tested the serial connection with the LOGICPORT logic analyzer. This was necessary to make sure, that the communication between the two boards will work without any inverter. We set up the STM32 with following parameters:

```
huart1.Instance = USART1; // use USART1, which means TxD=Pa10 and RxD=Pa9
huart1.Init.BaudRate = 9600;
huart1.Init.WordLength = UART_WORDLENGTH_8B;
huart1.Init.StopBits = UART_STOPBITS_1;
huart1.Init.Parity = UART_PARITY_NONE;
huart1.Init.Mode = UART_MODE_TX_RX;
huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart1.Init.OverSampling = UART_OVERSAMPLING_16;
```

The result of the test was, that the STM32 microcontroller communicates the same way as the Arduino board does. This means, that the boards can communicate with each other without any further inverter hardware needed.

4.4.3 RS232 Communication Protocol

Authors: Daniel Kunde, Sercan Catalkaya

After confirming that both microcontroller use positive logic, we developed the driver for the RS232 interface. The driver provides a function to open the serial connection with 115200 Baud, a function to send and receive data and another function to close the connection.

RS232Driver

```
#include <Arduino.h>

uint8_t message;

void RS232_serial_init(){
    Serial1.begin(115200);
}

void RS232_sendMessage(uint8_t message){
    Serial1.write(message);
}

void RS232_readByte(){
    message = Serial1.read();
}

void RS232_endSerial(){
    Serial1.end();
}

uint8_t RS232_getMessage(){
    return message;
}
```

Das Protokoll speichert die empfangenen daten in einem array, sodass der message dispatcher sie abholen kann. Die festgelete größe der empfangs und send arrays wurde mit den entsprechenden gruppen abgesprochen und beträgt 5.

Our protocol stores the received data from the STM microcontroller in an array. The message dispatcher gets the data from these arrays. The required sizes of the dataReceive and sendData arrays were discussed with the other groups and set to 5.

RS232Protocol

```
/* RS232 Protocol
 *
 * author: Sercan Catalkaya, Daniel Kunde
 *
 */

#include "RS232_driver.h"

#define nReceiveBytes 5
#define nSendBytes 5

uint8_t dataReceive[nReceiveBytes];
uint8_t sendData[nSendBytes];


// Receive Data from STM and save then to array variable dataReceive
void receiveDataFromSTM(){
  int count=0;
  while(Serial.available() > 0){
    RS232_readByte();
    dataReceive[count] = RS232_getMessage();
    count++;
  }
}

// send data to STM
void sendDataToSTM(){
  for(int i=0; i<sizeof(sendData);i++){
    RS232_sendMessage(sendData[i]);
  }
}

// Getter method for the received bytes
uint8_t * getReceiveBytes(){
  return dataReceive;
}

// Setter - Which Bytes to send
void setSendData(uint8_t * dataToSend){
  for(int i = 0; i<sizeof(dataToSend);i++){
    sendData[i]=dataToSend[i];
  }
}
```

Testing the data transfer between Arduino and STM

We connected the Arduino and STM microcontroller using an RS232 cable and tested our protocols. The communication between both microcontrollers worked out and we were able to receive and send data.

Appendix

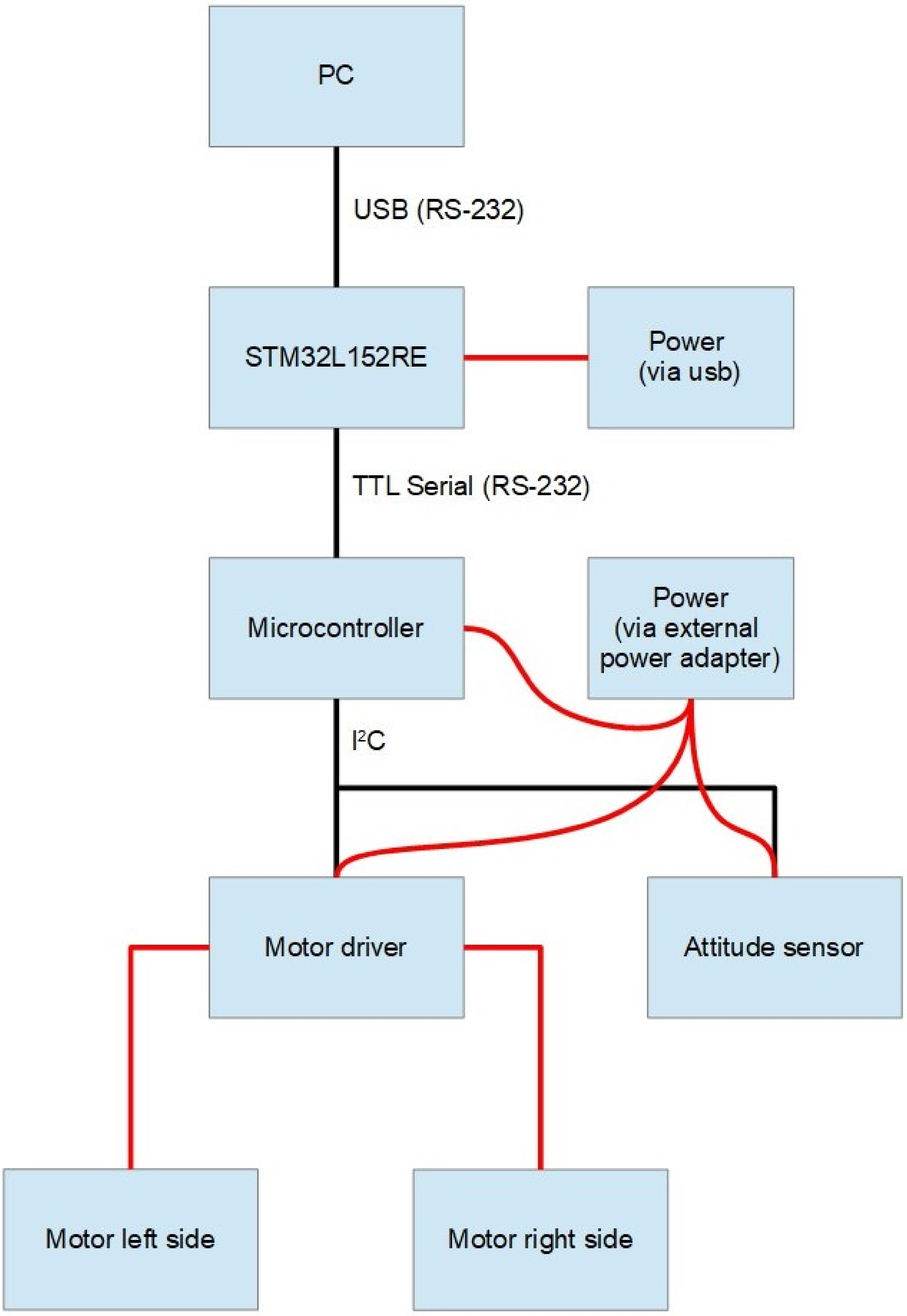
LOGICPORT project file

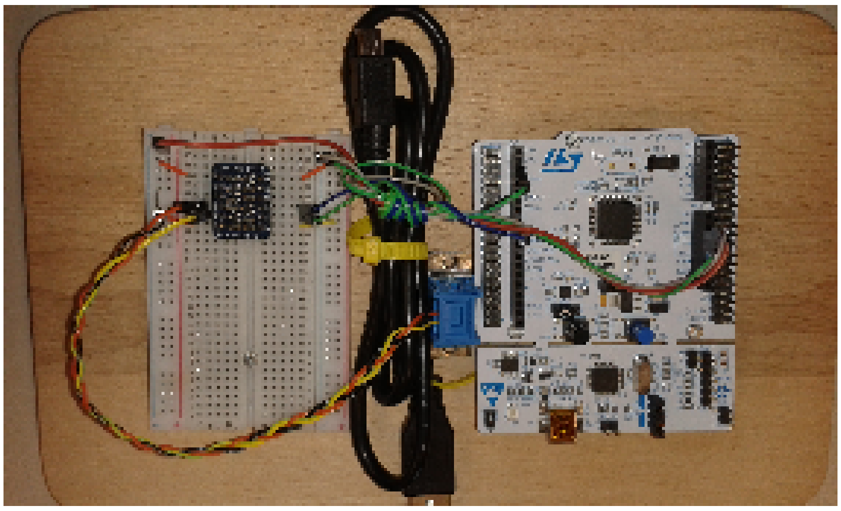
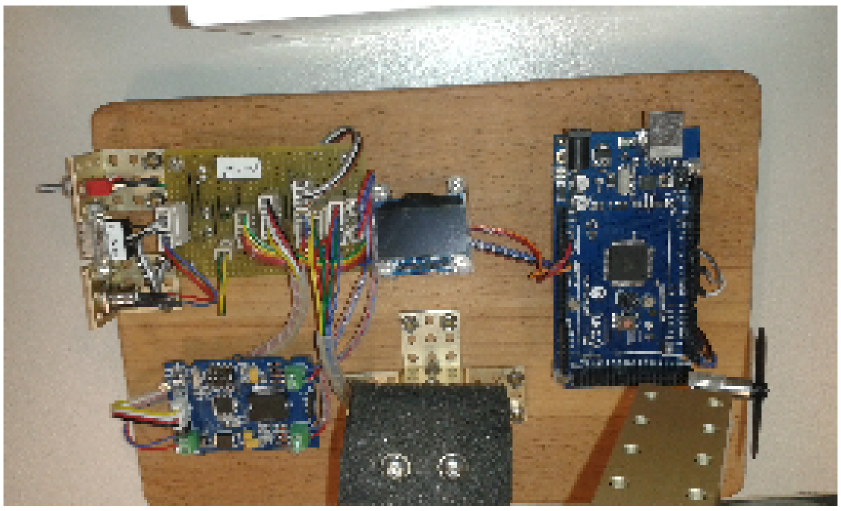
5 Hardware

Authors: Tim Niebuhr, Andre Sarich

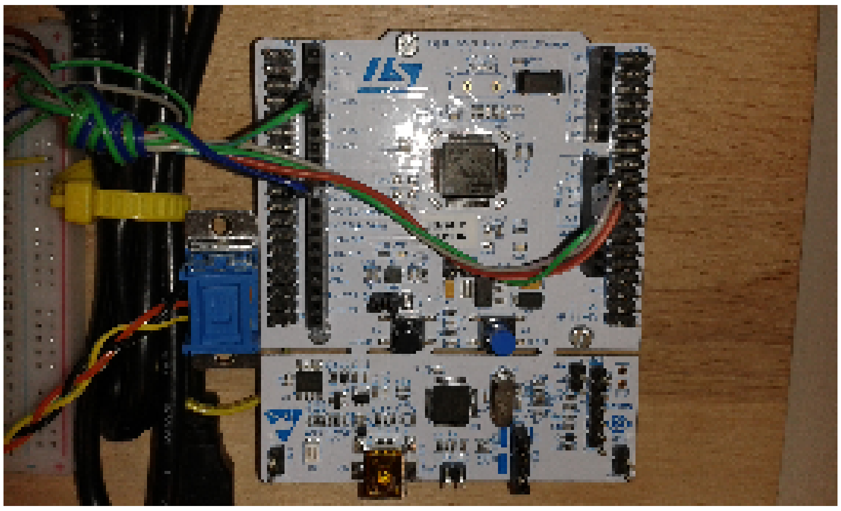
5.1 Contents

5.2 Overview


PC
USB (RS-232)
STM32L152RE
Power (via usb)
TTL Serial (RS-232)
Microcontroller
Power (via external power adapter)
I²C
Motor driver
Attitude sensor
Motor left side
Motor right side

5.3 Hardware components



5.3.1 STM NUCLEO-32L152RE

Name: STM NUCLEO-32L152RE

Data sheet: http://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf

Connector layout shown in data sheet at page 34.

A table with connectors and Pins is shown at page 50.

5.3.2 Microcontroller

Name: Arduino mega 2560

Data sheet: http://www.robotshop.com/media/files/PDF/ArduinoMega2560Datasheet.pdf

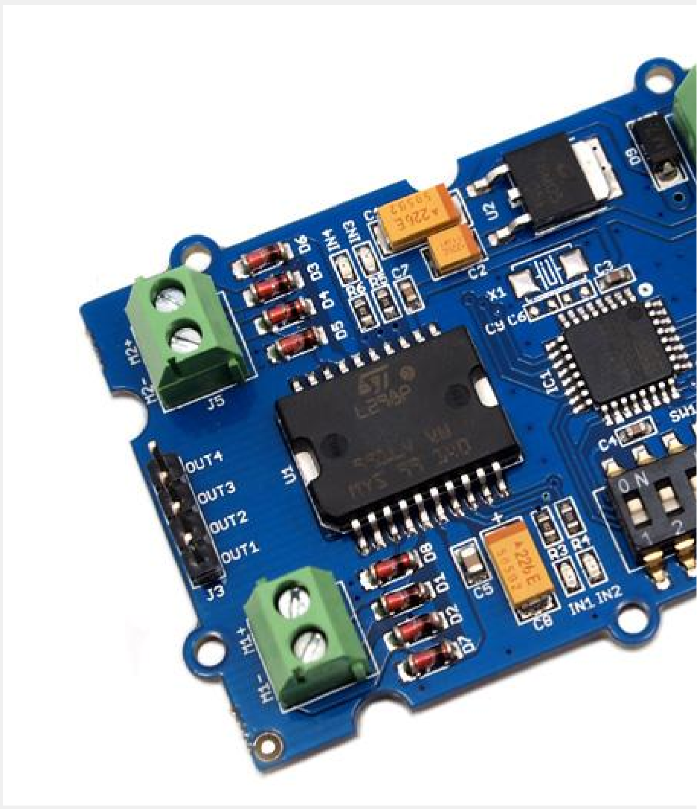
5.3.3 Motor driver

Name: Grove I^{2}C Motor Driver V1.3

Data sheet: http://wiki.seeed.cc/Grove-I2C_Motor_Driver_V1.3/

5.3.4 Specifications

Item	Min	Typical	Max	Unit
Working Voltage	6	-	15	VDC
Max Output Current per channel	0.5			A
Maximum Total current	1.0			A
Input/output voltage on I2C bus	5			V
Communication protocol	I2C			/



5.3.5 Features

- Grove Compatible

- I2C Interface
- Adjustable motor speed and rotation direction
- Changeable slave address by hardware

Grove - I2C Motor Driver V1.3

5.3.6 Motors

Name: DC Motor Crazyflie Nano Quadcopter

Data sheet: http://www.watterott.com/de/Crazyflie-Nano-Quadcopter-6x15-mm-spare-motor-BC-CM-01-A

There are two equal motors (one left and one right on the rotating beam). Both are controlled by the Motor driver Grove I^{2}C Motor Driver V1.3.

5.3.7 2 x Motor Crazyflie Nano Quadcopter with each 1 x Propeller

Description	Spare 6x15 mm DC coreless motor for the Crazyflie Nano Quadcopter	Spare counter rotating propellers for the Crazyflie Nano Quadcopter
Specifications	• Diameter: 6 mm • Length: 15 mm • Shaft length: 3.5 mm • Shaft diameter: 0.8 mm • Weight: 1.7 g • Kv: 12000 rpm/V • Rated voltage: 4.2 V • Rated current: 810 mA • Wire length: 67 mm	• Size: 45 mm • Fits shaft: 0.8 mm

5.3.8 Attitude sensor

Name: Invensense MPU-6050

Data sheet: https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

Register Map and Descriptions: https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

6 IDE for Arduino - Atmel Studio 7.0

Authors: Daniel Kunde, Sercan Catalkaya

6.1 Contents

For the Installation on Labor PC

We need the IDE on 5 lab. computers. PC number inct_01-05

Usage of Atmel Studio in our project environment

There should be a new VM named "win10_projekt" on your Linux-Desktop which has the Atmel Studio IDE installed.

The username for the VM is stud and the password should be known.

If you need to transfer data between the VM and Linux computer, there is a shared folder that you can use: /home/stud/Win_Linux_Share

If you need to use a USB device within the project environment, you have to use the VM option "Geräte USB" and select your USB device.

About Atmel Studio 7

http://www.atmel.com/microsite/atmel-studio/

6.2 Requirements

Before installing Atmel Studio 7.0 check, if you meet the follow minimal system requirements: [Atmel Studio 2017]

Supported Operating Systems

- Windows 7, Service Pack 1
- Windows 8/8.1
- Windows 10
- Windows Server 2008 R2 Service Pack 1 or higher
- Windows Server 2012 and Windows Server 2012 R2

Supported Architectures

- 32-Bit (x86)
- 64-Bit (x64)

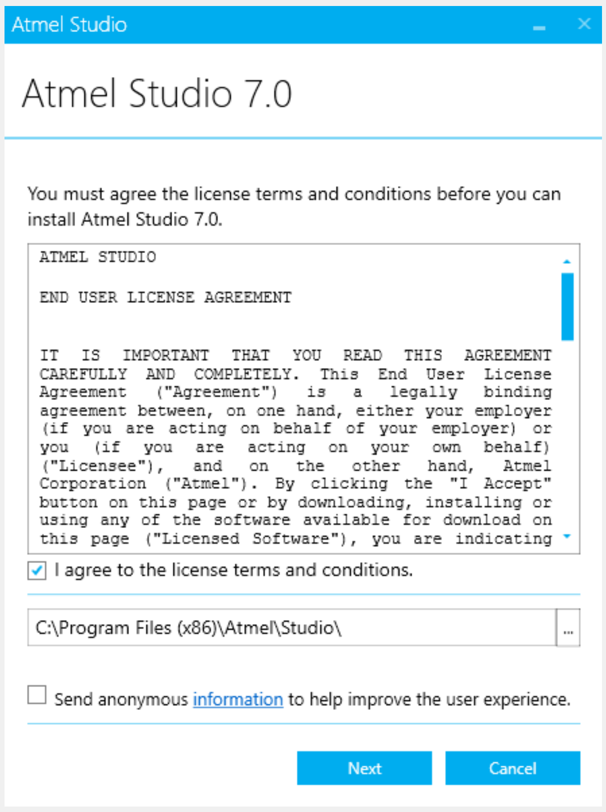
Hardware Requirements

- 1.6 GHz or faster processor
- RAM:
    - 1 GB RAM for x86
    - 2 GB RAM for x64
    - An additional 512 MB RAM if running in a Virtual Machine
- 6 GB of available hard disk space

The Arduino IDE needs to be installed as well, if we want to be able to use their libraries in Atmel Studio.

6.3 Installation Atmel Studio 7.0

Download the newest version of Atmel Studio 7 from the official website. In our example we use Atmel Studio 7.0. After the download, open the .exe file and follow the installation guide.


Atmel Studio

Atmel Studio 7.0

You must agree the license terms and conditions before you can install Atmel Studio 7.0.

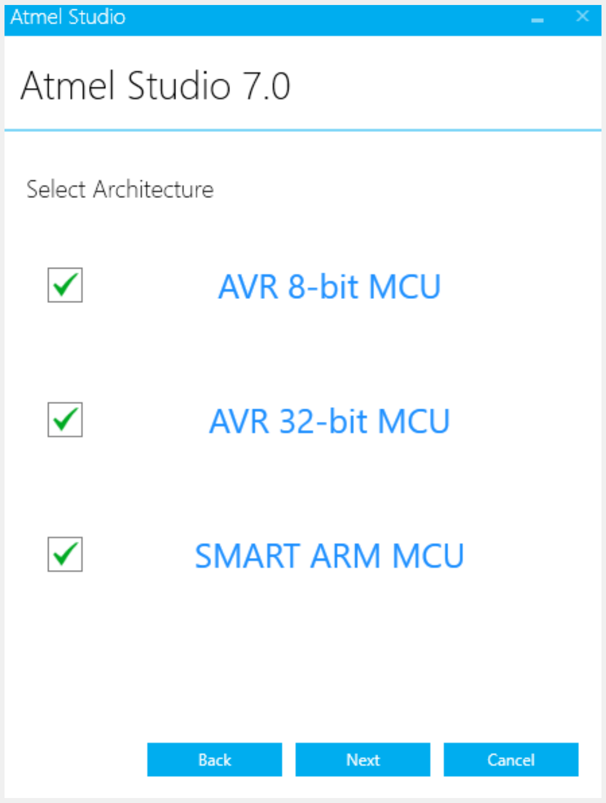
```
ATMEL STUDIO

END USER LICENSE AGREEMENT


IT IS IMPORTANT THAT YOU READ THIS AGREEMENT
CAREFULLY AND COMPLETELY. This End User License
Agreement ("Agreement") is a legally binding
agreement between, on one hand, either your employer
(if you are acting on behalf of your employer) or
you (if you are acting on your own behalf)
("Licensee"), and on the other hand, Atmel
Corporation ("Atmel"). By clicking the "I Accept"
button on this page or by downloading, installing or
using any of the software available for download on
this page ("Licensed Software"), you are indicating
```

[✓] I agree to the license terms and conditions.

C:\Program Files (x86)\Atmel\Studio\

[ ] Send anonymous information to help improve the user experience.

Read and agree to the license terms.


Atmel Studio

Atmel Studio 7.0

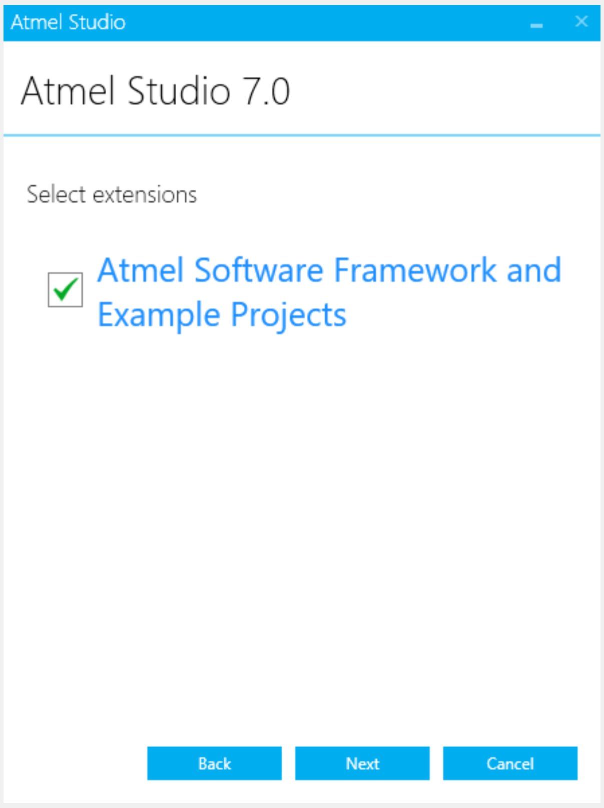
Select Architecture

[✓] AVR 8-bit MCU

[✓] AVR 32-bit MCU

[✓] SMART ARM MCU

With the next step you can select the architecture which Atmel Studio will use. There are three options. For our case we use the Arduino Mega 2560 which is an 8-bit microcontroller. So we have the option to only select "AVR 8-bit MCU".[Arduino 2017]

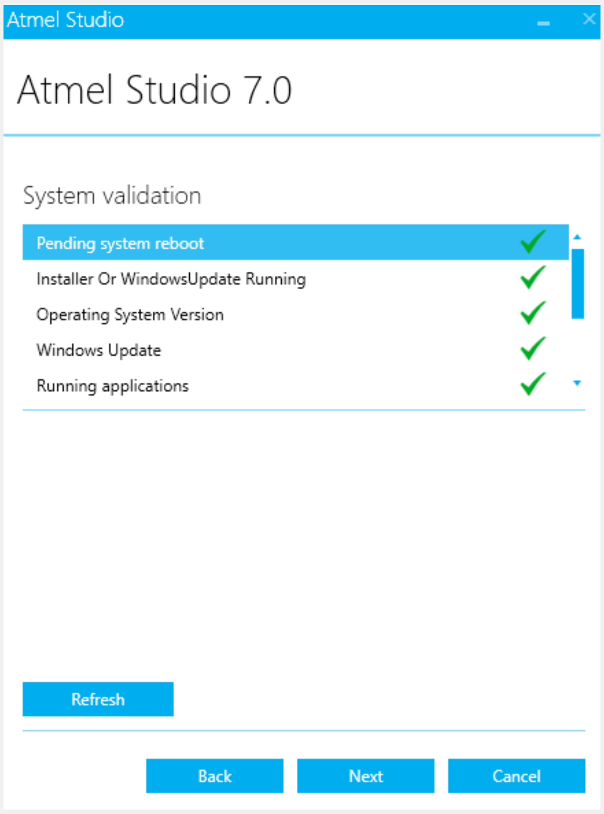

Atmel Studio

Atmel Studio 7.0

Select extensions

✓ Atmel Software Framework and Example Projects

[Optional] If you want the Atmel Software frameworks and example Project, then choose this option. The Atmel Software framework contains a software library and embedded software. [ASR 2017]

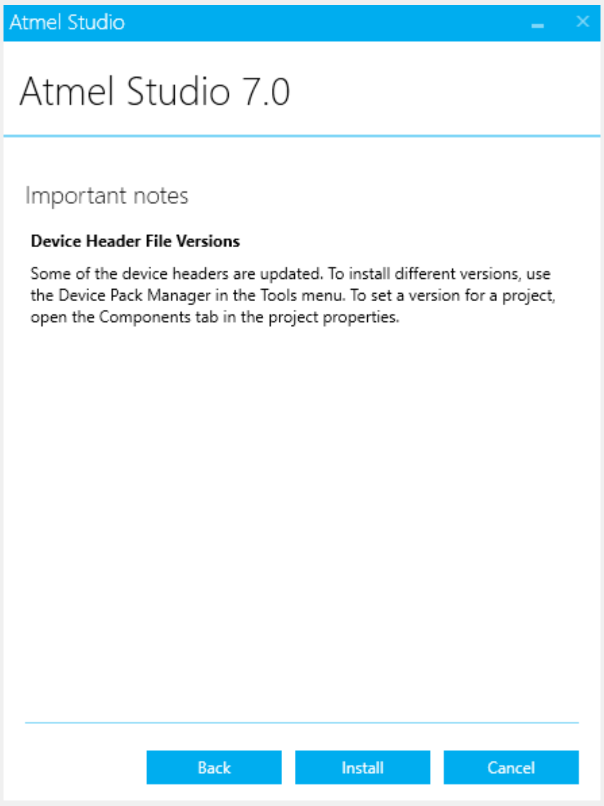

Atmel Studio

Atmel Studio 7.0

System validation

Pending system reboot	✓
Installer Or WindowsUpdate Running	✓
Operating System Version	✓
Windows Update	✓
Running applications	✓

Atmel Studio will check your system validation. All points in this table must be a green tick.


Atmel Studio

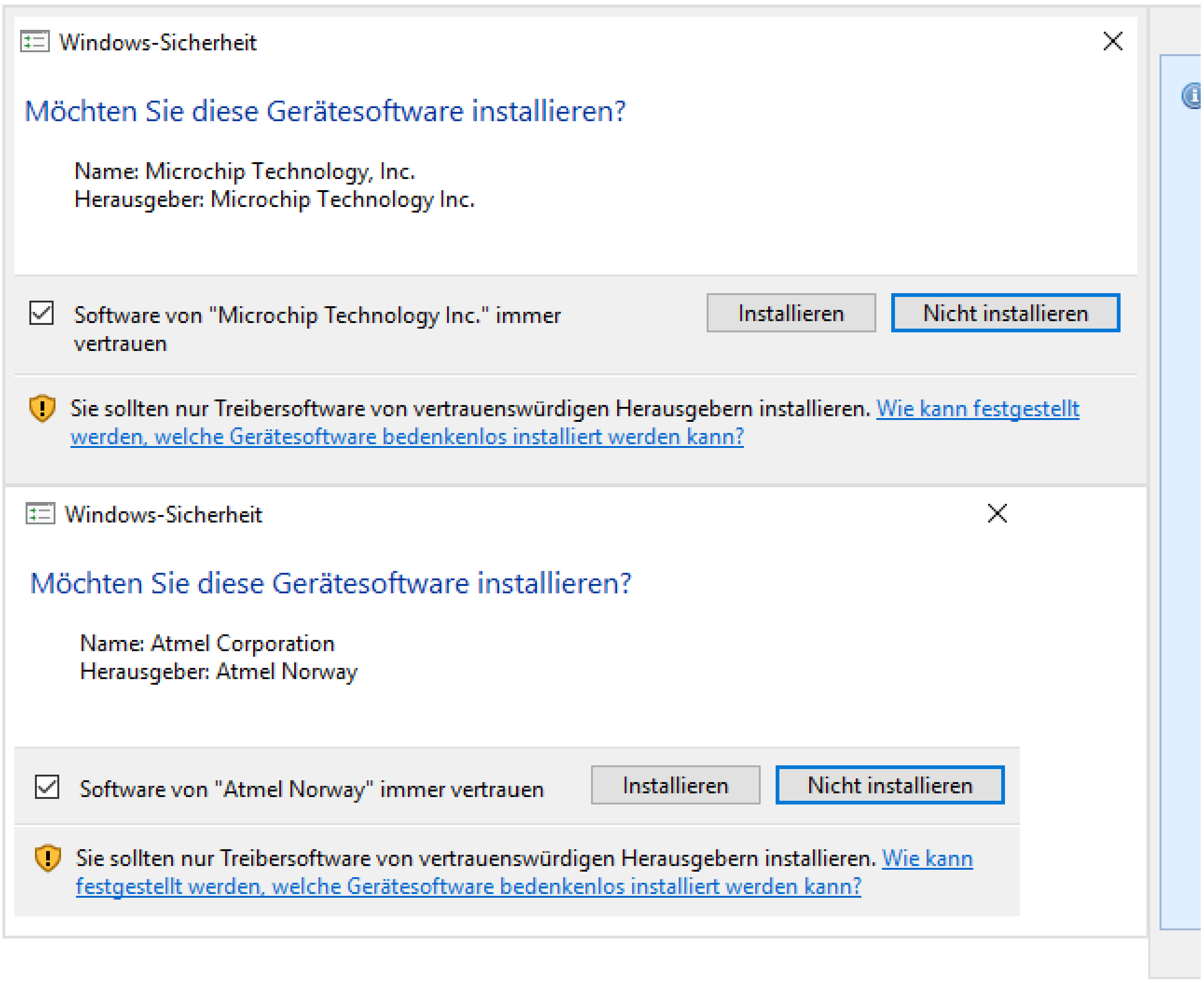
Atmel Studio 7.0

Important notes

Device Header File Versions

Some of the device headers are updated. To install different versions, use the Device Pack Manager in the Tools menu. To set a version for a project, open the Components tab in the project properties.

Install Atmel Studio by clicking on install.



Windows-Sicherheit

Möchten Sie diese Gerätesoftware installieren?

Name: Microchip Technology, Inc.
Herausgeber: Microchip Technology Inc.

☑ Software von "Microchip Technology Inc." immer vertrauen

Sie sollten nur Treibersoftware von vertrauenswürdigen Herausgebern installieren. Wie kann festgestellt werden, welche Gerätesoftware bedenkenlos installiert werden kann?

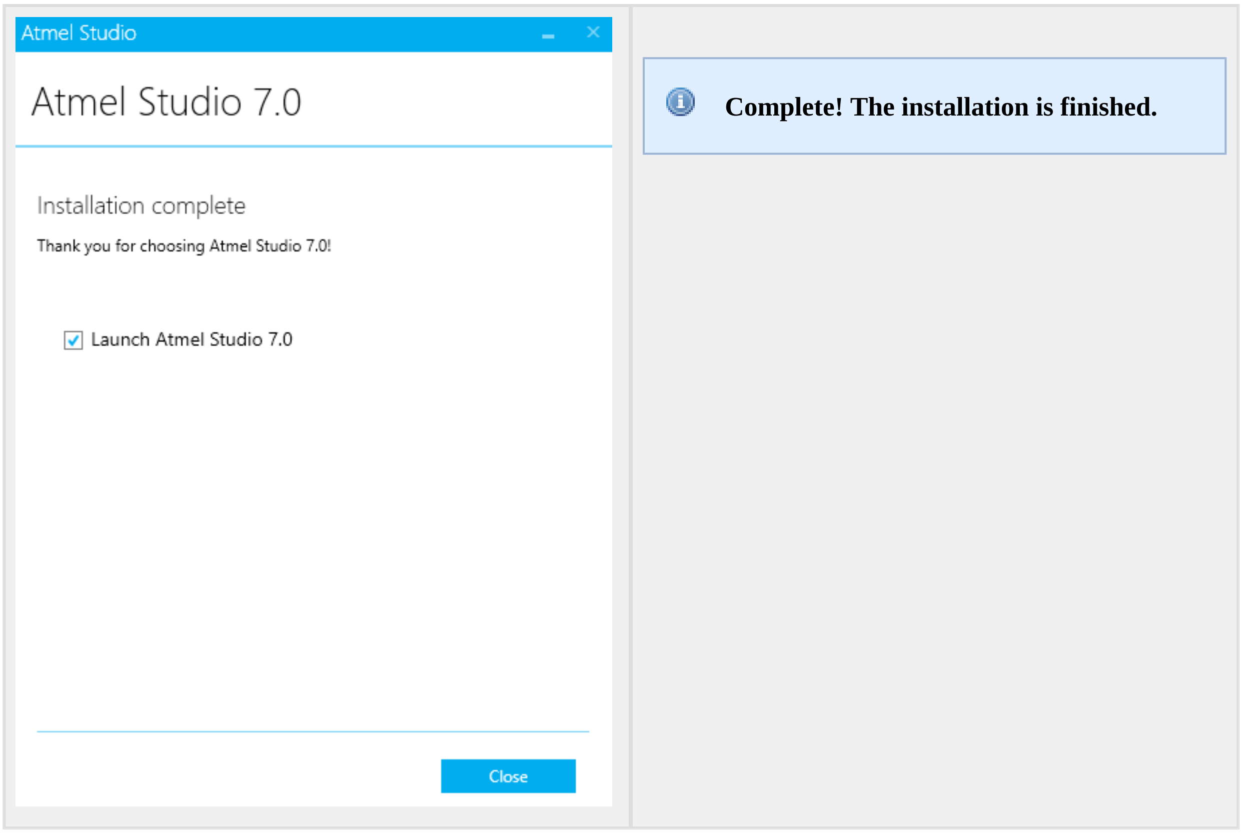
Windows-Sicherheit

Möchten Sie diese Gerätesoftware installieren?

Name: Atmel Corporation
Herausgeber: Atmel Norway

☑ Software von "Atmel Norway" immer vertrauen

Sie sollten nur Treibersoftware von vertrauenswürdigen Herausgebern installieren. Wie kann festgestellt werden, welche Gerätesoftware bedenkenlos installiert werden kann?



Installation complete

Thank you for choosing Atmel Studio 7.0!

☑ Launch Atmel Studio 7.0

Complete! The installation is finished.

6.3.1 Installation: Arduino IDE

There are two ways to install the Arduino IDE on Windows. One way is a full installation with an installer and the other is a zip file, which doesn't require admin rights. For our project, we use the zip file, version 1.8.5. Download the file from the official Arduino website with the following link:

https://www.arduino.cc/en/main/software

After you downloaded the zip file, extract the file in a directory of your choice.

6.3.2 Configuration: Atmel Studio

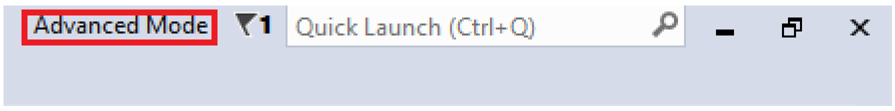
Atmel Studio provides an option to create a project from an Arduino sketch file. It is necessary to create a sketch file in Arduino, which will be used as the basis by Atmel Studio for the created project.

Creating a Project in Atmel Studio 7 for the Arduino Mega 2560

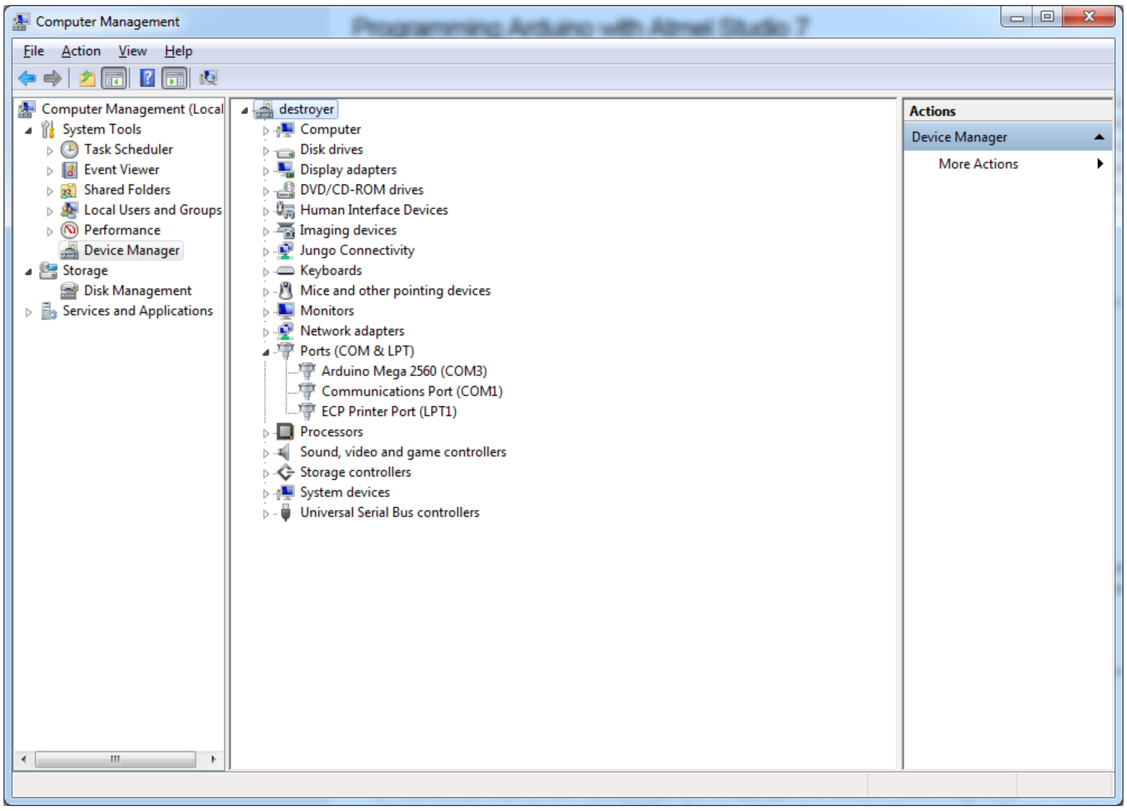
1. Click on "File -> New -> Project" or alternatively use the shortcut Ctrl+Shift+N
2. Make sure that you are on the tab "Install" and the selected language is C/C++ and click on "Create project from Arduino sketch"
3. Name your project and choose the location, confirm with "OK"
4. Select the path to the Arduino sketch file

5. Select the Arduino IDE path
6. Choose the board type “Arduino/Genuino Mega or Mega 2560”
7. Select the device “ATmega2560 (Mega 2560)”

To be able to upload the project file to the Arduino Mega 2560 microcontroller, you will have to configure an external tool in Atmel Studio. But first, you will need to make sure that the “Advanced Mode” is enabled in the IDE. The option for that is in the top right corner of the IDE.



It is also necessary to know which COM port is being used by the Arduino board. You can look it up by opening the Device Manager in Windows and then opening the tab "Ports". After plugging in the Arduino board into an USB port of the computer, the device and used COM port should show up.



- Computer Management (Local
  - System Tools
    - Task Scheduler
    - Event Viewer
    - Shared Folders
    - Local Users and Groups
    - Performance
    - Device Manager
  - Storage
    - Disk Management
  - Services and Applications

- destroyer
  - Computer
  - Disk drives
  - Display adapters
  - DVD/CD-ROM drives
  - Human Interface Devices
  - Imaging devices
  - Jungo Connectivity
  - Keyboards
  - Mice and other pointing devices
  - Monitors
  - Network adapters
  - Ports (COM & LPT)
    - Arduino Mega 2560 (COM3)
    - Communications Port (COM1)
    - ECP Printer Port (LPT1)
  - Processors
  - Sound, video and game controllers
  - Storage controllers
  - System devices
  - Universal Serial Bus controllers

[DeviceMa 2017]

Configuring an external tool in Atmel Studio 7 for the Arduino Mega 2560:

1. Click on “Tools -> External Tools…”
2. Click on “Add” and name the title “Arduino Mega 2560” (You may choose the as you will)
3. Provide the path to the avrdude.exe in the “Command:” text box. It is usually located in C:\Program Files (x86) \Arduino\hardware\tools\avr\bin\avrdude.exe
4. Put in the following parameters and values in the “Arguments:” text box: [Arduino on Atmel 2017]

Arguments	Discription
-C	Path_to_avrdude.conf – a path to avrdude.conf, you can find this file in Arduino IDE’s installation directory. Ensure you enter the path in quotes in case there are any spaces in your path.
-p	part number – AVR processor model
-c	programmer type – programmer type being used to program the board
-P	port – COM port you should have noted in stage 2
-b	baud rate – baud rate for the programmer
-D	disables auto erase for flash memory (required for wiring programmer used with Arduino Mega 2560)
-U	<memorytype>:<operation>:<path to HEX file>:<format> – this is where you should leverage Atmel Studio’s variables to always point to freshly complied and generated HEX file. List of variables and their descriptions can be found at Atmel’s help page.

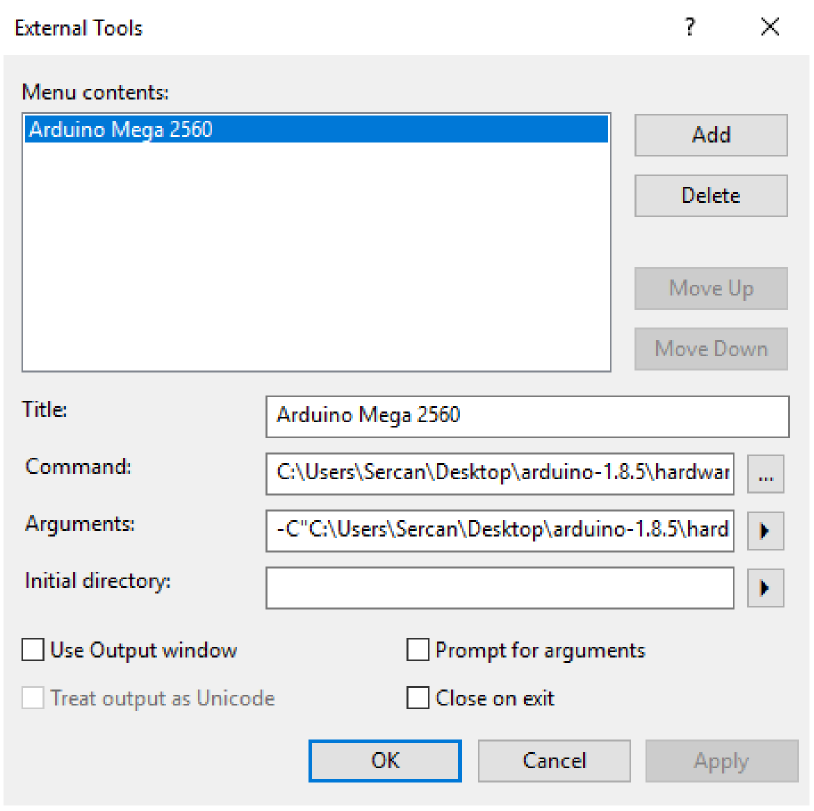
In our case, the input for the “Arguments” text box looks like this:

Depending on where the Arduino IDE has been installed, the path to the avrdude.conf might differ.

```
-C"C:\Program Files (x86)\Arduino\hardware\tools\avr\etc\avrdude.conf" -patmega2560 -cwiring -P\\.\COM3 -b115200 -D -Uflash:w:"$(ProjectDir)Debug\$(TargetName).hex":i
```

5. Leave the “Initial directory” text box blank and deselect the checkbox options below

6. To save the configuration click on apply and close the window by clicking on OK


External Tools

Menu contents:

Arduino Mega 2560

Title: Arduino Mega 2560

Command: C:\Users\Sercan\Desktop\arduino-1.8.5\hardwar

Arguments: -C"C:\Users\Sercan\Desktop\arduino-1.8.5\hard

Initial directory:

☐ Use Output window ☐ Prompt for arguments

☐ Treat output as Unicode ☐ Close on exit

Now you are able to upload your project to the Arduino Mega 2560 board by clicking on it on the tab “Tools -> Arduino Mega 2560”

Creating and testing a sketch with Atmel Studio on Arduino Mega 2560

The communication and upload between Arduino and Atmel Studio should be tested. We test it with a simple LED sketch. The Arduino contains a LED on board and can selected by pinnumber 13. This LED should blink for a duration of 1 sec low and high.

Here is the example sketch:

```
Blink Sketch

const int ledPin =  13;

void setup() {
    pinMode(ledPin, OUTPUT);
}

void loop() {
    digitalWrite(ledPin, HIGH);
     delay(1000); //ms
     digitalWrite(ledPin, LOW);
     delay(1000); //ms
}
```

To test this sketch in Atmel Studio do the following steps:

- Create a new sketch file in Arduino IDE and copy the example sketch text in our new sketch file.
- Then create a new project in Atmel Studio based on the description “Creating a Project in Atmel Studio 7 for the Arduino Mega 2560” and select the new sketch file.
- Compile your project by selecting build -> build solution or alternatively press F7
- Program the Arduino board by selecting tools -> Arduino Mega 2560

6.4 Sources

[Arduino 2017]:

Datasheet Arduino:

http://web.archive.org/web/20170711121845/http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf

[Atmel 2017]:

http://www.atmel.com/tools/atmelstudio.aspx

[ASR 2017]:

http://www.atmel.com/tools/avrsoftwareframework.aspx

[Arduino on Atmel 2017]

https://slightlyovercomplicated.com/2015/11/13/programming-arduino-with-atmel-studio-7/

[DeviceMa 2017]

https://slightlyovercomplicated.files.wordpress.com/2015/11/device_manager_com3.png

7 Time-Triggered, co-operative scheduling

Authors: Tim Niebuhr, Andre Sarich

7.1 Contents

7.2 Introduction

7.2.1 Co-operative scheduling

We will be using co-operative scheduling, which means, that each process called by the sceduler will work uninterrupted until it finishes. This causes the scheduler to run processes sequentially, only one at a time. This contrasts the method of preemptive scheduling, which allows processes to be interrupted in favor of higher-priority ones.

We prefer co-operative scheduling, since we want the system to work under real-time conditions. The validaion of these is made possible by co-operative scheduling. The main concern is the length of the processes: Every process needs to fulfill its real-time conditions, so that the CPU can handle other processes in time. For this purpose it is advantageous for every process to not use up too much time.

Extensive explanation of cooperative scheduling: http://www.safetty.net/download/pont_pttes_2001.pdf (p. 246f)

Example code of cooperative scheduling with function pointers: http://www.safetty.net/download/pont_pttes_2001.pdf (p. 256ff)

Example code of splitting tasks into simpler subtasks: http://www.safetty.net/download/pont_pttes_2001.pdf (p. 316ff)

7.2.2 Scheduling in multiprocessor systems

Clock synchronisation

We will be using co-operative scheduling on both the Arduino and the STM32. Since we have two processors, we need to synchronize them. That can be done by clock synchronisation. First we designate a "master", which sets a clock for the "slave" to use. (In our case the STM Board takes the role of the master, while the Arduino board is the slave.) The master will send periodic tick-messages, dictated by its own clock, to the slave, which uses these ticks in place of its clock.

Drafted solutions for Clock Synchronisation: http://www.safetty.net/download/pont_pttes_2001.pdf (p. 555ff)

Data transfer

Any potential data that needs to be transmitted from master to slave will be contained in the tick message, which is sent by the message dispatcher. Likewise, the slave responds to the tick-message with an acknowledge-message which contains data that needs to be transmitted back to the master.

Drafted solutions for Data Transfer: http://www.safetty.net/download/pont_pttes_2001.pdf (p. 583ff)

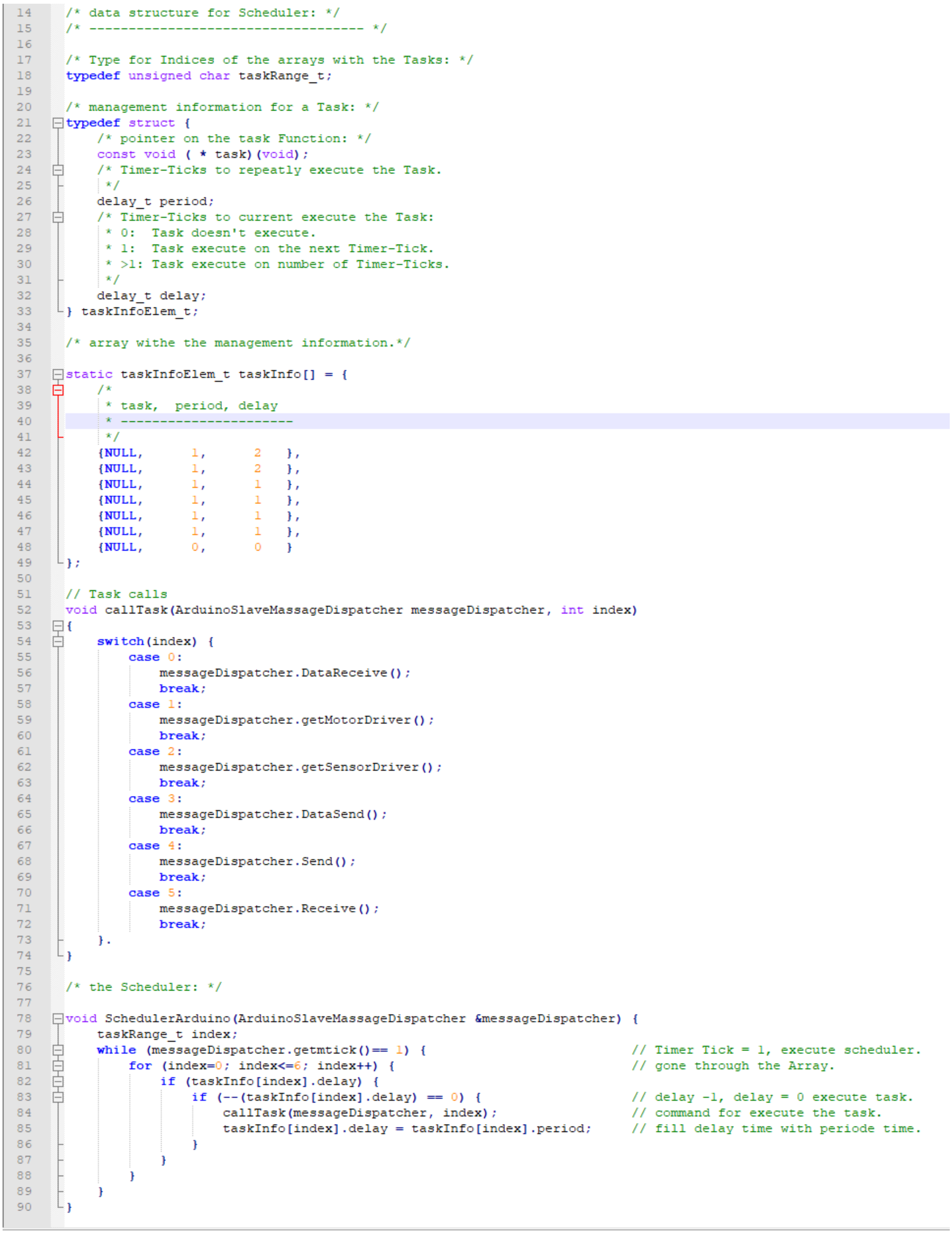
Error handling

We need error handling in case the connection between master and slave is cut or malfunctions otherwise. By measuring the time between two tick messages, the slave can detect when the ticks don't arrive in the desired timeframe. If that happens, the slave may react accordingly by going into a safe state, in our case by stopping the motors.

The master does error handling in case an acknowledge message is missing. It first stops sending tick messages (thus putting the slave into a safe state), then adresses the error by either shutting down; restarting the network by restarting itself; or engaging a backup slave.

Drafted solutions for Error Handling: http://www.safetty.net/download/pont_pttes_2001.pdf (p. 596ff)

Project code



```
/* data structure for Scheduler: */
/* ----------------------------------- */

/* Type for Indices of the arrays with the Tasks: */
typedef unsigned char taskRange_t;

/* management information for a Task: */
typedef struct {
    /* pointer on the task Function: */
    const void ( * task)(void);
    /* Timer-Ticks to repeatly execute the Task.
     */
    delay_t period;
    /* Timer-Ticks to current execute the Task:
     * 0:  Task doesn't execute.
     * 1:  Task execute on the next Timer-Tick.
     * >1: Task execute on number of Timer-Ticks.
     */
    delay_t delay;
} taskInfoElem_t;

/* array withe the management information.*/

static taskInfoElem_t taskInfo[] = {
    /*
     * task,  period, delay
     * ----------------------
     */
    {NULL,      1,      2    },
    {NULL,      1,      2    },
    {NULL,      1,      1    },
    {NULL,      1,      1    },
    {NULL,      1,      1    },
    {NULL,      1,      1    },
    {NULL,      0,      0    }
};

// Task calls
void callTask(ArduinoSlaveMassageDispatcher messageDispatcher, int index)
{
    switch(index) {
        case 0:
            messageDispatcher.DataReceive();
            break;
        case 1:
            messageDispatcher.getMotorDriver();
            break;
        case 2:
            messageDispatcher.getSensorDriver();
            break;
        case 3:
            messageDispatcher.DataSend();
            break;
        case 4:
            messageDispatcher.Send();
            break;
        case 5:
            messageDispatcher.Receive();
            break;
    }.
}

/* the Scheduler: */

void SchedulerArduino(ArduinoSlaveMassageDispatcher &messageDispatcher) {
    taskRange_t index;
    while (messageDispatcher.getmtick()== 1) {                     // Timer Tick = 1, execute scheduler.
        for (index=0; index<=6; index++) {                         // gone through the Array.
            if (taskInfo[index].delay) {
                if (--(taskInfo[index].delay) == 0) {              // delay -1, delay = 0 execute task.
                    callTask(messageDispatcher, index);            // command for execute the task.
                    taskInfo[index].delay = taskInfo[index].period;   // fill delay time with periode time.
                }
            }
        }
    }
}
```

Sources:

http://www.safetty.net/download/pont_pttes_2001.pdf, Chapter 14-16, 25-27

https://www.safetty.net/download/pont_eres2_2016_extract.pdf, Chapter 2, 9, 12

Code examples can be found at https://www.safetty.net/publications/the-engineering-of-reliable-embedded-systems-second-edition/ttrds

8 Conclusions at the End of Winter Term 2017/18

Author: Prof. Dr. Jan Bredereke

This project achieved its main goals. However, due to lack of time, we could not complete the actual implementation.

The main teaching goal for the students was:

- How to construct a hard real-time and distributed embedded system by using a distributed time-triggered architecture.

The main goals of the project organizers were:

- To gain practical experience with the real-time operating system **FreeRTOS**, and
- to gain practical experience with the micro-controller **STM32**

for a time-triggered architecture.

A secondary goal of the project organizers was:

- To gain experience on how to use **Custom Off-The Shelf Drivers** in a time-triggered architecture.

We summarize our experiences with regard to these aspects in the following subsections. Furthermore, we describe what is still missing, and what is necessary to complete the implementation.

8.1 Using FreeRTOS for a Time-Triggered Architecture

The open-source real-time operating system FreeRTOS turned out to be well suited for use in a time-triggered architecture. Its documentation firstly describes a use in an event-triggered architecture. But the concepts and features necessary for a time-triggered architecture are provided and documented as well.

Concerning the documentation support for time-triggered versus event-triggered systems, FreeRTOS exeeds the commercial real-time operating system QNX. We investigated QNX in last year's project [Bre+17]. There, it turned out that designing a time-triggered architecture with QNX appears to be very well feasible. But you have to know well the concepts you want to apply. Otherwise, the QNX documentation can be misleading easily.

The tool chain worked well for us. We used the System Workbench for STM32. It is a free, multi-OS software development environment based on Eclipse. Furthermore, we used the graphical configuration tool STM32CubeMX. It is available without cost for all major desktop operating systems. It provided considerable help to configure a software project for our microcontroller board.

8.2 Using an STM32 for a Time-Triggered Architecture

Using a microcontroller requires a suitable Board Support Package (BSP) containing the nessary drivers and interface line definitions. The BSP is necessary for accessing and making use of all the hardware components of the microcontroller.

In the current project, this went entirely smooth for the STM32 microcontroller with the FreeRTOS operating system. There was a current and mature BSP available.

This was a better experience than in last year's project [Bre+17]. There, we attempted to use Airbus's e.Cube computer which is suitable for space. We did this with QNX. However, it turned out that the most current BSP by the board manufacturer was for QNX version 6.4.1, while at that time QNX already was at version 6.6.0. There did not appear to be hope for a more current BSP by the manufacturer. We therefore switched from the e.Cube to a Beagle Bone Black microcontroller. But the BSP for the BeagleBone Black did not have the quality we expected, either.

8.3 Using Custom Off-The Shelf Drivers in a Time-Triggered Architecture

In the project of the previous year [Bre+17], we found that often it is not possible to use a Custom Off-The-Shelf (COTS) driver in a time-triggered architecture. All the COTS driver libraries we wanted to use then were based on interrupts. Therefore they were based on the event-triggered paradigm. This kind of interrupts does not integrate into the scheduling scheme of a time-triggered system. Furthermore, the time-triggered, cooperative approach demands that each task must yield the processor after a a fixed and, in particular, after a short period of time. If necessary, the task must be organized such that it continues its work in the next time slot. None of the COTS drivers used was designed in this way. This required us to redesign them from scratch.

While preparing for the current project, we specifically looked for COTS drivers providing a mode of operation based on timeouts instead on interrupts. And indeed, we found suitable low-level drivers for all relevant interfaces. We had to rewrite the high-level drivers which are using these low-level drivers. But this was no problem because of the simplicity of the functionality of the high-level drivers used here.

To summarize, we were able to solve the problem of COTS drivers suitable for a time-triggered architecture by looking early in the project for such drivers.

8.4 Incomplete Implementation of the Angle Control Demonstrator

The students did not complete the implementation of the Angle Control Demonstrator. The main reason for this concerned the self-organization while working on the project. The work in small groups mostly went well. But the definition and the communication of the interfaces went less well. Similarly, the students learned that they should have felt more responsible for planning and testing the integration of the system.

The teacher took the risk for this deliberately. He could have provided more guidance, but this would have prevented the students from making their own, valuable experience.

8.5 Outlook

This project report contains a wealth of information about the individual components. Therefore, the next steps are not about finding suitable components or about writing entirely new code. Instead, the components need to be fitted together in the right way. In order to continue work on the Angle Control Demonstrator, first one will need:

- precise specifications of the interfaces between the components, suitable for the time-triggered paradigm
- a plan on how to validate that the components meet their specifications, including at least a rough estimate of their temporal properties
- a plan on how to integrate the components, and on how to validate that the integrated system meets its requirements
- a process to ensure that the plans make progress as intended, in order to enable corrective measures where necessary

After all of this has been established, the (mostly) existing code can be re-cast into the new casting moulds of the now matching interfaces.

8.6 References

[Bre+17] Jan Bredereke, Julian Greilich, Benjamin Hesseln, Jan Lehrke, Nils Müller, Jonas Pufahl, Jens Sager, Markus Salomon, Benjamin Schäfer, Tobias Schmitz, Maximilian Schönenberg, Nikolas Schreck, Peter Tschubij, Mirco Wittrien, and Rico Thiele (Feb. 2017): A Time-Triggered Architecture For an Attitude Control System With Space Technology. http://homepages.hs-bremen.de/~jbredereke/downloads/wp_embeds-projektbericht-time-triggered-2017.pdf (last visited 2 Mar. 2018).

9 Appendix: Structure of the oral presentation

- 1. Einführung/Aufgabenstellung (Prof. Jan Bredereke)
- 2. Winkelsteuerungsmodell (Henrik Giessel, Kai Nortmann)
- 3. PID-Regelung (Marvin Pöperny, Olga Tschernobai, Hans Martin Pfennig)
- 4. Zeitgesteuertes Scheduling (Tim Nibuhr, André Sarich)
    - 4.1. Allgemein
    - 4.2. Verteilte Systeme
    - 4.3. Unser System und Protokoll
- 5. Arduino und zeitgesteuerte Treiber (Christian Zöller, Eike Diekmann, Hermann Wafo)
- 6. freeRTOS (Rene Wanzow, Arthur Guz)
    - 6.1. Was ist das? Was stellt es zur Verfügung?
    - 6.2. Warum nutzt man es?
    - 6.3. Praktische Erfahrungen (Installation, BSP, Dokumentation)
    - 6.4. RTOS und zeitgesteuerte Verarbeitung + Treiber
- 7. Zusammenfassung und Ausblick (Daniel Kunde, Sercan Catalkaya)

Präsentation:

https://docs.google.com/presentation/d/1TkSozreX1WnYvx_sCWPMF-C5MHaXit3JYSIygCr_Sjs/edit#slide=id.p