Smart Mechatronics

Hochschule Bochum
Bochum University
of Applied Sciences

Master Thesis

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science (M.Sc.) in
Electrical engineering

Design and implementation of a real-time embedded system for a heterogeneous multiprocessor platform

Autor: Philip Geuchen
philip.geuchen@stud.hs-bochum.de
Matriculation number : 016200549

Examiner: Prof. Dr.-Ing. Arno Bergmann
Supervisor: Dipl.-Math. Kai Morwinski

Submission date: 01.11.2021

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur Abschlussarbeit:
"Design and implementation of a real-time embedded system for a heterogeneous multiprocessor platform"

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Unterschrift: *Ort, Datum*:

Contents

List of Abbreviations

ADC	Analog Digital Converter
AHB	Advanced High-performance Bus
API	Application Programming Interface
ASAM	Association for Standardization of Automation and Measuring Systems
ASCII	American Standard Code for Information Interchange
CDT	C/C++ Development Toolkit
CONSENS	CONceptual design Specification technique for the ENgineering of complex Systems
CPU	Central Processing Unit
CTO	Command Transfer Object
DAC	Digital-to-Analog Converters
DAG	Directed Acyclic Graphs
DAQ	Data Acquisition
DC	Direct Current
DDR	Double Data Rate
DLL	Dynamic Link Library
DM	Deadline Monotonic
DMA	Direct Memory Access

DTO Data Transfer Object

ECU Electronic Control Unit

EDD Earliest Due Date

EXTI Extended interrupts and events controller

GIMP GNU Image Manipulation Program

GNU GNU s Not UNIX

GPIO General Purpose Input/Output

GPU Graphics Processing Unit

GTK GIMP-Toolkit

HAL Hardware Abstraction Layer

HSEM Hardware Semaphore

I2C Inter-Integrated Circuit

IPCC Inter-Processor Communication Controller

IPC Inter-Processor Communication

IRQ Interrupt Request

ISA Instruction Set Architecture

ISR Interrupt Service Routine

IWDG Independent WatchDoG

I/O Input/Output

LDF Latest Deadline First

LVGL Light Versantil Graphics Library

MCU MicroController Unit

MMU	Memory Management Unit
MPSoC	MultiProcessor System-on-Chip
MPU	Microprocessor Unit
MSVC	Microsoft Visual C++
NPU	Neural Processing Unit
NVIC	Nested Vectored Interrupt Controllers
ODT	Object Descriptor Table
OpenAMP	Open Asymmetric Multi Processing
OS	Operating System
PC	Personal Computer
PCB	Printed Circuit Board
PIL	Process In the LoopPIL
PWM	Pulse Width Modulation
RAM	Random-Access Memory
RCP	Rapid Control Prototyping
RM	Rate Monotonic
RPMsg	Remote Processor Messaging
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
STF	System Target File
USBH	Universal Serial Bus Host
TCP/IP	Transmission Control Protocol/Internet Protocol

TDL Task Descriptor List

TLC Target Language Compiler

TT Time Triggered

WCET Worst-Case Execution Time

WLAN Wireless Local Area Network

USART Universal Synchronous/Asynchronous Receiver Transmitter

USB Universal Serial Bus

XCP Universal Measurement and Calibration Protocol

Symbol Directory

Symbol	Meaning	Unit
$\underline{A}, \underline{A}_d$	Continuous, discrete system matrix	
a	Acceleration	$\mathrm{m\,s^{-2}}$
$\underline{B}, \underline{B}_d$	Continuous, discrete input matrix	
bit	Bit	bit
$\underline{C}$	Output matrix	
C_i	Worst case execution time	s
$\underline{D}$	Feedthrough matrix	
D_i	Relative deadline of J_i	
d_i	Absolute deadline of J_i	s
E	Expected value	
F	Force	$\mathrm{kg\,m\,s^{-2}}$
f	Frequency	Hz
f_i	Actual end time of J_i	
$\underline{G}, \underline{G}_d$	Continuous, discrete system noise matrix	
G_x	Transfer function of x	
h	Position	m
$\underline{I}$	Identity matrix	
J	A set of job	
J_i	A job	
J_x	Mass moment	$\mathrm{kg\ m^2}$
$\underline{K}$	Matrix of Kalman gain	
k	Index ($t = k \cdot T_s$)	
l_i	Laxity of J_i	s
m	Mass	kg
n	Number	

Symbol	Meaning	Unit
$\underline{\hat{P}}$	Covariance matrix of the estimation error	
$\underline{Q}$	Covariance matrix of the system noise	
R	Resistance	Ω
$\underline{R}$	Covariance matrix of the measurement noise	
r_i	Release time of J_i	s
$\underline{S}_B, \underline{S}_B^*$	Continuous, discrete observability matrix	
s_i	Actual start time of J_i	s
T	Sample time	s
t	Time	s
T_i	Period / Interval length	s
T_{idle}	Relative idle time of a core	
$\underline{u}$	Input vector	
u_i	Task utilization	
U_c	Utilization of a core	
U_{max}	Maximum utilization of τ	
U_{sum}	Total utilization of τ	
V	Voltage	V
v	Velocity	$\mathrm{m\,s^{-1}}$
$\underline{x}$	State vector	
$\underline{\hat{x}}(t)$	Predicted state vector	
$\underline{\tilde{x}}(t)$	Corrected state vector	
$\underline{y}$	Output vector	
$\underline{z}$	System/process noise vector	
$\Delta\underline{y}$	Difference of the output vector	
$\underline{\varepsilon}$	Estimation error	
θ_x	Robot tilt angle	rad
σ^2	Variance	
τ	A task system	
τ_i	A task	

1 Introduction

MultiProcessor System-on-Chip (MPSoC) are used more and more frequently in sectors of industry, signal processing, and embedded systems due to their multiple usabilities and increasing performance. [1]

Heterogeneous processor architectures offer promising opportunities in current and future safety-critical innovations where real-time [2] is required. [3]
Heterogeneous MPSoC platforms offer the possibility to execute different tasks with mixed criticalities in one MPSoC simultaneously. [4]

The implementation of a heterogeneous MPSoC in a cyber-physical system opens the possibility to distribute its tasks between different independent computing cores. [3]

In this master thesis, the software for a heterogeneous MPSoC is designed and implemented, which performs real-time bound tasks on a real-time capable core, while a non-real-time capable core performs tasks that require an Operating System (OS). The control of an inverted pendulum represents the tasks, that are executed on the real-time capable core. The execution of a graphical application and network services represent the tasks performed by the core running an OS.

The main focus lays on the design and implementation of real-time capable software components using model-based design. In the preceding bachelor thesis [5], the feasibility of implementing software on the separate computing cores of the STM32MP1 MPSoC has been demonstrated. A previous development project [6] adapted an existing model-based software development tool designed for microcontrollers [7], based on MATLAB Simulink [8], to the real-time capable core of the MPSoC.

1.1 Objectives

The objective of this work is to develop real-time capable software for the heterogeneous multiprocessor platform STM32MP1. It is required, that the software for the real-time capable core of the platform is developed by model-based design using MATLAB Simulink, see item A.1.1. Other objectives are partial platform support required for the model-based implementation and the use of the real-time capable core as a processor in the loop with the external mode via Universal Measurement and Calibration Protocol (XCP). The XCP messages have to be forwarded via the core running the OS to enable a Transmission Control Protocol/Internet Protocol (TCP/IP) connection to the development computer. The heterogeneous platform is implemented in the inverted pendulum to demonstrate that the real-time capable core of the platform can meet the real-time critical boundaries. Figure 1.1 shows the implementation of the STM32MP1 in the inverted pendulum, which is called self-balancing robot in the further course of the thesis.

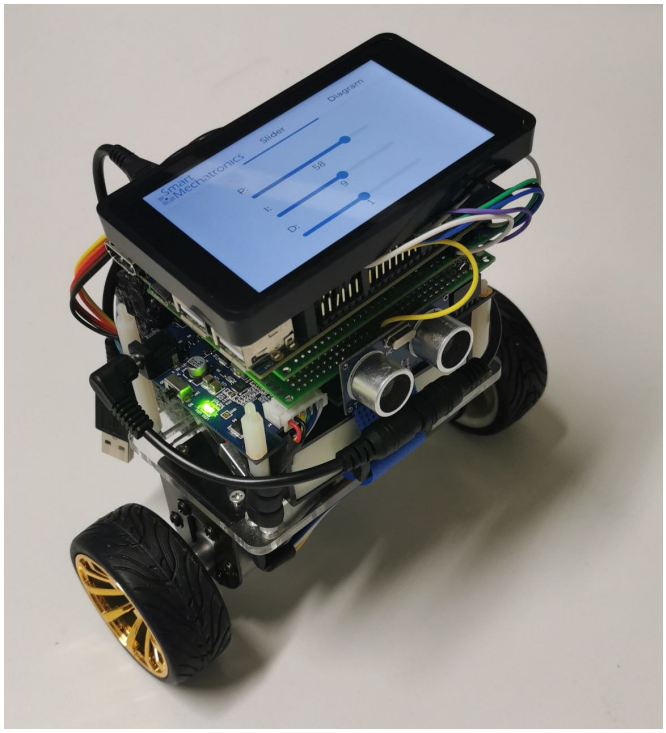


Figure 1.1: Self-balancing robot controlled by the STM32MP1

2 Foundations/Theory

This chapter explains definitions and terms that should help to understand the project. Readers who are familiar with the topics heterogeneous multiprocessors, code generation by Simulink, and scheduling can read on at chapter 3.

2.1 Heterogeneous multiprocessor platforms

Typically, MPSoCs have multiple sets of identical cores, called clusters, and can have programmable logic tiles, such as Graphics Processing Unit (GPU)s or Neural Processing Unit (NPU)s. Typical MPSoCs have a hypervisor, that allows the operating system to perform global scheduling of tasks. Heterogeneous MPSoCs have several different cores that have different Instruction Set Architecture (ISA)s. Due to the different ISAs, it is not possible to perform global scheduling by a hypervisor. Commonly, cores of heterogeneous MPSoCs execute different operating systems. The different ISAs require a separate compilation of the code to be executed by the heterogeneous cores. [3]

2.2 Introducing the STM32MP157C-DK2

The STM32MP157C-DK2 MPSoC from STMicroelectronics [9] consists of an Arm-based dual Cortex-A7 multicore cluster clocked at 650 MHz. This multicore cluster is abbreviated by Cortex-A7 in the following. The Cortex-A7 has a Memory Management Unit (MMU), which enables memory virtualization. A multi-purpose Linux operating system can be hosted on the Cortex-A7. [10]
The second core of the STM32MP157C-DK2 MPSoC is a single Arm-based Cortex-M4 processor, which is clocked at 209 MHz. This core is abbreviated by Cortex-M4 in the following. The Cortex-M4 is a MMU-less core [11], and can only handle lightweight operating systems such as FreeRTOS [12] or Micrium's RTOS [13]. [10]

The Cortex-A7 and Cortex-M4 operate independently but share peripherals such as General Purpose Input/Output (GPIO)s, Inter-Processor Communication Controller (IPCC), Static Random Access Memory (SRAM) and Hardware Semaphore (HSEM). Peripherals like Serial Peripheral Interface (SPI), Analog Digital Converter (ADC)s, Timers, or Direct Memory Access (DMA)s must be assigned to one of the processor cores. [14]

2.3 Application Mapping and Scheduling Problems

Mapping is a configuration and simplification method that includes all the way to implementation. Various analysis methods for the exploration of the design space for performance can be mapped considering system platform Application Programming Interface (API) descriptions of the services contained on the platform. [15]

A crucial design step is the mapping of applications to the available hardware platforms. This involves mapping the applications to execution times and processors. If as many scheduling decisions as possible are made in the design period, it is possible to provide a time constraint guarantee. The selection of the scheduling algorithm is about using a system with a combination of specific applications. [16]

In the application planned in this thesis, it is expected that the robot, while communicating with the host computer, will continue to read the data from the accelerometer and gyro sensor to calculate how to control the motor to maintain the balance.

Many modern embedded and cyber-physical systems are built on existing hardware platforms because the goal is to find the right combination of hardware and software to create a product that meets all specifications as efficiently as possible. This design method is called hardware/software codesign. A system built on a hardware platform is not designed through a synthesis process derived from the behavioral specification. Another reason for the limitations that lead to the reuse of hardware, as well as software is the increasing complexity and stringent requirements in time-to-market. The use of existing hardware platforms leads to the term platform-based design. [16]

In this context, a platform is described as a family of architectures that fulfill constraints to enable the reuse of hardware and software components. Thereby platforms represent abstraction layers to cover simplification in low levels. The use of the general

reuse technology has the goal of reducing development costs and times. The combination of hardware and software platforms leads to the systemplatform approach. [15]

When mapping applications to execution platforms by using platform-based design, there are also different design options. For example, a decision can be made between variants of a platform with different speeds or a different number of processors, or different communication architecture. [16]

The mapping problem is defined as follows [17]:
Given:

- a number of applications
- application use-cases
- a number of available architectures

Find:

- the applications are mapped to the processors
- selection of an appropriate scheduling technique (if not defined)
- selection of a target architecture (if not defined)

Objectives:

- deadline compliance and/or performance maximization
- cost and energy consumption minimization and perhaps other objectives

To be able to deal with the scheduling problem in more detail, some symbols and definitions have to be explained previously. In this thesis, the following definitions are taken from [16]:
Definitions:

- Every task τ_i execution is called a job J. This implies that for a task τ_i there is an associated set of jobs $J(\tau_i)$. The set of jobs of a task may not be finite due to the possibility of repetitions.
- Tasks are called periodic if they are released once in a time unit T_i, where T_i is the period of the task.
- If there is a lower bound on the length of the interval between adjacent releases of a task, the task is called sporadic, where the interval length is also referred to as T_i.
- Aperiodic tasks are tasks that are not periodic and not sporadic. In task systems consisting only of periodic and sporadic tasks, the concept of hyper-periods can

simplify scheduling considerably. In this case, the interval length is also referred to as T_i.

- The hyper-period of a periodic or sporadic task system τ is defined as the least common multiple of the periods of the individual tasks.

Symbols[16]:

- a set of tasks $\tau = \{\tau_1, \cdots, \tau_n\}$
- a set of jobs $J = \{J_i\}$
- the release time r_i of J_i (at the time the execution becomes available)
- the Worst-Case Execution Time (WCET) C_i
- the absolute deadline d_i related to J_i
- the relative deadline D_i, the time between the availability of a job J_i to the time when the job must be finished $(D_i = d_i - r_i)$
- the laxity or slack l_i. In the case that $l_i = 0$, J_i is started immediately after the release $l_i = D_i - C_i$
- the actual start time s_i related to J_i
- the actual end time f_i related to J_i

Entirely Time Triggered (TT) systems

Entirely TT systems are systems in which a dispatcher processes a Task Descriptor List (TDL) planned during the design process. The task of the dispatcher is to process the TDL. The dispatcher does not make any decisions himself. In such systems, the dispatcher can be controlled by a timer. [16]

In entirely TT systems, a temporal control structure for all tasks is defined in a planning process. The scheduling that takes place in prior time, which takes into account the required priority and completion times between the tasks, eliminates the need for explicit coordination by an operating system at runtime. [2]

In hard real-time systems, predictability, by satisfying timing constraints on system behavior, is the most important concern. To ensure predictability in a complex system, pre-run-time scheduling is often the only practical option. [18]

The main disadvantage of TT systems is that this response to events can be quite poor. [16]

In this thesis, the scheduling of periodic and aperiodic tasks on a Cortex-M4 has to

be planned. Since all tasks are enabled by interrupts, it is not possible to use scheduling methods that require a dynamic priority assignment. Therefore, scheduling methods with dynamic priority assignments such as the Earliest Deadline First (EDF) [16] algorithm or the Least Laxity (LL) [16] algorithm are not considered here. Hence, scheduling algorithms based on static priority assignments are considered here.

Earliest Due Date (EDD) Algorithm

If a situation is considered where all jobs arrive at the same time, and the lateness is to be minimized, preemption of the jobs becomes unnecessary. In this situation, a rule established by Jackson in 1955 states that given a set of independent jobs with deadlines, any algorithm that executes the jobs in the order of nondecreasing deadlines, operates optionally to minimize the maximum lateness. Such an algorithm is called an EDD. Such algorithms can be statically scheduled if the deadlines are known in advance. The complexity of the EDD algorithm is $\mathcal{O}(n \log n)$. [19]

Scheduling Without Preemption

Scheduling algorithms without preemption require processor idle times to complete jobs with earlier deadlines that arrive at a later time. Such are called clairvoyants because they require knowledge about the future. An algorithm that keeps the processor idle even though it has jobs available is not called work conserving. [16]

Scheduling with Precedence Constraints

Priority rules can be mapped by Directed Acyclic Graphs (DAG)s G. The following applies [16]:

- $G = (\tau, E)$
- $E \subseteq \tau \times \tau$
- $E :=$ edges
- $\tau :=$ vertices (or nodes)

Figure 2.1 shows such a DAG. The vertex of an instance represents a task, and the edges correspond to the dependencies of the task. In multiprocessor systems, a DAG

can also be used to separate tasks into subtasks and to distribute them to different processors. Then, the vertices would correspond to the individual subtasks. [16]

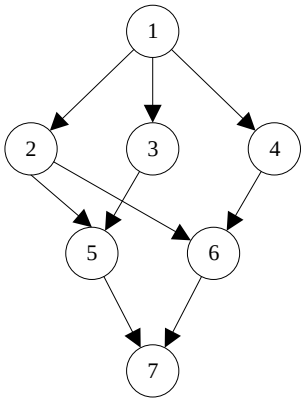


Figure 2.1: DAG example (cf. [16, p. 310])

The example in Figure 2.1 shows 7 tasks. The tasks are each assigned to a node. The edges express the order in which the tasks must be processed.

Latest Deadline First (LDF) Algorithm

In the case of simultaneous arrival times of dependent jobs, the LDF algorithm can lead to an optimal minimization of the maximum delay. The LDF starts listing the tasks with the largest dead time into a queue. In doing so, the LDF starts in the DAG at the bottom row with the tasks that do not have a successor. During runtime, this queue is processed from back to front. If the jobs occur asynchronously, a modified LDF algorithm can be selected. [16]
In the development of periodic scheduling algorithms, there are different goals than for aperiodic scheduling algorithms. Finding the minimum total length of a schedule, for example, is not an issue when dealing with tasks with infinite repetition. Periodic schedulers are considered optimal when they find a feasible schedule if one exists. For periodic, as well as sporadic task systems, a task utilization u_i can be defined according to [16, p. 312]:

$$u_i = \frac{C_i}{T_i} \tag{2.1}$$

Even for periodic tasks, T_i is the period and for sporadic tasks, T_i is the minimum separation of tasks, the task systems are treated with the same definition of task utilization. [16]

According to [16, p. 312], the maximum utilization U_{max} and the total utilization U_{sum} for task systems $\tau = \{\tau_1, \cdots, \tau_n\}$ are defined as follows:

$$U_{max} = \max_i(u_i) \tag{2.2a}$$

$$U_{sum} = \sum_i u_i \tag{2.2b}$$

Rate Monotonic (RM) Scheduling

Probably the best-known scheduling algorithm for independent periodic tasks is Rate Monotonic Scheduling. [16] It requires the following RM assumptions [20, p. 2]:

- All tasks that require hard deadlines occur periodically and have a constant interval between their occurrence
- All deadlines must be run-time constraints (each task must complete before it can be invoked again)
- All tasks are independent of each other (the invocation of one task is not related to the initialization or processing of another task)
- The runtime of each task is constant and does not change over time
- Non-periodic tasks in the system are special. They are used for initialization or troubleshooting and do not have hard real-time constraints themselves.

Another assumption made for this type of schedule is that the context switching is negligible. In mathematical notation, the assumptions are that $D_i = T_i$ and that C_i is constant and known for each task.

According to [20, p. 5-6] it follows that for a single processor and n tasks the accumulated utilization U_{sum} is not allowed to be exceeded:

$$U_{sum} = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1) \tag{2.3}$$

This leads to a maximum U_{sum} value of 0.7 for large n according to [20, p. 8]:

$$\lim_{n \to \infty} n(2^{1/n} - 1) = \log_e(2) = \ln(2) \approx 0.7 \tag{2.4}$$

In the case of monotonic scheduling, the priorities of tasks are a monotonically decreasing function of the period. This means that tasks with short periods are given high priority, and tasks with long periods are given low priority. The RM scheduling strategy works through fixed preemptive priorities. [16]

Figure 2.2 shows an example of RM scheduling for 6 periodic tasks. The tasks are numbered from τ_1 to τ_6. The double arrows in the respective task timeline symbolize the arrival time of a task and the deadline of the previous task. The tasks are sorted by the duration of the period. Task τ_1 has the shortest period and is therefore assigned the highest priority. Task τ_6 has the longest period and is therefore assigned the lowest priority. In Table 2.1, the worst-case execution time C_i, the period duration T_i, and the task utilization u_i are shown for the individual tasks.

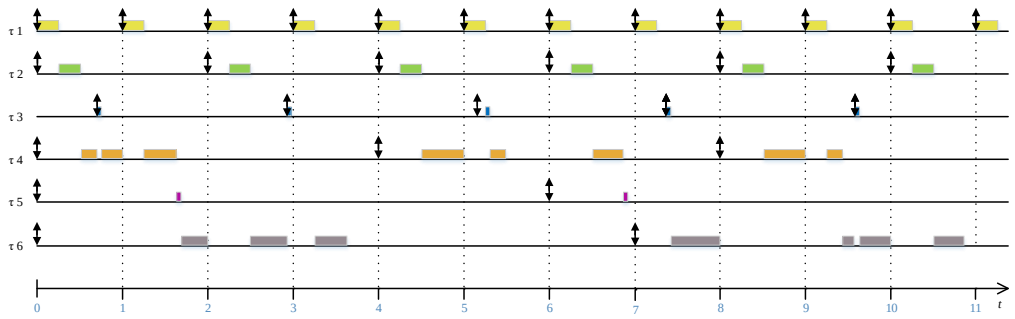


Figure 2.2: Example of a schedule generated using an RM scheduler

i	C_i	T_i	$u_i = C_i/T_i$
1	0.25	1	250 m
2	0.25	2	125 m
3	0.05	2.2	22.7 m
4	0.6	4	150 m
5	0.05	6	8.33 m
6	1.2	7	157 m

Table 2.1: Example RM scheduling task table

Verification can then be made to see if the accumulated utilization U_{sum} is less than $6(2^{1/6}-1)$.

$$U_{sum} = \sum_{i=1}^{n} \frac{C_i}{T_i} \approx 0.713 \leq 6(2^{1/6}-1) \approx 0.734 \tag{2.5}$$

As visible in equation (2.5), U_{sum} is less than $6(2^{1/6}-1)$. This means that enough idle time is available to guarantee schedulability for RM scheduling in this example.

Deadline Monotonic (DM) Scheduling

For tasks, whose deadline does not match the period duration, an extended RM scheduling can be applied, which is called DM. DM scheduling can handle tasks with explicit deadlines. Like RM, DM is based on static task priority. This is determined by considering the relative deadline D_i. If $D_i < Di'$, task τ_i is assigned the higher priority. For tasks that have explicit deadlines, equation (2.3) can be transformed into equation (2.6). [16]

$$U_{sum} = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{1/n}-1) \tag{2.6}$$

Scheduling periodic tasks with precedence constraints

Scheduling interdependent tasks is more complex than scheduling tasks that are performed independently. There are ways to reduce the scheduling effort [16]:

- Adding extra resources to simplify scheduling.
- Splitting tasks into dynamic and static, to make as many decisions as possible in the design process and minimize the number of decisions that need to be made dynamically at runtime.

Scheduling sporadic events

Events that occur sporadically can be associated with interrupts. If the priority of the interrupts is higher than the system priority, the sporadic events are processed when they occur. Such interventions in the scheduling lead to an unpredictable timing behavior for all periodic tasks. To prevent this, special sporadic task servers are used. These sporadic task servers periodically check whether sporadic tasks are ready. Sporadic task servers can be used to convert sporadic tasks into periodic tasks. This improves the predictability of the entire system. [16]

2.4 Concurrency on embedded hardware

Concurrency is ubiquitous in PC application development. Multithreading or processing is used to create concurrent control flows. In the development of embedded software, further forms of concurrency can be classified. [21]

Action that is driven by the program logic.
An application that consists of action traditionally has one entry point (main). Concurrent architectures can have multiple entry points, called tasks. [21]

External events (mainly triggered by peripheral hardware).
Such events are called interrupts. Interrupts are often executed with an increased security level and possibly in a different context than application code. [21]

Interrupts initiated by a task (mixed form of the first two items).
This form of interrupt is called “trap” or “software interrupt”. [21]

Independent hardware actions.
An example of a stand-alone hardware thread is the DMA-controlled filling of

an input buffer of a peripheral interface. [21]

Even with the three most important data transfer mechanisms of computer Input/Output (I/O) devices, two can be classified as concurrent. These data transfer mechanisms are polling, interrupts, and DMA. [22]

Polling does not belong to the concurrent procedures. [21]
But it should be explained here to emphasize one advantage of interrupts, and DMA.

Polling

Polling is the process of the processor capturing incoming data. This is usually performed by a capture subroutine that is called in a holding loop. [22]
Polling occurs periodically and actively by the processor. [21]

Interrupts

Interrupts are interrupting the execution of the main program to store data into a buffer. This data can be called or processed later by the main program. Background data acquisition can be realized by interrupts. Thereby the main program remains free from data polling routines. [22]

DMA

DMA reads data from a device independently of the processor and writes it to a system buffer. This data can be accessed or processed by the processor at a subsequent point in time. The DMA process runs completely independently of the processor and does not interrupt the processor.[22]

The polling strategy processes data by consuming unnecessary Central Processing Unit (CPU) cycles on average with a 50 % delay compared to the processing by an interrupt-based strategy. [21]

The embedded interrupt controller used in this thesis is described in section 2.5.

The strategy of implementing a DMA brings a further increase in data transfer speed compared to the interrupt strategy since a special piece of hardware is responsible for the data transfer. In the ideal case, the processor does not have to execute any instructions to transfer data. [22]

The theory on the DMA can be found in section 2.6.

2.5 Nested Vectored Interrupt Controllers

This chapter only covers the Nested Vectored Interrupt Controllers (NVIC) of Coretex-M4 processors.

The NVIC is an embedded interrupt controller. [21]
Figure 2.3 shows the NVIC in the Cortex-M4 implementation.

Figure 1. STM32 Cortex-M4 implementation

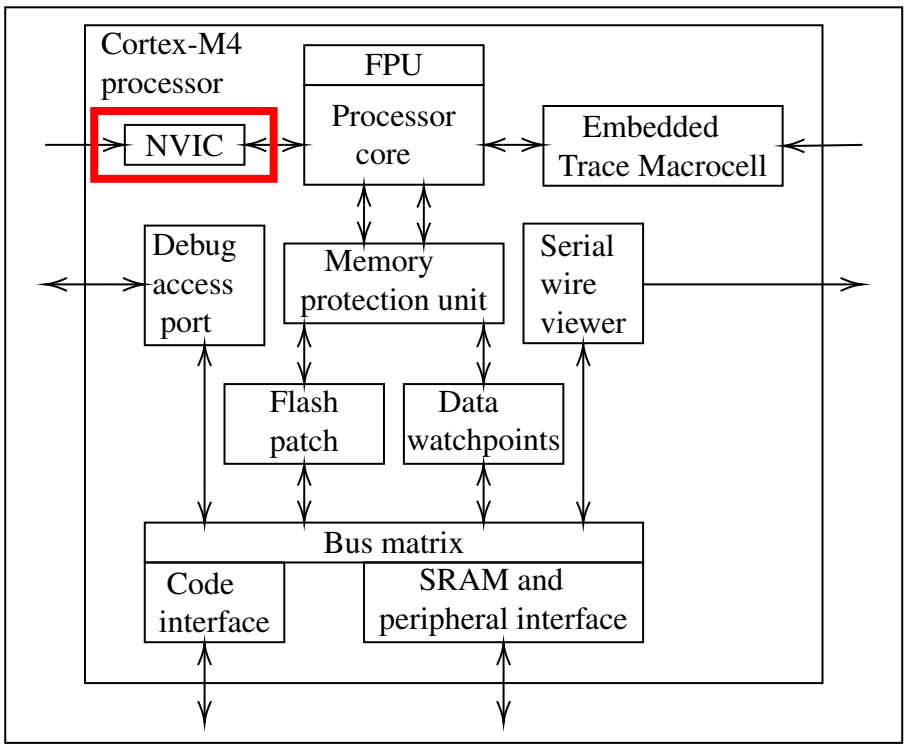


Figure 2.3: NVIC highlighted in the STM32 Cortex-M4 implementation

The maximum number of interrupts within an NVIC is 256, whereby the first 16 Interrupts are reserved for the processor core. All interrupts greater than 15 are defined by the processor manufacturer. This leaves a maximum of 240 interrupts defined by the processor manufacturer. The interrupts are stored in a *vectored* table. [21]

The NVIC allows assigning interrupts to a priority level. This priority level ranges from 0 to 255 for a Cortex-M4 processor. 0 is the highest priority and 255 is the lowest. [21, 23]

The implementation of the Cortex-M4 in the STM32MP157 allows only 16 programmable priority levels (4 bit). [24]

The NVIC supports a group priority mechanism to improve the priority control of the system. The group priority mechanism divides the interrupt priority register into two fields. The group priority and the sub-priority within a group. A higher group priority allows the handler to get ahead. Within a group, the sub-priority decides the order of processing. If several interrupts with the same group priority and the same subpriority are pending, the interrupt with the lowest interrupt number is processed first. This feature allows interrupts to be *nested*. [23]

The NVIC supports “tail chaining” and “late arrivals”. [21]

Tail chaining

If during the processing of an interrupt another interrupt with low priority is pending, the program does not return to the program flow after the processing of the interrupt, instead, the second interrupt is called. The program returns to the interrupted program sequence only after the interrupt *chain* has been processed.[21]

Late arrivals

If during the preparation for the handling of an interrupt (saving the registers on the stack) an interrupt with a higher priority occurs, the processor will handle the higher priority interrupt first and then proceed with the tail chaining. This saves a stacking sequence. [21]

In the following example, an NVIC has 4 group priority levels and 4 sub-priority levels. Four interrupts have been configured for the example. The interrupt priorities are configured according to the scheme {group priority, sub priority}.

- Interrupt 1 {3, 0}
- Interrupt 2 {1, 0}
- Interrupt 3 {1, 1}
- Interrupt 4 {1, 2}

The Interrupt Request (IRQ)s occur in a temporally separated manner as shown in Figure 2.4. Each interrupt results in an Interrupt Service Routine (ISR).

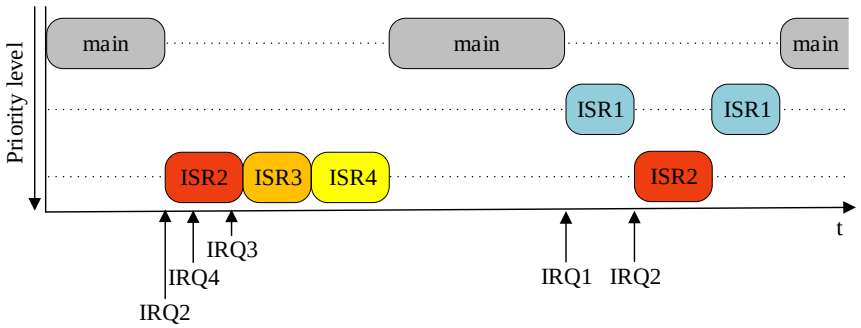


Figure 2.4: Timed processing of interrupts of different priorities

In the beginning, IRQ2 occurs during the processing of the main. While ISR2 is executed, IRQ4 occurs first and then IRQ3. IRQ3 and IRQ4 have the same group priority. But IRQ3 has the lower sub priority. For this reason, ISR3 is executed first and afterward ISR4. After ISR4 is executed the main is executed further. In the second half, IRQ1 occurs. ISR1 starts but is interrupted after a while by IRQ2, the reason being that IRQ2 has a higher group priority. After ISR2 has been processed, the program returns to ISR1. When ISR1 is finished, the program returns to the main.

2.6 Direct Memory Access

DMA refers to data access that has to take place directly between the storage device and the main memory. [22]

The DMA controllers are devices that perform data transfers on behalf of the CPU. The DMA controller can write data directly from an I/O device to a memory, or write data directly from a memory to an I/O device or transfer data directly from memory to another memory. [22]

The DMA controller typically manages multiple DMA channels that can be individually programmed. By activating a hardware DMA request signal, I/O peripherals used for data acquisition can usually signal the DMA controller that data needs to be read or written. Each channel's hardware DMA request signal is passed to the DMA controller, which monitors and handles this signal like how a processor monitors and handles an interrupt. The DMA controller's response to a DMA request is to perform

one or more data transfers. To enable the DMA controller's data transfer, the processor must enable the DMA channels. [22]

DMA controller is operating exactly like the CPU on the system memory and the I/O bus. The DMA controller operates as bus master as well as a bus slave. If the DMA controller operates as bus master, the DMA controller takes over the system bus (control, address, and data lines) from the CPU to transfer the data. [22]

The block diagram of the DMA controller of the STM32MP157 board is shown in Figure 2.5.

Figure 130. DMA block diagram

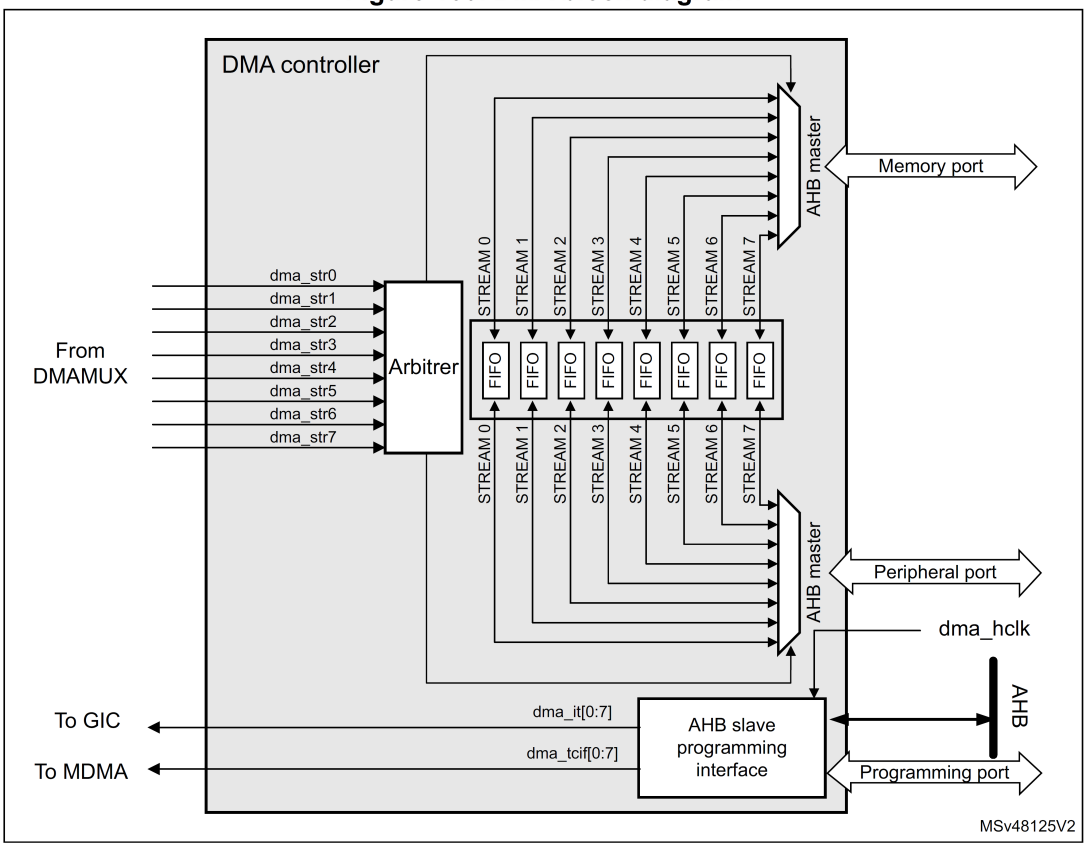


Figure 2.5: Block diagram showing the DMA controller of the STM32MP157 board [24, p. 1193]

The DMA controller implemented in the STM32MP157 executes the direct data transfer as master via the Advanced High-performance Bus (AHB). It is possible to program the channels for the following transactions:

- Memory to peripheral

- Peripheral to memory
- Memory to memory

The DMA controller has an AHB master port for accessing the memory, and another AHB master port for accessing the peripherals. To enable the DMA controller to perform memory to memory transmissions, the peripheral port also has access to the memory. The DMA controller is programmed via the AHB slave port. [24]

2.7 Serial Peripheral Interface

This section describes the SPI. SPI is a master-slave based serial communication protocol, with a data rate between 2 and 25 Mbps. SPI is generally used for communication between connections of devices on the same Printed Circuit Board (PCB). SPI classically uses 4 lines for communication. These communication lines are the clock, data input, data output, and slave select lines. For SPI connections, three frame formats are state of the art. These three frame formats are called Motorola SPI, National Semiconductor Microwire, and Texas Instruments Synchronous Serial Interface. [23]

SPI was first created by Motorola. [25]

Then Texas Instruments and National Semiconductor created their frame formats. [26]

Since master and slave each have a data line for sending and receiving, it is possible to create a parallel-serial data transfer. The SPI master sets the clock via clock line. If the master generates the slave select signal, the master sends one bit on the data output line and receives one bit on the data input line at each clock period. This leads to a full-duplex communication between master and slave. [23]

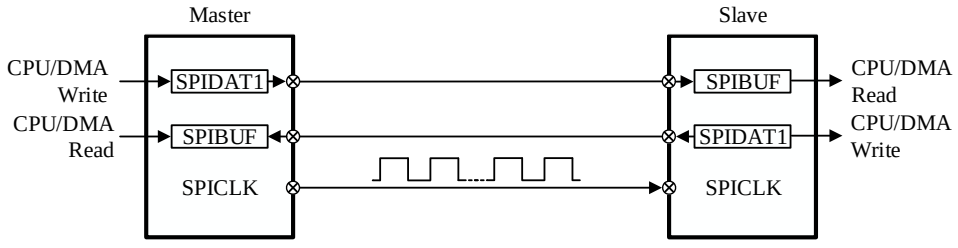


Figure 2.6: SPI routing

2.8 Universal Measurement Protocol

If a parameterization, as well as the simultaneous logging of measurement signals has to be done at runtime of a system, on an Electronic Control Unit (ECU), a rapid prototyping platform, or a Dynamic Link Library (DLL) on a Personal Computer (PC), a physical connection of the system to a development tool is required. Such a physical connection can be the XCP. The "X" of the abbreviation XCP stands for the exchangeability of the transport layer. [27]

The XCP is standardized by the ASAM MCD-1 XCP standard. [28] The Association for Standardization of Automation and Measuring Systems (ASAM) is an association of more than 350 companies in the automotive sector, and has set themselves the task of testing and standardizing toolchains in the automotive industry. [29]

The main objectives in the development of XCP have been a reduction of CPU load, Random-Access Memory (RAM) consumption and flash memory consumption on the XCP slave, as well as a maximization of the data transmission rate on the transport bus. XCP operates using memory type-specific access to read and write data. The standard defines access to parameters and measured variables by memory addresses. The access and interpretation of the data is described by the A2L file. [28]

The A2L file is ASCII readable, it defines interface-specific measurement and calibration parameters, as well as storage schemes, events, and conversion rules. [27]

This avoids the need for a hardcoded data access implementation on the ECU. Calibration and measurement data are stored in a generic XCP stack. [28]

The ECU receives memory access requests from the calibration system at runtime. The ECU responds to these memory access requests. This type of memory access allows different calibration and measurement tasks to be performed by different configurations of the calibration system without having to modify and recompile the code of the ECU. The XCP contains transport layer definitions for Ethernet (UDP/IP and TCP/IP), USB, FlexRay, CAN, and serial connections (SPI and SCI). [28]

The XCP is a packet-based master-slave principle. The calibration system is the host and the ECU is the slave. A slave can only communicate with one master. The master can communicate with several slaves. An example XCP bus structure is shown in Figure 2.7. [27]

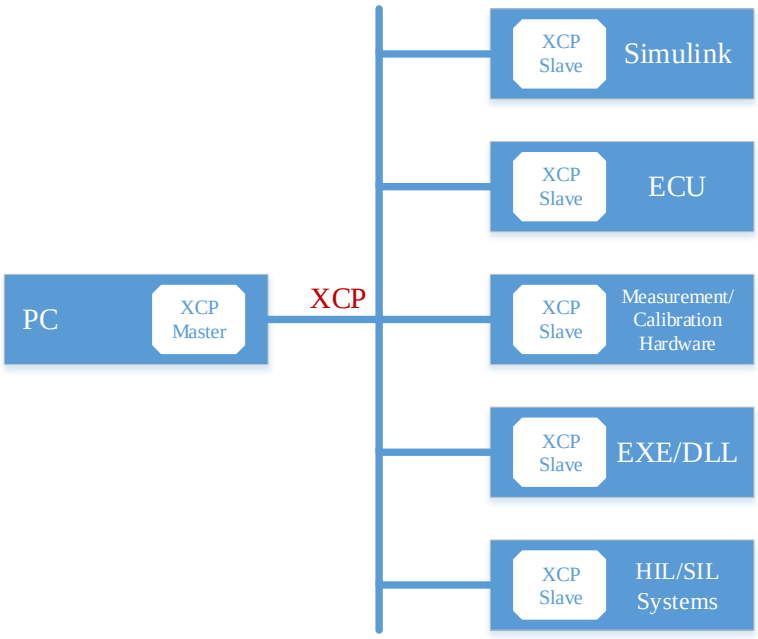


Figure 2.7: An example of a master-slave topology (cf. [27, p. 15])

An XCP message can be divided into two categories:

Command Transfer Object (CTO)

CTOs transmit commands. They are sent from the master to the slave. The slave reacts with a positive or negative response. Commands are for example CONNECT, UPLOAD, DOWNLOAD, MODIFY_BITS. These commands are each assigned to a unique number. [27]

Data Transfer Object (DTO)

DTOs are used for the exchange of synchronous measurement and adjustment data. The slave sends data synchronously to internal events via Data Acquisition (DAQ). [27]

DAQ is a measurement method that is used to transfer DTOs from slave to master. The Data exchange via DAQ is processed as follows:

The data exchange of DTOs is divided into two phases. An initialization phase includes the master instructing the slave which data have to be sent in response to the various events. At the end of the initialization phase, the master starts the measurement phase

at the slave. In the measuring phase the slave sends the requested data to the master. This happens without the request from the master. The measurement continues until a stop command is sent from the master to the slave. The data stream from the master to the slave also occurs in two phases. During initialization, the master advises the slave which data it will send during the measurement phase. During the measuring phase, the master sends the data to the slave, which transfers this data to the application. [27]
A DAQ-list is created from several Object Descriptor Table (ODT)s. To identify measuring objects unambiguously, their address and length are important. These two parameters are stored in an ODT.
Schematic Figure 2.8 shows the structure of an ODT. [27]

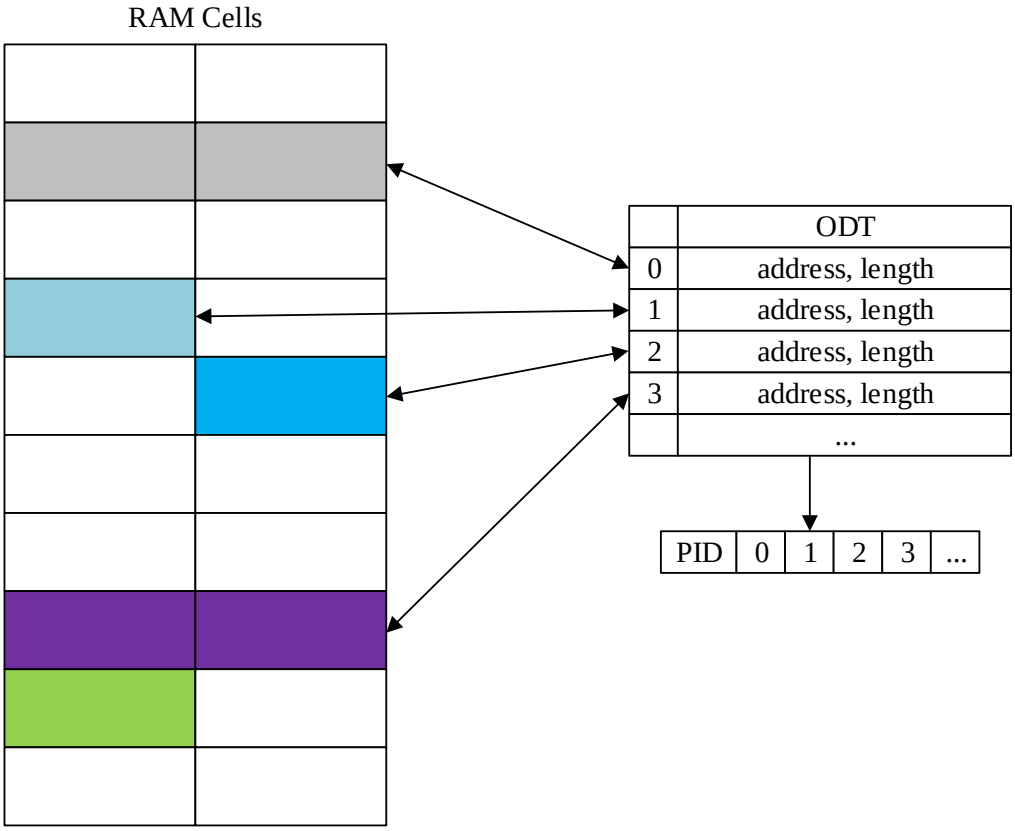


Figure 2.8: Assignment of memory locations of the slave to DAQs by ODTs (cf. [27, p. 38])

Several ODTs are combined into one DAQ list. As seen in Figure 2.9.

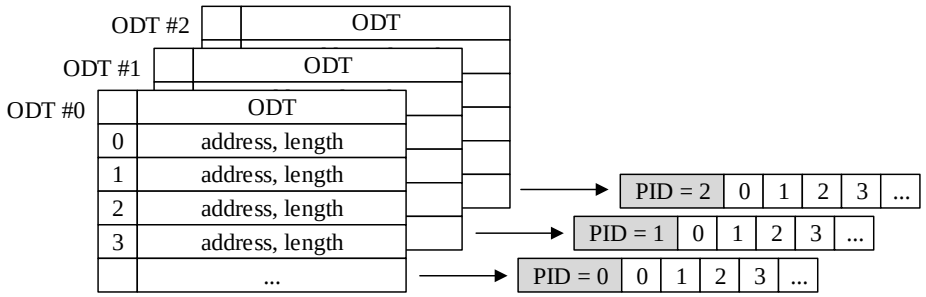


Figure 2.9: Example DAQ-list from three ODTs (cf. [27, p. 39])

2.9 CMake

CMake is a compiler-independent opensource system that manages the build process within an operating system. CMake was developed to fill the need for a cross-platform build environment for the Insight Segmentation and Registration Toolkit. The first implementation was done in 2000, when Bill Hoffman of Kitware took some of the key ideas from pcmaker, an earlier system, and extended them. CMake is extensible, and designed to have the ability to be used in conjunction with the native build environment. CMake works by placing configuration files in those source directories that are needed for the build process. These files are called CMakeLists.txt. Through these configuration files, standard build files such as makefiles on Unix and projects/workspaces on Windows Microsoft Visual C++ (MSVC) are created. These standard build files created by the configuration files compile source code, create wrappers, build libraries and executables. Libraries can be built static or dynamic by CMake. Through the support of in-place and out-of-place builds, multiple build processes can be created from one source tree. [30]

2.10 Simulink code generation process

In this section, an overview of the code generation process of MATLAB Simulink [8] is given.

The base of the code generation process using MATLAB Simulinkis is the System Tar-

get File (STF). The STF contains information about the code generation sequence of the process. In the STF, variables can be defined that are needed during code generation, such as the Target Language Compiler (TLC) variable that define the code format. STFs have the ability to inherit the properties of other STFs through inclusion. STFs are TLC files. [31]

TLC files are the files that control the way code is generated. They allow to generate platform specific code or to make adjustments in terms of runtime, code size or compatibility. [32]
Figure 2.10 shows in which code generation step TLC files are used.

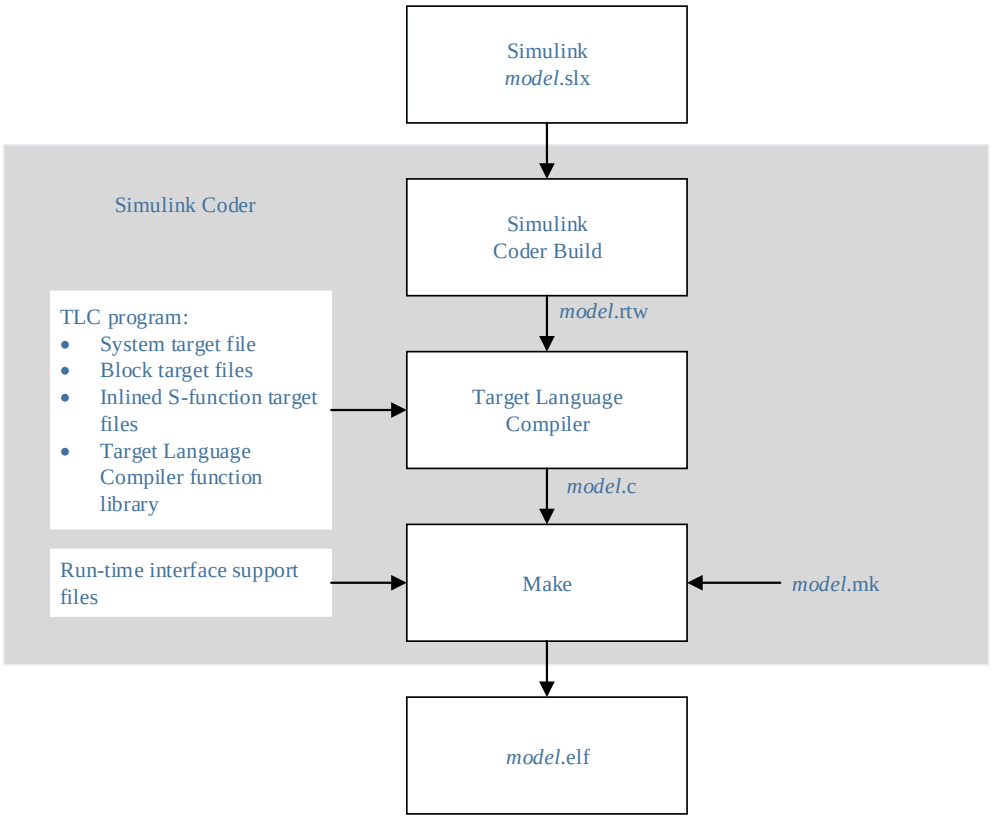


Figure 2.10: Position of the TLC file in the code generation process (cf.[32])

In figure 2.10 the code generation process of a simulink model (`model.slx`) to an executable (`model.elf`) for a hardware processor is presented.
The Simulink Coder creates a `model.rtw` file from a Simulink model. The `.rtw`-file

describes inputs, outputs, parameters, memory, states and other model components and their properties. The created `model.rtw` is then processed as input in the TLC. [33]

The Target Language Compiler generates source files and Makefile files from the `.rtw` file depending on the specified `.tlc` files. That can be seen in figure 2.11. The Makefile is used to build the source files into an executable. [32]

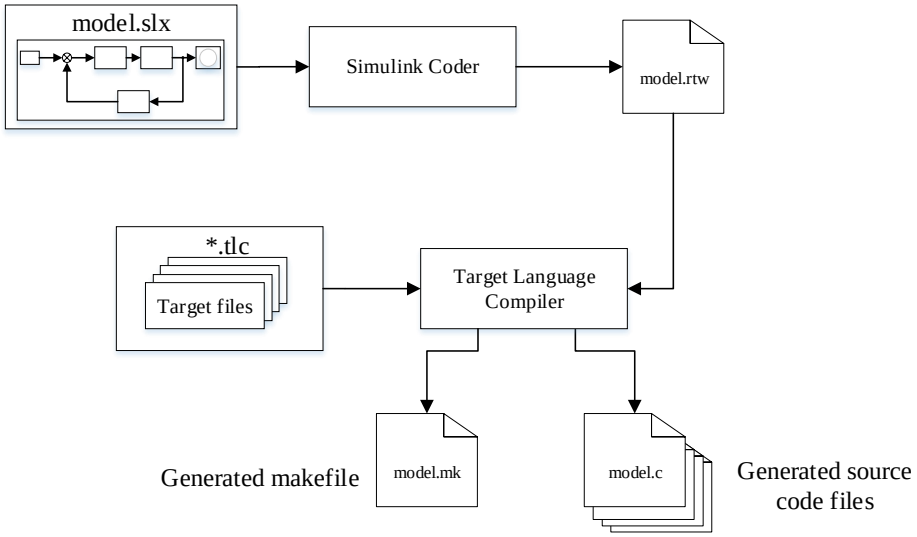


Figure 2.11: Schematic representation of processing the `model.rtw` file during code generation (cf.[32])

2.11 Kalman filter

The Kalman filter is a filter that computes states of a system for linear discrete-time signals by measuring noisy and partially redundant signals, using stochastic estimation techniques. Compared to many other stochastic estimation methods, the Kalman filter can be constructed iteratively and is therefore suitable for use in real-time systems. For the use of the Kalman filter, basic knowledge about state-space models is required. [34]

These are taken for granted in this chapter. Educational material on state-space models are available in sources [34, p. 23] [35, p. 633] [36, p. 16].

For the use of the Kalman filter, a system is required that has, for example, three interrelated variables. Of these three variables, two are measured and one is estimated. In

this example these variables are the measured acceleration $a(t)$, the estimated velocity $v(t)$ and the measured position $h(t)$. [34, p. 8]

$$a(t) = \dot{v}(t) = \ddot{h}(t) \tag{2.7}$$

This model is inserted into the state-space model. The equations of the state-space model are given in equation (2.8a) (state differential equation) and equation (2.8b) (output equation). [34, p. 7]

$$\dot{\underline{x}}(t) = \underline{A} \cdot \underline{x}(t) + \underline{B} \cdot \underline{u}(t) + \underline{G} \cdot \underline{z}(t) \tag{2.8a}$$

$$\underline{y}(t) = \underline{C} \cdot \underline{x}(t) + \underline{D} \cdot \underline{u}(t) \tag{2.8b}$$

Where $\underline{x}(t)$ is the state vector, $\underline{u}(t)$ is the input vector, $\underline{y}(t)$ is the output vector, $\underline{z}(t)$ is the system noise/process noise, $\underline{A}$ is the system matrix, $\underline{B}$ is the input matrix, $\underline{C}$ is the output matrix, $\underline{D}$ is the feedthrough matrix and $\underline{G}$ is the matrix of the system noise. [34]
If the state vector $\underline{x}(t)$ is now defined as shown in equation (2.9a), this results in the derived state vector $\dot{\underline{x}}(t)$, shown in equation (2.9b). [34, p. 8]

$$\underline{x}(t) = \begin{bmatrix} h(t) \\ v(t) \\ a(t) \end{bmatrix} \tag{2.9a}$$

$$\dot{\underline{x}}(t) = \begin{bmatrix} \dot{h}(t) \\ \dot{v}(t) \\ \dot{a}(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ a(t) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot \underline{z}(t) \tag{2.9b}$$

In equation (2.9b) it is simplified that the derivative of $a(t)$ results in zero. The change of $a(t)$ is taken into the system description by the system noise $z(t)$. [34]
If now the state-space equations are set up, taking into mind that the output vector $\underline{y}(t)$ consists of the measured quantities $h(t)$ and $a(t)$, the result is: [34, p. 8]

$$\dot{\underline{x}}(t) = \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}}_{A} \cdot \underline{x}(t) + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}_{B} \cdot u(t) + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{G} \cdot z(t) \tag{2.10a}$$

$$\underline{y}(t) = \begin{bmatrix} h(t) \\ a(t) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{C} \cdot \underline{x}(t) + \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{D} \cdot u(t) \tag{2.10b}$$

Discretization of the state-space model

To apply the filter to a digital system with a sampling time of T_s, it is necessary to transform the state-space model into the discrete-time domain. This is done with the help of the following equations: [34, p. 9]

$$\underline{A}_d = e^{\underline{A} \cdot T_s}, \qquad \underline{B}_d = \int_0^{T_s} e^{\underline{A} \cdot v} \cdot \underline{B}\, dv, \qquad \underline{G}_d = \underline{A}_d \cdot \underline{G} \tag{2.11}$$

The equations for the state-space model are then: [34, p. 9]

$$\underline{x}(k+1) = \underline{A}_d \cdot \underline{x}(k) + \underline{B}_d \cdot \underline{u}(k) + \underline{G}_d \cdot \underline{z}(d) \tag{2.12a}$$

$$\underline{y}(k) = \underline{C} \cdot \underline{x}(k) + \underline{D} \cdot \underline{u}(k) \tag{2.12b}$$

Results: [34, p. 9]

$$\underline{A}_d = \begin{bmatrix} 1 & T_s & \frac{T_s}{2} \\ 0 & 1 & T_s \\ 0 & 0 & 1 \end{bmatrix}, \qquad \underline{B}_d = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \qquad \underline{G}_d = \begin{bmatrix} \frac{T_s^2}{2} \\ T_s \\ 1 \end{bmatrix} \tag{2.13}$$

Observability of the system description

Before the Kalman filter can be applied to any system description, it is necessary to check whether the observability of the system is given. Observability describes whether, with a known input variable u(t) and a known output variable y(t), each state of the system can be determined within finite time. [34]

For a system to be observable, the observability matrix $\underline{S_B}$ must have rank n for a system of n-order: [34, p. 10]

$$\underline{S_B} = \begin{bmatrix} \underline{C} \\ \underline{C} \cdot \underline{A} \\ \underline{C} \cdot \underline{A}^2 \\ \vdots \\ \underline{C} \cdot \underline{A}^{n-1} \end{bmatrix} \tag{2.14}$$

To ensure that the discretized system is also observable, this should be tested on the discretized system. [34] The observation matrix for the discretized system $\underline{S_B^*}$ is shown in equation (2.15): [34, p. 11]

$$\underline{S_B^*} = \begin{bmatrix} \underline{C} \\ \underline{C} \cdot \underline{A}_d \\ \underline{C} \cdot \underline{A}_d^2 \\ \vdots \\ \underline{C} \cdot \underline{A}_d^{n-1} \end{bmatrix} \tag{2.15}$$

As soon as $\underline{S_B^*}$ has rank n, the system is observable. [37]
For the example system of $a(t)$, $v(t)$, $h(t)$ ($n = 3$), the rank of $\underline{S_B^*}$ is formed as follows: [34, p. 11]

$$\text{Rang}(\underline{S_B^*}) = \text{Rang}\left(\begin{bmatrix} \underline{C} \\ \underline{C} \cdot \underline{A}_d \\ \underline{C} \cdot \underline{A}_d^2 \end{bmatrix} \right) = 3 \tag{2.16}$$

If it is assumed that $T_s > 0$, the system of equations can be solved. The example is observable, because the rank of the observation matrix is equal to the order n of the system. [34]

System and measurement noise

When using a Kalman filter, it is necessary to identify the system and measurement noise. System errors are caused by model inaccuracies, measurement errors by noise of the sampled signal. It is only necessary to determine the variance of the errors. It is assumed that the noise sources are without mean values. [34]
In the example it is assumed that the derivative of $a(t)$ results in zero, and possible changes are described by the noise quantity $z(k)$. From the system noise, the variance $Q(k)$ can be determined as shown in eq2: [34, p. 12]

$$Q(k) = \mathrm{Var}(z(k)) = \sigma_v^2 \tag{2.17}$$

Since in the example, only one noise quantity appears in the modeling, the random quantity $z(k)$ and thus also the variance $Q(k)$ are scalar. [34]
In the design of the Kalman filter, it is assumed that the estimation error and system noise are uncorrelated. If further assumptions are made, the noise is without mean value, normally distributed, and the noise is white noise, the variance can be estimated as in the following example. [34]
Example: [34, p. 13] Maximum acceleration delta $10\,\mathrm{m\,s^{-2}}$ during T_s Maximum acceleration change is equal to $3 \cdot \sigma$
Calculation: [34, p. 13]

$$Q(k) = \sigma_v^2 = \left(\tfrac{10\,\mathrm{m}}{3\,\mathrm{s}^2}\right)^2 \approx 11.1\,\mathrm{m^2\,s^{-4}} \tag{2.18}$$

Measurement noise is generated, for example, by the quantization of signals or by other disturbances. To include measurement noise in the system, the measurement noise $\underline{v}(k)$ is superimposed on the output signal. The discrete system is then described as shown in equation (2.19b). [34, p. 13]

$$\underline{x}(k+1) = \underline{A}_d \cdot \underline{x}(k) + \underline{B}_d \cdot \underline{u}(k) + \underline{G}_d \cdot \underline{z}(d) \tag{2.19a}$$

$$\underline{y}(k) = \underline{C} \cdot \underline{x}(k) + \underline{D} \cdot \underline{u}(k) + \underline{v}(k) \tag{2.19b}$$

It is important to note that the input vector $\underline{u}(t)$ remains unchanged and that the estimation error and the measurement noise must be uncorrelated. If it is further assumed that the measurement noise is a noise without mean value, normally distributed, and white noise, the expected value $\mathrm{E}(\underline{v}(k)) = \underline{0}$. [34]
The equation for calculating the variance of the measurement noise is given in equation (2.20). [34, p. 14]

$$\underline{R}(k) = \mathrm{Var}(\underline{v}(t)) \tag{2.20}$$

If the input signals of the example are assumed to be without mean value, normally distributed, their variances can be calculated by $\mathrm{Var}(\underline{v}_h(k)) = \sigma_h^2$ and $\mathrm{Var}(\underline{v}_a(k)) = \sigma_a^2$. [34]
Furthermore, it is assumed that the two measurement noise variables do not influence each other, which means that they are stochastically independent. It follows that their covariance $\mathrm{Cov}(\underline{v}_h(k), \underline{v}_a(k))$ is equal to zero. [34]
If the measurement noise values remain the same over time or change only insignificantly, then the variance σ_h^2 and σ_a^2 can be estimated empirically by equation (2.21). [34, p. 14]

$$\mathrm{Var}(x) = \frac{1}{n-1} \cdot \sum_{k=1}^{n} (x(k) - \mathrm{E}(x))^2 \tag{2.21}$$

If $\sigma_h^2 \approx 20\,\mathrm{m}^2$ and $\sigma_a^2 \approx 0.2\,\mathrm{m}^2\,\mathrm{s}^{-4}$ are estimated for the example, $\underline{R}(k)$ is calculated as follows: [34, p. 14]

$$\underline{R}(k) = \mathrm{Var}(v(k)) = \begin{pmatrix} \mathrm{Var}(v_h(k)) & \mathrm{Cov}(v_h(k), v_a(k)) \\ \mathrm{Cov}(v_h(k), v_a(k)) & \mathrm{Var}(v_a(k)) \end{pmatrix} \tag{2.22a}$$

$$= \begin{pmatrix} \sigma_h^2 & 0 \\ 0 & \sigma_a^2 \end{pmatrix} \approx \begin{pmatrix} 20\,\mathrm{m}^2 & 0 \\ 0 & 0.2\,\mathrm{m}^2\,\mathrm{s}^{-4} \end{pmatrix} \tag{2.22b}$$

The variance $\underline{R}(k)$ describes how reliable the measured values are. In most cases, the state variables will change only slightly, if the noise variables are not estimated correctly. [34]

Kalman filter equation

The principle equations developed by Kalman are shown in equation (2.23) and equation (2.24) and are taken from [34, p. 15].
Correction:

$$\underline{\hat{y}}(k) = \underline{C} \cdot \underline{\hat{x}}(k) + \underline{D} \cdot \underline{u}(k) \tag{2.23a}$$

$$\Delta\underline{y}(k) = \underline{y}(k) - \underline{\hat{y}}(k) \tag{2.23b}$$

$$\underline{K}(k) = \underline{\hat{P}}(k) \cdot \underline{C}^T \cdot (\underline{C} \cdot \underline{\hat{P}}(k) \cdot \underline{C}^T + \underline{R}(k))^{-1} \tag{2.23c}$$

$$\underline{\tilde{x}}(k) = \underline{\hat{x}}(k) + \underline{K}(k) \cdot \Delta\underline{y}(k) \tag{2.23d}$$

$$\underline{\tilde{P}}(k) = (\underline{I} - \underline{K}(k) \cdot \underline{C}) \cdot \underline{\tilde{P}}(k) \tag{2.23e}$$

Prediction:

$$\underline{\hat{x}}(k+1) = \underline{A}_d \cdot \underline{\tilde{x}}(k) + \underline{B}_d \cdot \underline{u}(k) \tag{2.24a}$$

$$\underline{\hat{P}}(k+1) = \underline{A}_d \cdot \underline{\tilde{P}}(k) \cdot \underline{A}_d^T + \underline{G}_d \cdot \underline{Q}(k) \cdot \underline{G}_d^T \tag{2.24b}$$

The derivation of the Kalman equation can be found under [34, p. 85].
In the Kalman equations (2.23) and (2.24), it can be seen that an $\underline{\hat{x}}(t)$, and an $\underline{\tilde{x}}(t)$ vector are calculated. The state vector $\underline{\hat{x}}(t)$ is the predicted state vector, and $\underline{\tilde{x}}(t)$ is the corrected state vector.
The Kalman filter is accessed as follows: [34]

1. Calculation of the difference $\Delta\underline{y}(k)$ between the output variable $\underline{\hat{y}}$ and the current $\underline{y}$, seen in equation (2.23b)
2. The Kalman gain $\underline{K}(k)$ is calculated in equation (2.23c) to correct the estimated state vector $\underline{\hat{x}}(k)$, by computing the corrected state vector $\underline{\hat{x}}(t)$ in equation (2.23d)
3. In equation (2.23e) the covariance of the estimation error $\underline{\varepsilon}(k)$ is calculated, the estimation error is the difference of $\underline{\tilde{\varepsilon}}(k) = \underline{x}(k) - \underline{\tilde{x}}(k)$.
4. In equation (2.24b), the covariance of the estimation error $\underline{\hat{P}}(k)$ is extrapolated.

By using equation (2.23a), equation (2.23b), equation (2.23c), equation (2.23d), equation (2.23e), equation (2.24a), and equation (2.24b), the velocity can be estimated from the example.

3 Problem Analysis and Requirements

In order to determine the requirements of a mechatronic system, the CONceptual design Specification technique for the ENgineering of complex Systems (CONSENS) method initially sets up an environment model. Subsequently, requirements can be specified from the environment model. [38]

The environment model created for this purpose can be seen in Figure 3.1. It is intended to represent the information flows between the individual components. The energy flows are neglected in this environment model. The environment model provides an overview of which interfaces are implemented.

The entire requirement specification, can be found in the item A.1.1.

The requirements specification also includes customer specifications. For example, the firmware, that is executed on the Arm-based Cortex-M4 core, must be created by code generation using MATLAB/Simulink. In Simulink, the model must be monitored in External mode via XCP on TCP/IP. The External mode must also provide the ability to tune model parameters during runtime.

3.1 Environment model

The environment model in Figure 3.1 shows the Microprocessor Unit (MPU), represented in a blue system element. This figure does not show that there are two independent processors within the system element STM32MP1. This is illustrated in the hardware components connection map in chapter 4 figure 4.3.

The used hardware components of the self-balancing robot are explained in more detail in the Figure 5.6. The physical connection between the MPU and the hardware components of the self-balancing robot is established by a connection board.

Sensors and actuators constitute the most important environment elements. They are located on the driver board of the self-balancing robot. [39]

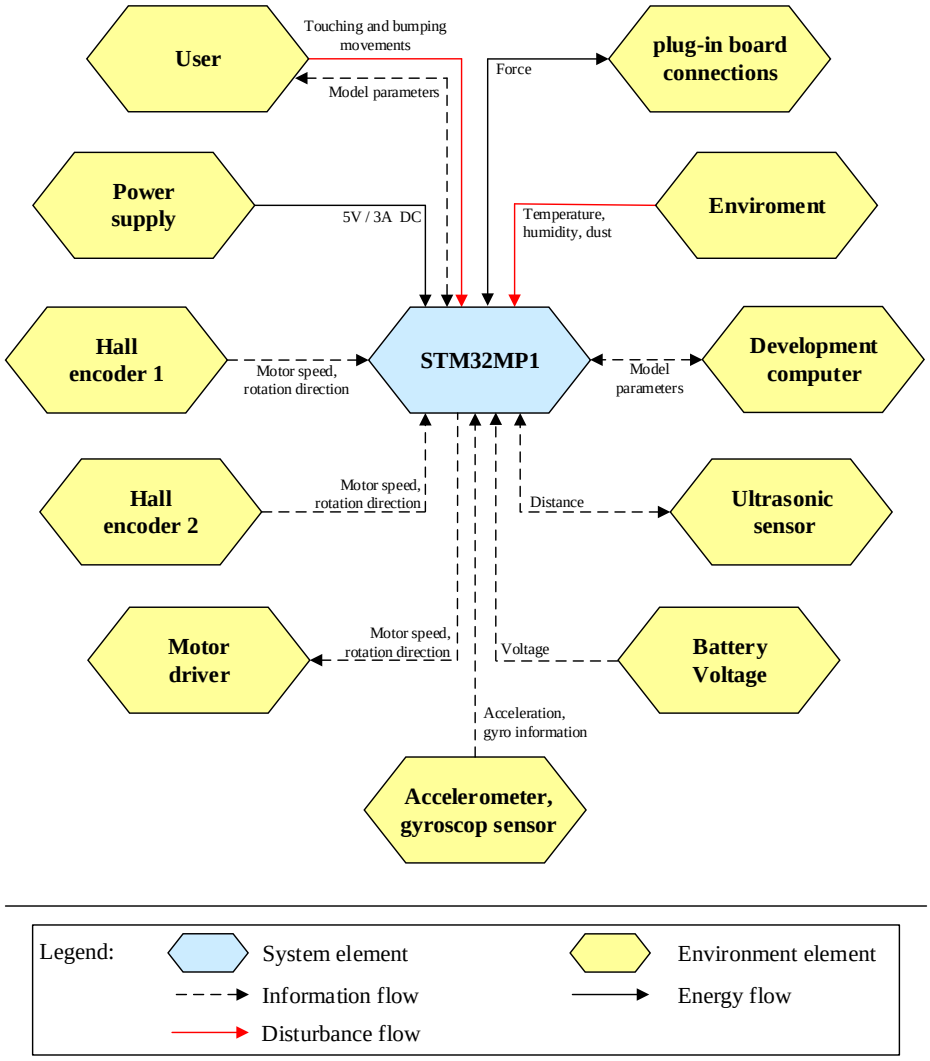


Figure 3.1: Environment model

3.2 Application scenarios

The applications scenario can be divided into two points:

Model-based design: Using MATLAB Simulink for rapid prototyping on the Cortex-M4.

The example application: The implement of software components for the balancing control of the self-balancing robot is used to show that the developed Simulink

target matches real-time boundaries. The application shows also the advantages of heterogeneous multiprocessor platforms.

3.3 System Requirements

The requirements specification A.1.1 lists all requirements for the software components for the MATLAB Simulink target for the STM32MP1 to be developed. The requirements for the development of software components for the example application are also listed in the requirements specification. The example application is a software to control the self-balancing robot. This example application is intended to show that the developed MATLAB Simulink target can match real-time requirements. item A.1.1.

The requirements specification lists the requirements, their risks, and the methods used to verify these requirements. The purpose of the requirements specification is to capture all requirements that are defined for the software components because it is necessary to verify if the requirements comply with the implementation.

4 Software design

The design process is divided into two sections. The first section describes the design of the MATLAB Simulink coder target for the real-time system, which is executed on the Cortex-M4 processor, and the design of a data bridge, whose task is the data exchange between the MATLAB development computer and the real-time system. This data bridge is executed on the Cortex-A7 processors. The second section describes the design of an example real-time application. Within this application, an inverted pendulum is to be controlled. The real-time capability of the system is demonstrated by this control. In addition, a graphical user interface for parameter visualization and the ability of input parameter adjusting is designed. The addition graphical interface is not a subject of the real-time System. The main task of the graphical application is to demonstrate that during the independent execution of the real-time application on the Cortex-M4, all the advantages provided by an embedded Linux system could be used.

4.1 Design of the Simulink coder target

One of the main decisions in the design process and the design of the software architecture is defined by the requirements specification item A.1.1. The Req_01 specifies that the model step must be called by a timer interrupt. In the previous coder target [40], the model step is called by a FreeRTOS [41] task. Req_02 specifies that external mode via XCP must be integrated into the Simulink code generation process.

According to [31] models can be executed in real-time on the target hardware if the model step is managed by a real-time operating system, or if the model step is called in the context of an ISR on bare-metal target hardware.

The planning therefore provides that the model step is called by a timer interrupt. The design of how the model steps are to be called is shown in figure 4.1.

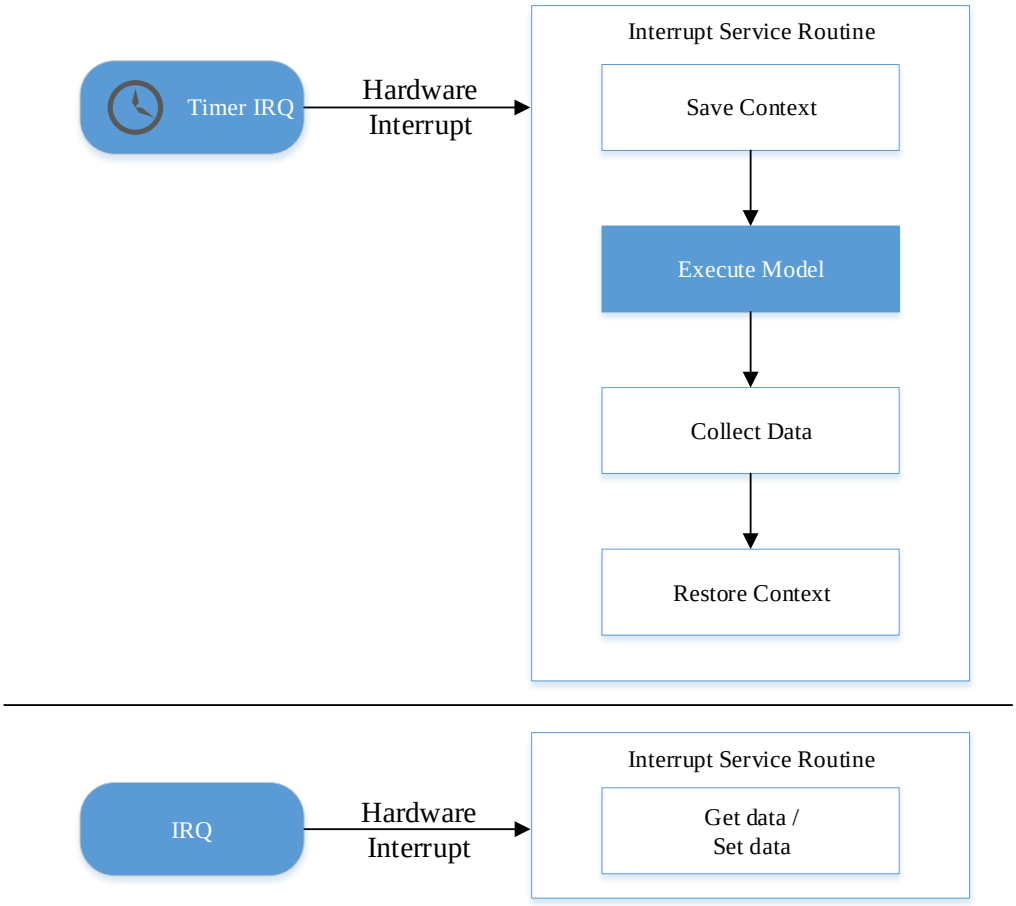


Figure 4.1: In the upper image area, the call of the model step is shown. Separately from this area, it can be seen that further peripheral interrupts can occur independently from the processing of the model step.

To enable the generation of asynchronous hardware interrupts, a hardware interrupt block must be created. The asynchronous interrupts are used to read or to write data. The implementation of asynchronous interrupts is described in section 5.4.

The data packages of the external mode must be sent via the TCP/IP over Ethernet or Universal Serial Bus Host (USBH). According to [14] it is not possible to map these interfaces to the Cortex-M4 processor of the STM32MP1. This makes it necessary to route the XCP messages via the shared memory over the Cortex-A7 processor. This Cortex-A7 processor must pass the messages bi-directionally via a TCP/IP server to the host computer on which the Simulink process is executed. This requires a connec-

tion between the development computer and the Cortex-A7 processor. To create this connection a TCP/IP server is set up on the Cortex-A7. The host computer is the client for the TCP/IP connection. The XCP master is executed on the host computer [42], the Cortex-M4 processor assumes the role of the client for the XCP connection. From the XCP point of view, the Cortex-A7 processor only passes the messages and does not participate in the actual XCP communication.
The communication process described, is shown in figure 4.2.

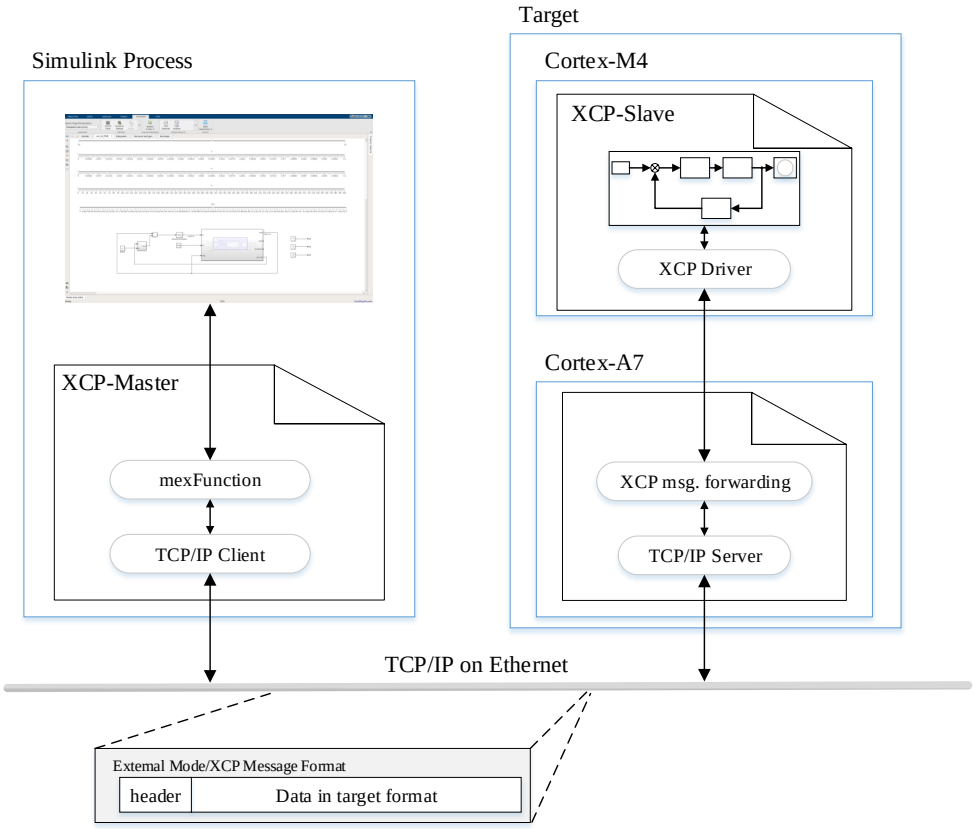


Figure 4.2: Communication design for the implementation of the External Mode communication using XCP over TCP/IP

4.2 Design of the distributed system

The design of the distributed application is based on a closer look at the existing hardware. Thereby it is extracted how the individual hardware devices must be controlled.

After considering the control of the hardware devices, the hardware component connection diagram shown in figure 4.3 is drawn up.

This plan shows how the hardware has to be connected to the peripheral devices of the STM32MP1. During this mapping process, it must be decided if the hardware devices are assigned to the real-time capable system (firmware running on the Cortex-M4) or to the system running the Linux operating system (application running on the Cortex-A7).

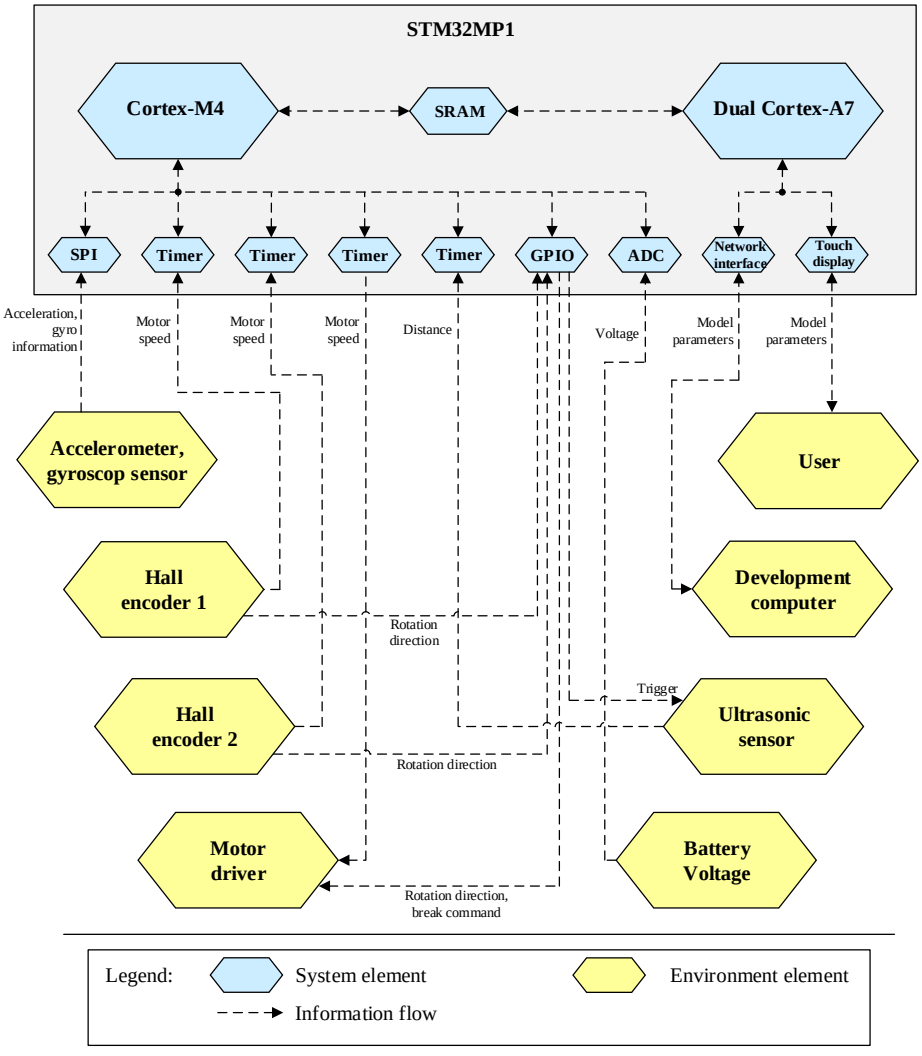


Figure 4.3: Hardware components connection map

Figure 4.3 shows a schematic diagram of the STM32MP1 MPU on the top of the pic-

ture in the gray box. The Cortex-M4 processor is shown in the gray box on the left side. It is connected to the Cortex-A7 processor via the SRAM block. Below the processors, the required peripheral hardware of the STM32MP1 is shown schematically. These are connected to the yellow environment elements. These represent the given hardware of the self-balancing robot, as well as the user and the development computer.

The decision whether to assign the yellow environment elements to the Cortex-M4 or the Cortex-A7 is based on several criteria:

1. Which processor can the hardware peripheral block of the MPU be assigned to during pin configuration via STM32CubeMX 4.1
2. Is it important that the element is integrated into the real-time capable application? 4.2
3. On which processor is the estimated implementation effort minimized?4.3

Table 4.1 were collected by using STM32CubeMX [43] via the STM32CubeMP1 v1.4.0 ecosystem [44].

Hardware peripheral block	ARM Cortex-M4	Dual ARM Cortex-A7
SPI	yes	yes
Timer	yes	yes
GPIO	yes	yes
ADC	yes	yes
Network interface	no	yes
Touch display	no	yes

Table 4.1: Mapping of hardware configuration possibilities using STM32CubeMX

Table 4.2 is based on information that was collected in the requirement specification A.1.1.

Interfaces	Real-time capability
Accelerometer, gyroscop sensor	yes
Hall sensor 1 & 2	yes
Motor driver	yes
Battery voltage	no
Ultrasonic sensor	yes
Development computer	no
User	no

Table 4.2: Interface real-time capability required

The information shown in table 4.3 cannot be reconstructed. It is based on the empirical values collected within preliminary projects at the STM32MP1 MPU.

Hardware peripheral block	Effort on Cortex-M4	Effort on dual Cortex-A7
SPI	medium	high
Timer	low	medium
GPIO	low	low
ADC	medium	high
Network interface	not possible	low
Touch display	not possible	medium

Table 4.3: Estimated implementation effort on Arm-based Cortex-M4 compared to the estimated implementation effort on Arm-based dual Cortex-A7

The assignment of the hardware peripheries to the processors is derived from tables 4.1, 4.2, and 4.3. This assignment is shown in figure 4.3.

After assigning the peripheries to the processors, the applications for the two processors are planned. This is described in the following section 4.2 and section 5.9.

Design of the real-time application

The real-time application, is implemented by the MATLAB Simulink target.
To control the hardware peripheral blocks assigned to the real-time system by MATLAB Simulink, hardware-related Simulink blocks are developed for these peripheries. For this, the functions, shown in table 4.4 must be provided by hardware-related Simulink blocks.

Hardware peripheral block	Function
SPI	Sending data packages
	Receiving data packets
Timer	Set PWM
	Get Counter
GPIO	Set outputs
	Get inputs
ADC	Get ADC data value

Table 4.4: Planned hardware-related Simulink blocks

The implementation of the hardware-related Simulink blocks is described in section 5.5.

To develop a concurrent real-time application, the polling, explained in section 2.4, has to be avoided. To banish polling from the application, the implementation of hardware specific interrupts and DMA is used. The real-time application is divided into serveral ISRs. During the following consideration all occurring ISRs are called tasks.
To perform scheduling, the required tasks are analyzed. The tasks are shown in table 4.5. This requires the implementation of the hardware-related Smulink blocks.
To analyze the task, the system priority of the task is configured with the lowest adjustable system priority. To be able to observe the times when the task is active from the outside, a GPIO is set to the high level at the beginning of the task, and the GPIO is reset when the task is completed. At the toggling GPIO, the duration, as well as the periodicity of the task, can be analyzed by using a high-frequency oscilloscope 4.7. The firmware used for analysis is generated with the external mode and with the use of the compiler optimization `-O3`.

Task no. i	Task short description
0	Model task, executes the generated Simulink model.
1	Sensor has values task, detects when in the values of the MPU6500 are ready for reading.
2	Sensor values received task, the task is executed when the sensor data has been received via DMA.
3	Sensor value request transmission commplet task, the task is executed when data has been transmitted to the MPU6500.
4	Set PWM Values task, the task sets the PWM Duty Cycle.
5	Get Timer Task, the task reads the counter register of the timer and the level of a GPIO pin.
6	Get Timer Task, the task reads the counter register of the timer and the level of a GPIO pin.
7	IPCC message received task, the task detects whether a message has been received from the main processor.
8	ADC half received task, the task signals that the half data of the ADC DMA transmission has been done.
9	ADC complete received task, the task signals that the complete ADC DMA transmission has been done.
10	Get Timer Task, the task reads the counter register of the timer.
11	Get Timer Task, the task reads the counter register of the timer.
12	IPCC message received task, the task detects whether a message has been send from the main processor.

Table 4.5: List of required tasks in the real-time application

To provide an overview, table 4.6 assigns the tasks to the respective hardware components.

Hardware component	Interrupt source	Task no.
Simulink model	Timer	0
Accelerometer, gyroscop sensor	EXTI	1
	DMA-Streams	2, 3
Motor driver	Timer	4
Hall encoder 1	Timer	5
Hall encoder 2	Timer	6
Development Computer	IPCC	7
Battery voltage measuring point	DMA-Stream	8, 9
Ultrasonic sensor	Timer	10, 11
User	IPCC	12

Table 4.6: Overview of tasks, hardware components and interrupt sources

Table 4.7 shows the high-frequency oscilloscope used for the analysis of the tasks.

Measuring device	KEYSIGHT MSOS054A
Series number	MY57160102
Inventory number	170722

Table 4.7: Measuring device

Table 4.8 shows the measured period times T_i and the worst case execution time C_i in μs. The measurements were taken with the measuring instrument shown in table 4.7. In table 4.8 it can be seen that in column T_i the tasks τ_0, τ_4, τ_8 τ_9, τ_{10}, and τ_{11} have been assigned the value var_i. This is because these tasks are dependent on time-variable events.

Task no. i	T_i / µs	C_i / µs
0	var_i	99.818
1	999.47	46.467
2	997.62	5.4812
3	997.91	4.0167
4	var_i	6.8316
5	501.57	4.1124
6	501.57	4.1124
7	1000000	2.9357
8	var_i	4.6819
9	var_i	4.6437
10	var_i	6.4102
11	var_i	6.4102
12	8907.7	1.2778

Table 4.8: Period T_i and worst-case execution time C_i of task system τ

Dependencies of the time-variable events:

Task τ_0: T_0 of τ_0 depends on the model step of the control system. This is determined in section 5.8.

Task τ_4: τ_4 defines how often the Pulse Width Modulation (PWM) value of the motors is set. The T_4 value can be set via the block mask parmeter Frequency ($= 1/T_4$), shown in figure 5.15, of the block from section 5.5.

Task τ_8: T_8 depends on how fast the ADC data request block 5.5 is called. The sample time of the block position within the Simulink model determines T_8 (for example, ADC data request block on Simulink root: $T_8 =$ model sample time).

Task τ_9: $T_9 = T_8$ the period time of τ_9 behaves like the period time of τ_8

Task τ_{10}: T_10 depends on the trigger frequency of the ultrasonic sensor. It is triggered by a GPIO pin. The trigger period is defined by a square wave signal as shown in figure 5.49.

Task τ_{11}: $T_{11} = T_{10}$ the period time of τ_{11} behaves like the period time of τ_{10}

In figure 4.4 the worst case execution time C_i of the tasks are visualized.

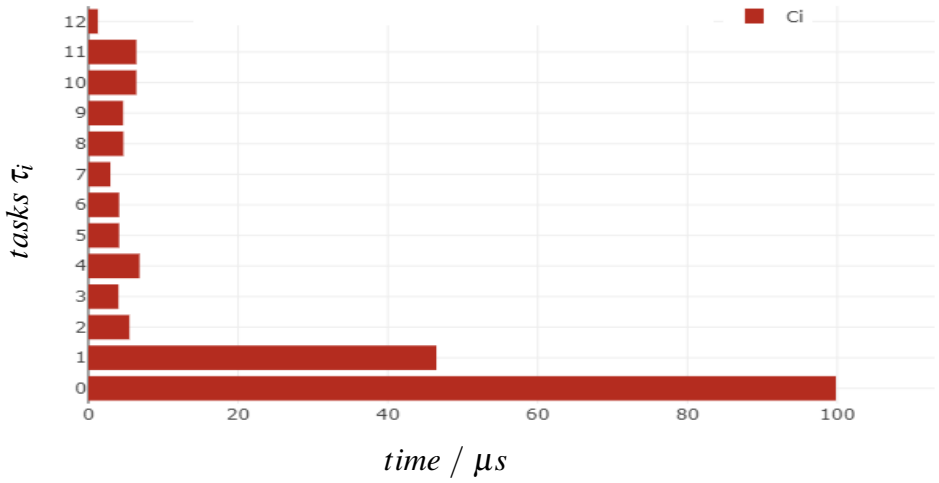


Figure 4.4: Worst case execution time C_i of the required tasks τ_i

When planning the scheduling of the real-time tasks, it is assumed that the release time r_i, is negligibly small. The calculated utilizations u_i, as well as the total utilization U_{sum}, are calculated in table 4.9 for RM and DM. The deadlines used for the calculation of the total utilization U_{sum} of DM are taken from the requirements (Req_05, Req_06, Req_07, Req_08, Req_09 A.1.1). In the calculation of the total utilization U_{sum} of RM, equation (2.1) applies to the task utilizations u_i and equation (2.3) to U_{sum}. For DM, the task utilizations are calculated by C_i/T_i, and U_{sum} is calculated by equation (2.6).

Task no. i	T_i / μs	C_i / μs	D_i / μs	RM u_i	DM u_i
0	1000	99.818	500	0.099818	0.19964
1	997.47	46.467	500	0.046492	0.092934
2	997.62	5.4812	500	0.0054943	0.010962
3	997.91	4.0167	500	0.0040251	0.008033
4	500	6.8316	100	0.013663	0.068316
5	501.57	4.1124	50	0.0081991	0.082248
6	501.57	4.1124	50	0.0081991	0.082248
7	1000000	2.9357	1000	2.936E-06	0.002936
8	1000000	4.6819	1000	4.682E-06	0.004682
9	1000000	4.6437	1000	4.644E-06	0.004644
10	100000	6.4102	100	6.41E-05	0.064102
11	100000	6.4102	100	6.41E-05	0.064102
12	8907.7	1.2778	1000	0.0001434	0.001278
$U_{sum} \approx$				0.186	0.686

Table 4.9: Calculation of task and total utilization for RM and DM

The maximum permissible total utilization U_{sum} for $n = 13$ tasks is calculated according to equation (2.3) in equation (4.1).

$$U_{sum} \leq 13(2^{1/13} - 1) \approx 0.712 \tag{4.1}$$

The total utilizations U_{sum} calculated in table 4.9 are smaller than the value resulting from equation (4.1). Therefore, they are permissible. Since deadlines are defined in the requirements, DM is applied. The priority is now assigned according to the size of the deadline D_i of the task, described in section 2.3. If the deadlines are the same, the task with the higher utilization is prioritized. The lowest assigned priority value has the highest priority. Tasks that are called via the same interrupt or that have the same deadline get the same interrupt priority.
The assignment of task priorities is shown in table 4.10.

Task no. i	Priority value
0	4
1	5
2	6
3	7
4	2
5	1
6	1
7	9
8	8
9	8
10	3
11	3
12	9

Table 4.10: Task priority assignment

The priority values of table 4.10 specify the interrupt priorities which are inserted into the interrupt blocks during model development 5.7.

Design of the non real-time application

When designing the non real-time application, the following points must be observed:

- The XCP messages must still be passed to the Application.
- A graphical application must be implemented that allows the user to monitor and tune model parameters.

During the software design 4 tasks are identified:

Task 1: Graphical application and sending set model parameters to the Cortex-M4.

Task 2: Receiving model parameters intended for display in the graphical application.

Task 3: Receiving XCP messages from the Cortex-M4 and forwarding them via TCP/IP to the development computer.

Task 4: Receiving XCP messages from the development computer and forwarding them to the Cortex-M4.

An overview of the planned implementation is shown in figure 4.5.

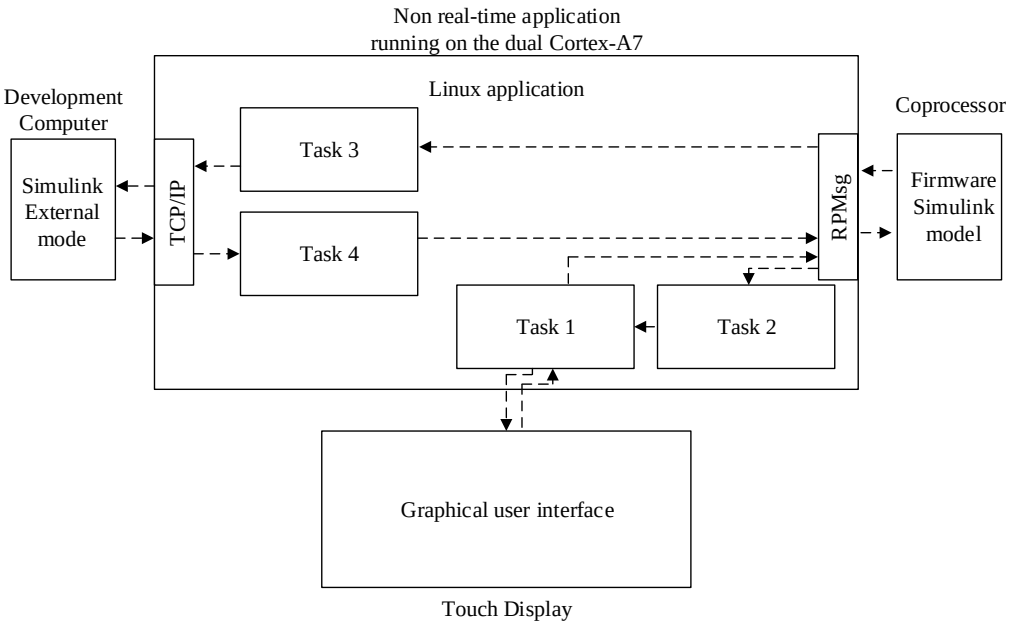


Figure 4.5: Schematic diagram of the non real-time application planning

It is planned to execute the task of the graphical application periodically within an idle loop and to distribute the remaining 3 tasks to threads.

5 Software implementation

The software implementation chapter describes the implementation of the Simulink coder target, the external mode via XCP on TCP/IP, the Simulink blocks, the real-time firmware, the non-real-time application, and the build process of the real-time firmware.

5.1 Customization of a Simulink target for the Cortex-M4

As specified in the technical requirement Req_01 A.1.1, the code executed on the Cortex-M4 must be generated from a Simulink model using the embedded coder [45]. Some platforms are supported by board support packages, like the STM Discovery Boards [46], the NXP S32K1 series [47], the Texas Instruments C2000 [48] and the Raspberry Pi [49]. To give a few examples.
Such a board support package is not available by the time this master thesis is started. It is still possible to generate code for unsupported platforms by customizing the code generation process in Simulink. [31]
To understand the code generation process, it is necessary to take a closer look at the used components.

Implementation of the file customization templat

The STF that has to be selected, according to A.1.1, is the "Embedded coder" `ert.tlc` file. To customize the code generation process of the "Embedded coder" a file customization template is developed.
The file customization template is selected by the "File customization template" option under "Code generation/Templates". Figure 5.1 shows that the default file customiza-

tion template `example_file_process.tlc` is selected. In figure 5.1 the option "Generate an example main program" is selected. The main program is generated by associated TLC files of the option "Target operationg system". [31]

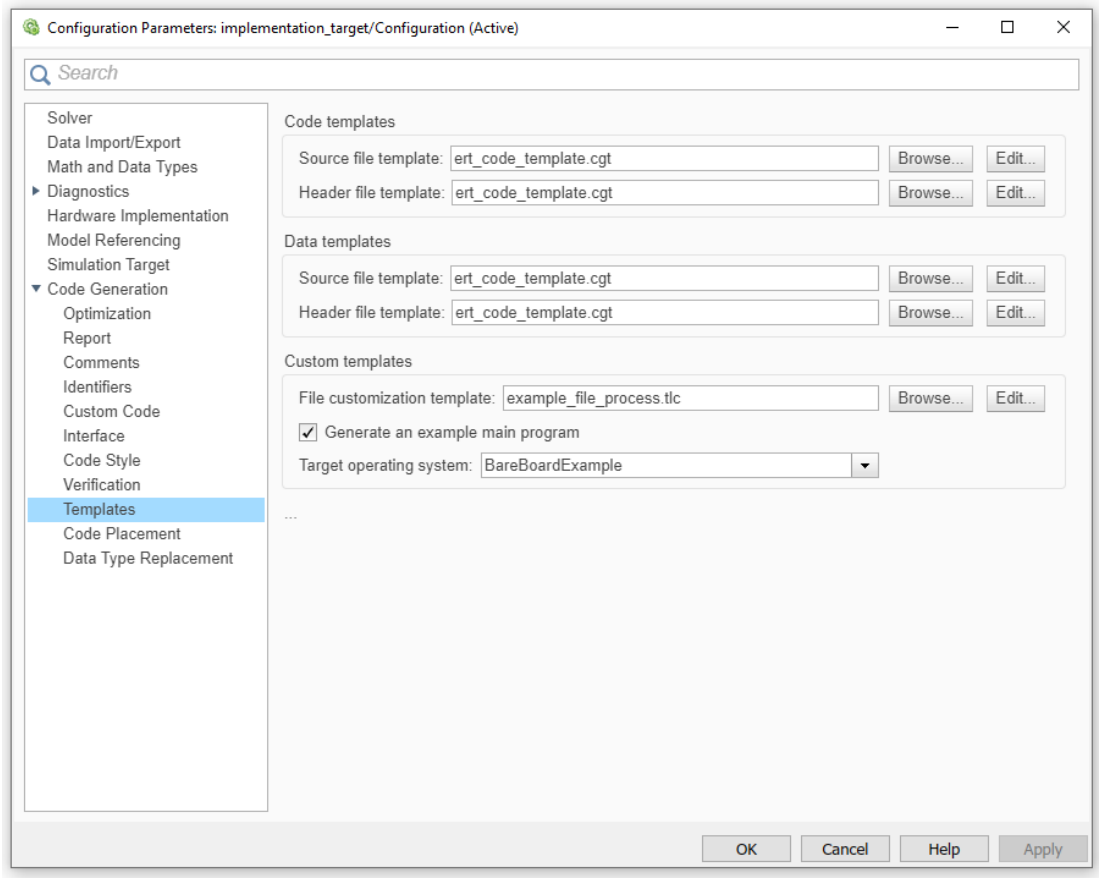


Figure 5.1: Simulink code generation settings for file customization template and the option to generate a main program

To write a file customization template, the `example_file_process.tlc` [50] is inspected.

The abstract of the file indicates, that it is an embedded coder sample file, used to supplement the generated source code and create additional files. The TLC code of the `example_file_process.tlc` file creates, if the variable `ERTCustomFileTest` has the value `TLC_TRUE`, a `timestwo.c` and a `timestwo.h` file. Then a `#define` is added to the public header of the model and another `#define` is added to the private headers of the model. Then

bareboard_srmain.tlc is included. The example_file_process.tlc file shows an example of how files can be created by the STF, how code can be inserted into generated files, and that a TLC file can include other TLC files. If the variable ERTCustomFileTest does not have the value TLC_TRUE, the example_file_process.tlc does not affect the code generation process. By default, the ERTCustomFileTest variable is commented out, so the example_file_process.tlc has no effect on the standard code generation process. [50]

To determine which file customization templates are already available in Simulink, the Matlab root directory is scanned. The file customization templates also called coder targets, are shown in listing 5.1.

```
1 codertarget_bareboard.tlc
2 codertarget_file_process.tlc
3 codertarget_mainwithoutOS.tlc
4 codertarget_multiratemultitasking.tlc
5 codertarget_multiratesingletasking.tlc
6 codertarget_singleratesingletasking.tlc
```

Listing 5.1: Existing file customization templates from the MATLAB root directory [51]

After inspecting the existing file customization templates, a custom file customization templates is developed especially for the Cortex-M4 processor of the STM32MP1.

The file customization templates codertarget_mainwithoutOS.tlc [52] is the closest to the file customization templates that has to to be developed. There are also significant differences.

Main differences of the codertarget_mainwithoutOS.tlc [52] and the custom file customization templates being developed:

- The codertarget_mainwithoutOS.tlc [52] generates a main function.
- The custom file customization templates must not generate a main function, because the main function is generated from STM32CubeMX.
- The scheduling of the codertarget_mainwithoutOS.tlc [52] is based on the fact that the timing of the model step is based on the system clock of the target.

- The scheduling of the custom file customization templates model step must be based on a timer interrupt call.
- The `codertarget_mainwithoutOS.tlc` [52] supports targes that have a scheduler and targets that do not have a scheduler.
- The custom file customization templates should only support the STM32MP1 Cortex-M4 running without a scheduler.
- The `codertarget_mainwithoutOS.tlc` [52] supports the use of a bootloader background task.
- The custom file customization templates shall not support a bootloader background task. Instead of a bootloader, the initialization of the hardware of the STM32MP1 Cortex-M4 shall be initialized by the initialization functions of the C-project generated by STM32CubeMX.

Main commonalities of the two file customization templates:

- Both file customization templates call the model step function at predefined intervals. Defined at the Solver settings under *"Periodic sample time constraint"*
- Both file customization templates contain initialization routines. This means the initialization of the Simulink blocks, as well as the timing configuration of the model step call.
- Both file customization templates contain functions that integrate the external mode.

Before presenting snippets of the resulting custom file customization templates, the model call from the `main` is shown in figure 5.2. The C code shown corresponds to a summary of the code that is generated during the generation process.

The `main`, shown on the left side in figure 5.2, starts with the reset of all peripherals, the initialization of the flash interface, and the initialization of the system clock. After that, peripheral devices like GPIOs and DMAs are initialized. After the generated initialization routines of STM32CubeMX are finished, the function `start_model_Task`, which is framed in blue, is called. In the function `start_model_Task`, seen on the right side in the `simulink_model_call.c` file, the initialization of the external mode is done at the beginning. This is followed by the initialization of the model. Next, the final time of the external mode is configured by the function `rtSetTFinalForExtMode` and the initialization of the external mode is checked by the function `rtExtModeCheckInit`. Subsequently, the

start_model_Task function is paused in the rtExtModeWaitForStartPkt function until the request of the development computer to start the model is received. This pause can be skipped by a define, set in the build process. If a connection error occurs during communication with the development computer in the rtExtModeWaitForStartPkt function, the variable rtmStopReq is set to 1, and the execution of the model is interrupted using the rtmSetStopRequested function. If this is not the case, a start message is sent to the development computer by the function rtERTExtModeStartMsg. The described functions of the simulink_model_call.c file are taken from the code generation using the codertarget_mainwithoutOS.tlc [52].

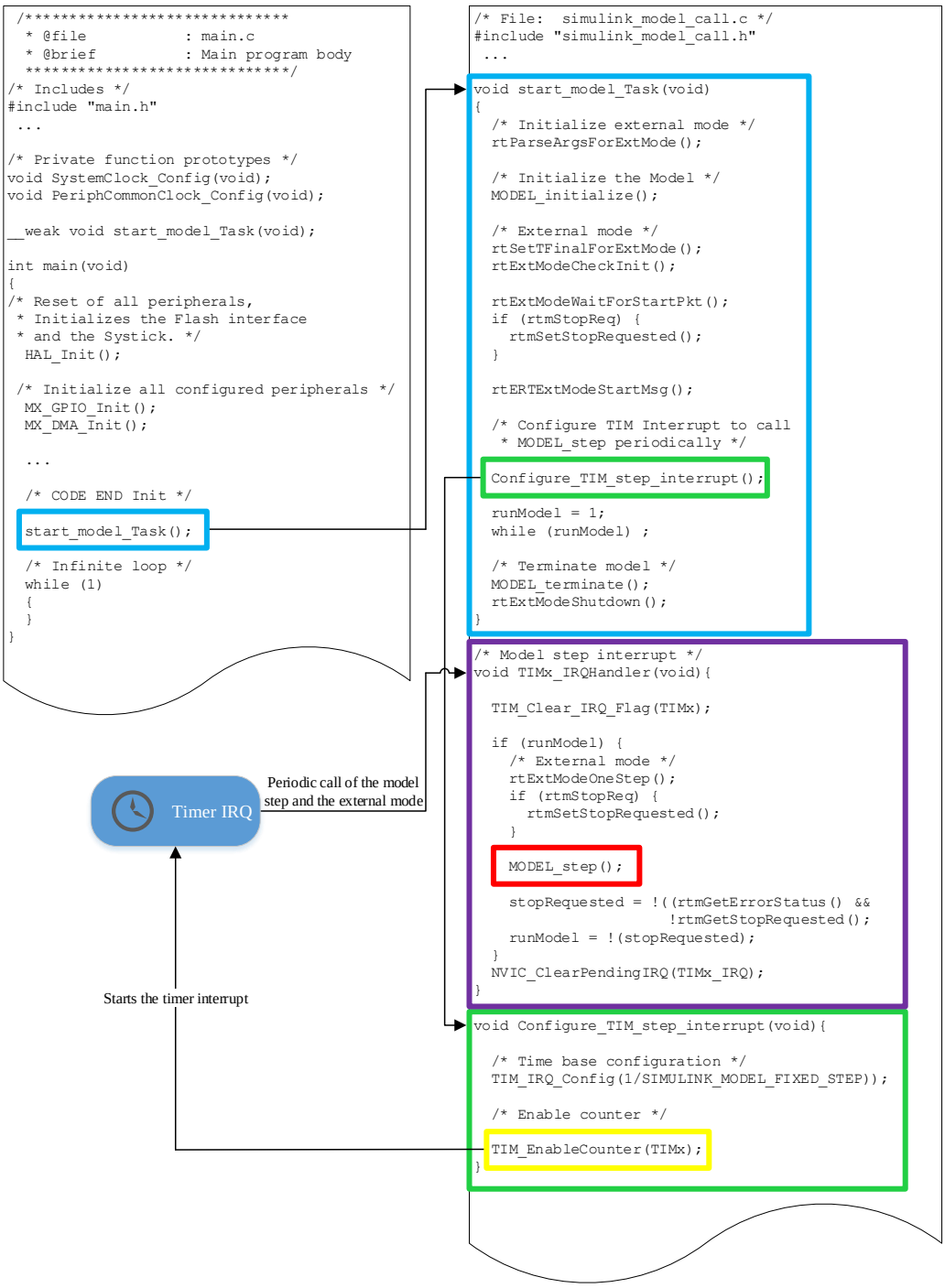


Figure 5.2: Illustration how the generated model is called from the STM32CubeMX C-project

Then the green framed function `Configure_TIM_step_interrupt` is called. This function configures and starts the timer interrupt. If the timer was started there are timer IRQ occuring periodically. These IRQ call the corresponding interrupt service routine `TIMx_IRQHandler`. It is framed in purple. Within the ISR, the Timer IRQ flag is cleared by the macro `TIM_Clear_IRQ_Flag`. If the variable `runModel` has the value 1 the function `rtExtModeOneStep` is called first. Within this function, the XCP background task is executed. If the variable `rtmStopReq` was set, the function `rtmSetStopRequested` is executed. The next function `MODEL_step`, framed in red, calls the Simulink model step. After the Simulink model step has been executed, the functions `rtmGetErrorStatus` and `rtmGetStopRequested` are used to check whether a stop request is present. If a stop request is present, the variable `runModel` is set to 0. Afterward, the interrupt flag of the NVIC is cleared by the macro `NVIC_ClearPendingIRQ`.
After the function `Configure_TIM_step_interrupt` was called the variable `runModel` is set to 1. And the processor remains in the while loop as long as the variable `runModel` is not reset. This while loop is periodically interrupted by the timer interrupt, which calls the external mode and the model step. If the variable `runModel` is set to 0 within the timer ISR the functions `MODEL_terminate` and `rtExtModeShutdown` are called. `MODEL_terminate` contains deinitialization routines of Simulink blocks or memory frees. The function `rtExtModeShutdown` sends a message about the shutdown to the development computer and cleans up the allocated memory of the external mode. This represents the basic process of a firmware generated by the embedded coder and the developed coder target.

For the more detailed description of the developed coder target, some TLC directives must be known.
The first non-empty character of a TLC directive must be a percent sign. For example, a TLC variable is declared as shown in the first line of listing 5.2. If a TLC variable should be used as an expression then the variable has to be enclosed in %<>, seen in listing 5.2. [53]

```
%assign string = "Hello World"
%assign expr =
/* Print out
printf("%<expr>");
```

Listing 5.2: TLC variable declarations and use of expressions

For Example the TLC generates from listing 5.2 the code shown in listing 5.3.

```
/* Print out Hello World */
printf("Hello World");
```

Listing 5.3: Generated c code from listing 5.2

Two percent signs mark a single line comment. [53]

```
%% Comment
```

Listing 5.4: TLC single line commands

Single and multi line commands can also be written as seen in listing 5.5. [53]

```
/% Comment
% Also comment %/
```

Listing 5.5: TLC single or multi line command

MATLAB functions are called as seen in listing 5.6. [53]

```
%matlab plot(x,y)
```

Listing 5.6: Using MATLAB functions in TLC files

If conditions can be used for example as shown in listing 5.7. In listing 5.7 the TLC function `ISEQAL` is used. This function checks, in this example, whether the parameter *i* has the value 1.0. [53]

```
%if ISEQUAL(i, 1.0)
/* If i has the value 1.0 this comment is transferred into the
    code */
%endif
```

Listing 5.7: Using if conditions in TLC files

The custom codertarget is called `codertarget_STM32MP1.tlc`. It starts with a comment header. Then several TLC variables are created. The most important ones can be seen in listing 5.8.

```
1 %assign srcBaseName = LibGetMdlSrcBaseName()
2 %assign MODELBASERATE = CompiledModel.SampleTime[0].
    ClockTickStepSize
```

Listing 5.8: Declaration and initialization of TLC variables.

The function `LibGetMdlSrcBaseName` returns the Simulink model name. [54]

The name of the simulink model is stored in the variable `srcBaseName`. The model sample time is transferred from variable `CompiledModel.SampleTime[0].ClockTickStepSize` to variable `MODELBASERATE`.

The C file `simulink_model_call.c` from figure 5.2, that is to be generated by the TLC, needs a header file to include headers, defines and function prototypes. The creation of this header can be seen in listing 5.9.

```
1  %% Create simulink_model_call.h
2  %assign simulink_model_call_hdr_file = LibCreateSourceFile(
       "Header", "Custom", "simulink_model_call")
3  %openfile hdr_includes
4  %% Including header files
5  /* Including headers */
6  #include "
7  #include "
8  #include "stm32mp1xx_it.h"
9  #include "tim.h"
10 %closefile hdr_includes
11 %openfile hdr_declarations
12 %% Declarate model defines and function declarations
13 /* Model fixed step */
14 #define SIMULINK_MODEL_FIXED_STEP %<MODELBASERATE>
15 /* Function prototypes */
16 void start_model_Task(void);
17 void TIM7_IRQHandler(void);
18 void Configure_TIM7_step_interrupt(void);
19 %closefile hdr_declarations
20 %<LibSetSourceFileSection(simulink_model_call_hdr_file,
     "Includes", hdr_includes)>
21 %<LibSetSourceFileSection(simulink_model_call_hdr_file,
     "Declarations", hdr_declarations)>
```

Listing 5.9: Creation of the `simulink_model_call.h`

The `LibCreateSourceFile(type, creator, name)` function creates a new C or C++ file. If the file already exists, the existing file is referenced. Allowed parameters for the input parameter `type` of the function are "`Header`" or "`Source`". This parameter determines the file extension (`*.c` or `*.h`). The input parameter `creator` specifies who creates this file. The input parameter `name` specifies the name of the file to be created. [54]

The commands `%openfile` and `%closefile` create a buffer. This buffer stores the lines between the command `%openfile` and `%closefile`. [53]

The `LibSetSourceFileSection(fileH, section, value)` function in-

serts text buffers into a specified section of a file. The input parameter `fileH` determines into which file the text buffer is inserted. The input parameter `section` determines in which section of the file the text buffer will be inserted. Sections are for example: "`Includes`" and "`Defines`". The input parameter `value` specifies the text buffer that will be inserted. [54]

The TLC generates the code shown in listing 5.10 when first listing 5.8 and then listing 5.9 are specified in a TLC-file and a Simulink model named `test.slx` is opend.

```
1  #ifndef RTW_HEADER_simulink_model_call_h_
2  #define RTW_HEADER_simulink_model_call_h_
3  /* Including headers */
4  #include "test.h"
5  #include "test_private.h"
6  #include "stm32mp1xx_it.h"
7  #include "tim.h"
8  /* Model fixed step */
9  #define SIMULINK_MODEL_FIXED_STEP       0.001
10 /* Function declarations */
11 void start_model_Task(void);
12 void TIM7_IRQHandler(void);
13 void Configure_TIM7_step_interrupt(void);
14 #endif
15 /* RTW_HEADER_simulink_model_call_h_ */
```

Listing 5.10: `simulink_model_call.h` file generated by the TLC

The source file `simulink_model_call.c` is generated in the same way as the corresponding header. Therefore, in the following only the generation of the C functions known from figure 5.2 is described. The complete `codertarget_STM32MP1.tlc` is available in the A.1.2.

Listing 5.11 shows the implementation of the `start_model_Task` function, by the use of the `codertarget_STM32MP1.tlc` file.

```
%% Create simulink_model_call.c
...
void start_model_Task(void){
    %<RTMSetErrStat(0)>;
    %if ExtMode
        /* Initialize external mode */
        rtParseArgsForExtMode(0, NULL);
        %endif
    /* Initialize the Model */
    %<srcBaseName>_initialize();
    %if ExtMode
        %<SLibGenERTExtModeInit()>
    %endif
    %if ISEQUAL(CompiledModel.SuppressErrorStatus,0)
        runModel = %<ERTStopCheck()>;
    %endif
    /* Configuration of the TIM7 interrupt */
    Configure_TIM7_step_interrupt();
    /* Idle while loop*/
    while(runModel);
    /* Termination */
    %<srcBaseName>_terminate();
    %if ExtMode
        rtExtModeShutdown(%<NumSynchronousSampleTimes>);
    %endif
}
```

Listing 5.11: TLC implementation of the function `start_model_Task`, according to [52]

In listing 5.11 it is seen that in the function `simulink_model_Task` at the beginning the expression `RTMSetErrStat` is added. Subsequently, if the TLC variable `ExtMode` has the value `TRUE`, the parsing of the arguments for the external mode is performed. Afterwards the mode is initialized. If the external mode is activated, then the initialization of the external mode is carried out. If the variable `CompiledMode.SupressErrorStatus` has the value 0, a function is called, which checks the model status. Subsequently, the configuration of the timer interrupt

is executed.

During the implementation, timer 7 was chosen for the implementation of the model step, this decision was made during the hardware peripheral configuration in section 5.6.

Afterwards the idle while loop is inserted into the C file. After the while loop the termination of the model and the shutdown of the external mode follows, if the external mode is configured.

Listing 5.12 shows to the configuration of the interrupt that calls the model step.

```
1  void Configure_TIM7_step_interrupt(void){
2          LL_TIM_InitTypeDef     timInitStruct;
3          /* Time base configuration */
4          timInitStruct.Prescaler          = __LL_TIM_CALC_PSC(
               SystemCoreClock, (1/SIMULINK_MODEL_FIXED_STEP)
               *10000);
5          timInitStruct.CounterMode        =
               LL_TIM_COUNTERMODE_UP;
6          timInitStruct.Autoreload         = __LL_TIM_CALC_ARR(
               SystemCoreClock, tim_initstruct.Prescaler, (1/
               SIMULINK_MODEL_FIXED_STEP));
7          timInitStruct.ClockDivision      =
               LL_TIM_CLOCKDIVISION_DIV1;
8      /* Initialization of the TIM7 peripheral */
9          LL_TIM_Init(TIM7, &tim_initstruct);
10         /* Enable TIM7 interrupt */
11         LL_TIM_EnableIT_UPDATE(TIM7);
12         /* Start TIM7 counter */
13         LL_TIM_EnableCounter(TIM7);
14 }
```

Listing 5.12: TLC implementation of the function/ISR `TIM7_IRQHandle`

In listing 5.12 it can be seen that the following timer settings are done by the use of the `LL_TIM_Init` [55] function in line 9.

These settings are:

1. The calculated prescaler value from line 4, is set to the prescaler register, shown in figure A.1.

2. The counter mode, seen in line 5, is set to upcounting, by setting the DIR bit of the TIM control register 1, shown in figure A.6.
3. The calculated auto-reload value, seen in line 6, is set to the Auto-Reload register, shown in figure A.2.
4. The clock division is set to 1 in line 7, by setting the CKD bitfield of the TIM control register, seen in figure A.6.

Then the UIE Bit form TIM DMA/Interrupt Enable Register is set in line 11 to enable the timer interrupt. The register is seen in figure A.7. At least the timer is started in line 13, by setting the CEN bit in TIM control register 1, shown in figure A.6.

The c functions used in listing 5.12 are taken from [55]. The macros to calculate the values of the prescaler and the auto-reload in line 4 and 6 are taken from [56].

The macros calculate the prescaler according to equation (A.1), and the auto-reload value according to equation (A.2).

Listing 5.13 shows the model step that is processed by the ISR of the timer interrupt.

```
1  %% Create simulink_model_call.c
2  ...
3  /* Model Step Interrupt */
4  void TIM7_IRQHandler(void)
5  {
6          LL_TIM_ClearFlag_UPDATE(TIM7);
7          if (runModel){
8          %if ExtMode
9              %<FcnGenerateExtModeOneStep()>
10         %endif
11             %<srcBaseName>_step();
12         stopRequested = !(%<ERTStopCheck()>);
13             %if HONORRUNTIMESTOPREQUEST || ExtMode
14             runModel = !(stopRequested);
15         %endif
16         }
17 NVIC_ClearPendingIRQ(TIM7_IRQn);
18 }
```

Listing 5.13: TLC implementation of the function/ISR `TIM7_IRQHandle`

The processing of the timer ISR, which calls the model step, runs as follows:

Line 6: The macro `LL_TIM_ClearFlag_UPDATE` [55] resets the UIF bit of the timer status register, seen in figure A.8.

Line 7: Checks that variabl `runModel` is not equal to 0.

Line 8: TLC if the external mode is selected during code generation, the if condition in line 8 will insert line 9 in the code.

Line 9: (Only if ExtMode=1) Inserts the functions for calling the external mode, if `ExtMode` = 1. The MATLAB function `FcnGenerateExtModeOneStep` is taken from [52].

Line 10: End of the if conndition from line 8.

Line 11: Call of the model step. Here the generated code of the Simulink model is called.

Line 12: It is checked whether a stop request is present. The MATLAB function `ERTStopCheck` is taken from [52].

Line 13: TLC if condition inserts code if there is a runtime limited or if the external mode is enabled.

Line 14: The variable `runModel` is set to the value non `stopRequest`. `stopRequest` is set by the external mode one-step function or by the model step if the run-time limit is reached.

Line 15: End of the if conndition from line 13.

Line 17: Clears the pending flag of the NVIC. The macro `NVIC_ClearPendingIRQ` is taken from [57].

To generate code, the system target file of the embedded coder `ert.tlc` is selected in the Simulink “Hardware Settings” under “Code Generation”. The option “Generate code only” is selected, too. This is seen in figure 5.3.

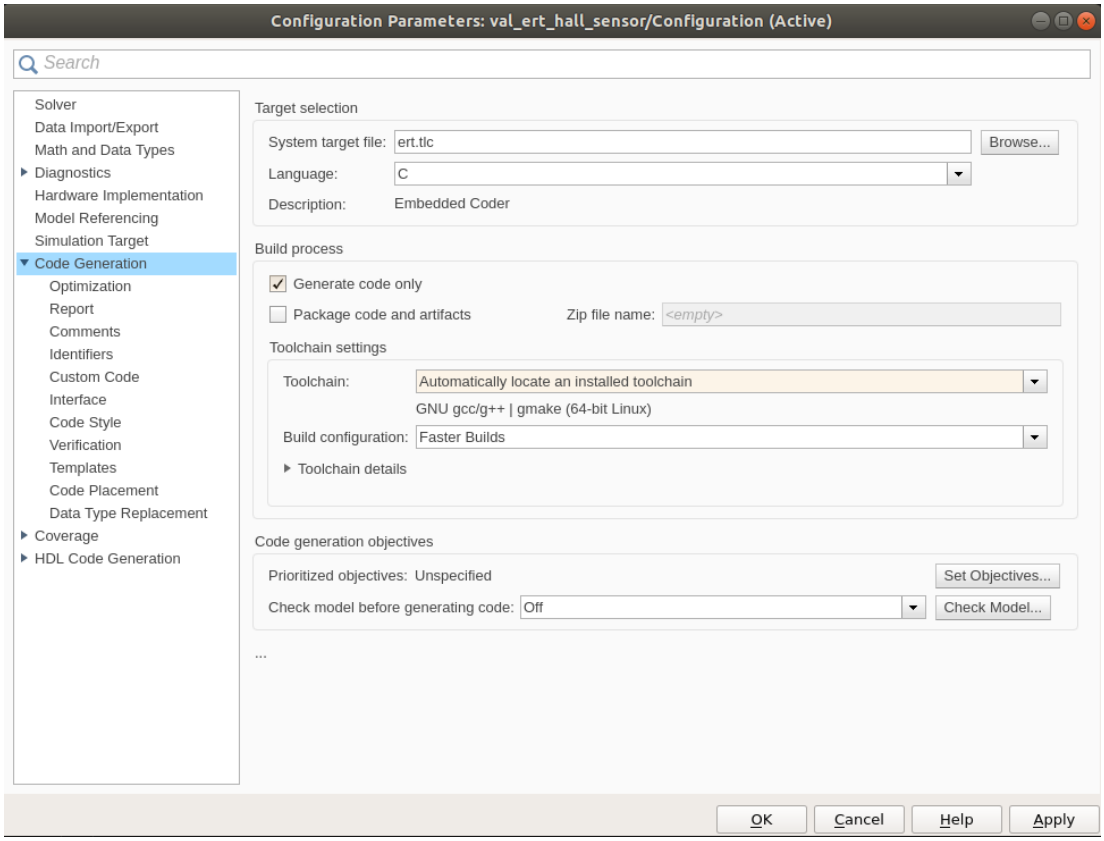


Figure 5.3: Selecting the system target file for code generation

The developed “File customization template” `codertarget_STM32MP1.tlc` is selected in the Simulink ‘Hardware Settings” under “Templates”. This is seen in figure 5.4.

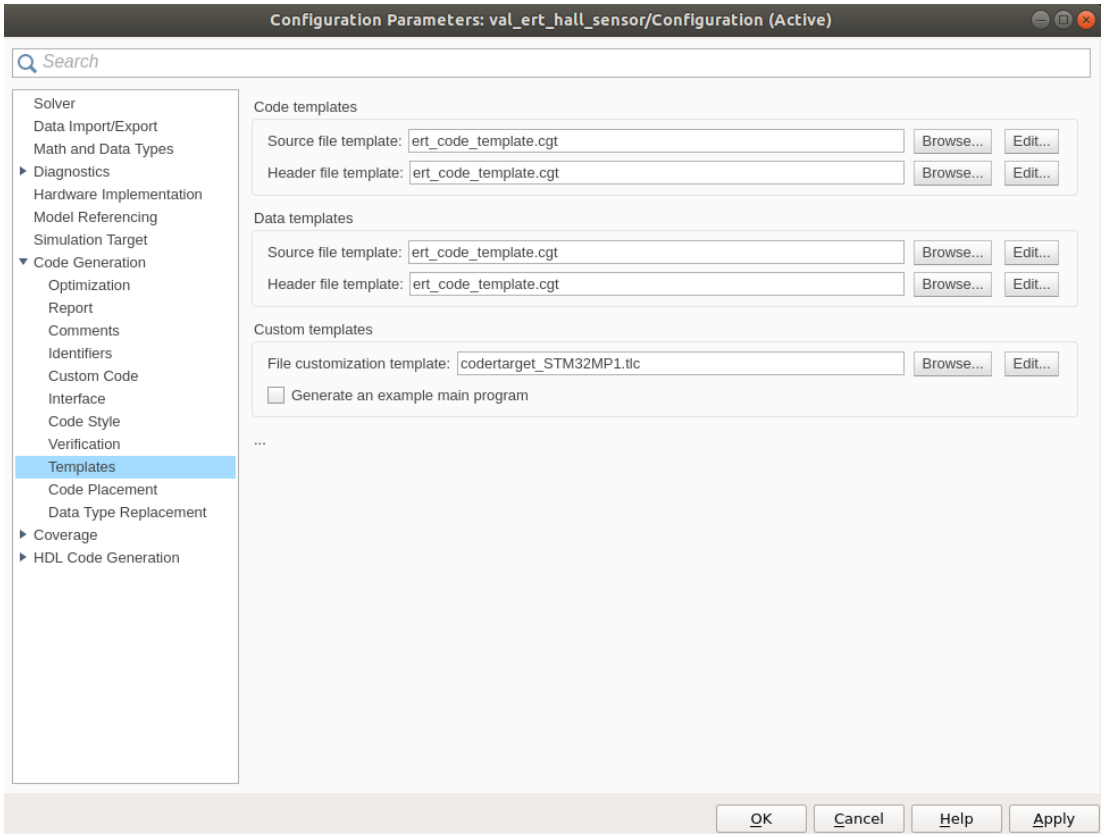


Figure 5.4: Selection of the developed coder target for the STM32MP1

After these settings have been made, the code can be generated.

5.2 Implementation of the External mode via XCP on TCP/IP

The external mode turns the Simulink model into a bidirectional interface to the real-time application generated from the Simulink model. The real-time application can be started, stopped, signal paths can be observed and parameters can be adjusted by the external mode. [58]

A schematic communication diagram of the external mode connection is shown in figure 5.5.

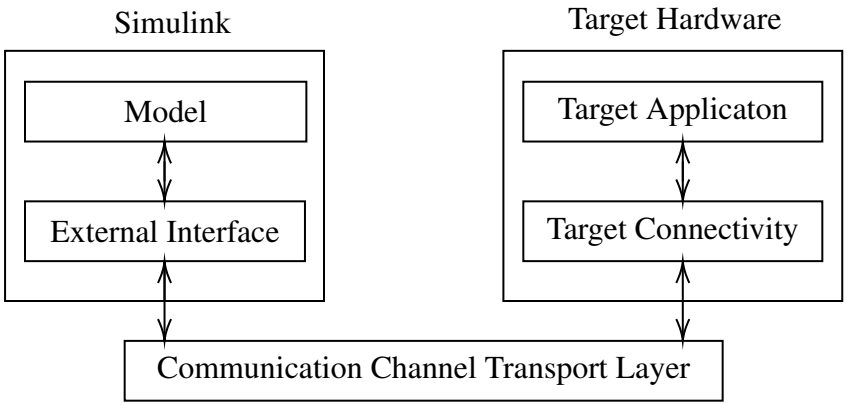


Figure 5.5: External mode transport layer between the development computer and the target hardware (cf. [59])

The external mode can be implemented through different transport layers such as XCP, TCP/IP or serial. [59]

The implementation of the different transport layers is based on the implementation of the rtiostream API. [42, 60]

The rtiostream API [61] is used to create a physically independent communication channel for the exchange of data between processors. The rtiostream API is based on four functions. The function `rtIOStreamOpen` is used to initialize the channel. The function `rtIOStreamSend` is used to send data over the interface and the function `rtIOStreamRecv` is used to receive data from the channel. The fourth function `rtIOStreamClose` deinitializes the communication channel opened by the first function. [61]

The implementation of the XCP brings the advantage of a more lightweight communication software stack on the real-time hardware side compared to the basic implementation of the external mode. [59]

Figure 5.6 shows a schematic diagram of the external mode via XCP.

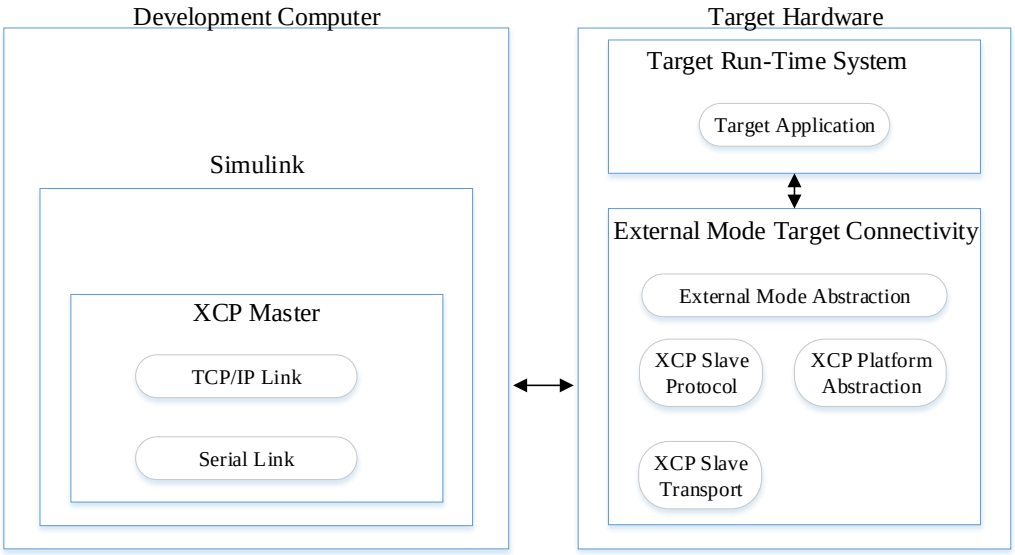


Figure 5.6: Schematic diagram of the external mode via XCP transport layers (cf. [42])

It can be seen that the XCP master (server) is located on the side of the development computer. This is different when using the external mode without XCP. In the external mode implementation without XCP there is an external mode server on the target hardware side. [62]
This external mode server on the target side can be transfomed to a XCP master running on the hardware of the development computer by implementing the external mode via XCP. [42, 62]
This is the reason, why the implementation of the external mode via XCP leads to a smaller communication software stack than the implementation of the external mode without XCP. [42, 59, 62]
In addition, the implementation of the external mode via XCP supports the monitoring of signals in the Simulation Data Inspector, and in the Logic Analyzer. These features are not supported by the basic external mode. [59]
Based on these reasons, it was determined during the requirements engineering process that the external mode must be implemented via XCP (Req_02 A.1.1).
The implementation of the external model via XCP is done in four steps: [42]

1. Adding the external mode functions to the Simulink target.

2. Adding the XCP slave protocol layer to the build process.
3. Adding the XCP slave transport layer to the build process
4. Creating an XCP platform abstraction layer

These four steps are explained next:

1. The adding of the external mode functions is already considered in section 5.1 during the development of the coder target. The external mode functions are included at the positions that are wrapped by the if condition:

```
1 %if ExtMode
2     externalModeFunctions();
3 %end
```

Listing 5.14: If ExtMode conndition, for the implementation of the external mode commands

2. Adding the XCP slave protocol layer according to ASAM MCD-1 XCP [28] standard to the build process. [42]
The source files are located at:
`matlabroot/toolbox/coder/xcp/src/target/slave/protocol/src`
These sources are added to the CMake project.

3. Adding the XCP slave transport layer according to ASAM MCD-1 XCP [28] standard to the build process. [42]
The source files are located at:
`matlabroot/toolbox/coder/xcp/src/target/slave/transport/src`
These sources are added to the CMake project.

4. The XCP platform abstraction layer consists of the XCP driver, which implements the physical communication channel and the implementation of static memory allocation as well as the implementation of other target hardware-specific functionalities, such as a delay implementation. [42]

For the implementation of the XCP driver, a XCP custom abstraction layer [63] and a rtiostream communication channel based on the rtiostream API [64] are created.

The XCP custom abstraction layer defines platform-specific functionalities, like mutual exclusion, a sleep function, data logging, address conversion, set memory, and copy memory. [42]

The developed XCP custom abstraction layer can be found at item A.1.2:
`Smart_RCP/02_Software/target/arch/STM32MP1_Source/Xcp/xcp_inc/xcp_platform_custom.h`

Since in section 4.1 it is planned to transfer the external mode XCP messages between the Cortex-M4 and the development computer through the Cortex-A7, a rtiostream layer is developed to provide a physical connection between the Cortex-M4 and the Cortex-A7.
The developed rtiostream communications channel can be found at item A.1.2:
`Smart_RCP/02_Software/target/arch/STM32MP1_Source/Smart_STM32MP1_rtiostream`
The rtiostream communications channel is implemented using the IPCC and the Open Asymmetric Multi Processing (OpenAMP) Framework on the Cortex-M4 side of the Inter-Processor Communication (IPC).
Figure 5.7 shows an overview of the IPC.

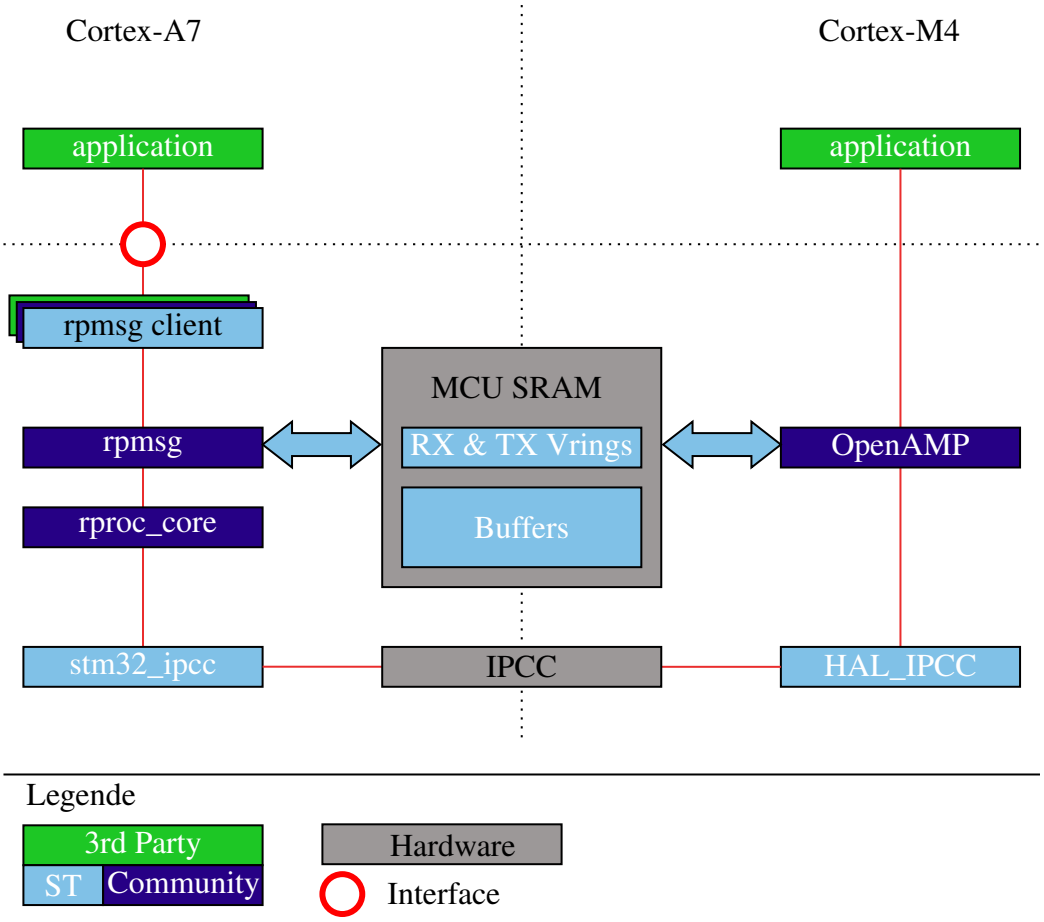


Figure 5.7: IPC structure (cf. [65])

The implementation of the IPC is based on the Remote Processor Messaging (RPMsg) and Mailbox mechanisms. [66]

As show on the Cortex-M4 side in figure 5.7, the IPCC and the OpenAMP framework is needed for the implementation. The required configuration for these components can be seen in figure A.20 and figure A.21.

There are two different ways to implement buffer exchange between the Cortex-A7 and the Cortex-M4. The first way is the “Direct buffer exchange mode”. In this mode, only effective user data is transferred by RPMsg buffer during data transfer. The memory allocation is hard defined in the code (default = 512 B). This mode can be implemented with small effort. The mode transmits data with a maximum speed of 5 MB s^{-1}. Each transmitted message triggers an interrupt at the receiving processor by the IPCC. For

example, a 512 B RPMsg transmitted at 1 MB s^{-1} would trigger about 2000 IRQs per second. The second way is the "Indirect buffer exchange mode". This mode uses RPMsgs to pass references to the effective data buffers. These data buffers can be of any size. The data access can be done on cached or none cached memory, Double Data Rate (DDR) or MicroController Unit (MCU) SRAM, DMA, or any master peripheral. This method requires an increased implementation effort compared to the first mode. [67]

For the implementation of the rtiostream communication channel the "Direct buffer exchange mode" is used. The Cortex-M4 sided implementation is similar to the Code example `OpenAMP_TTY_echo` [68].

If a Cortex-M4 interrupt is triggered by the IPCC, the received message is stored in a ring buffer using the `VIRT_UART0_RxCpltCallback` [69] during the ISR. This ring buffer is read out within the `rtiostreamRecv` function. The message is sent using the function `VIRT_UART_Transmit` [69] within the `rtiostreamSend` function.

On the Cortex-A7 side the IPC is implemented by the Linux remoteproc framework [70] and the mailboxservice stm32_ipcc [71]. [66]

For the Cortex-A7, an application is implemented that forwards the external mode XCP messages as shown in figure 5.8.

Within the Linux application the file discriptor `/dev/ttyRPMSG0` is opened by the system call `open` [72].

Opening the file descriptor enables the exchange of RPMsgs through the `ioctl` [73] system call.

This process is implemented by the functions `copro_openTtyRpmsg`, `copro_readTtyRpmsg`, and `copro_writeTtyRpmsg` of the `copro.c` [74] file. The function `rtIOStreamOpen` of the file `rtiostream_tcpip.c` [75] implements the TCP/IP server on the Cortex-A7. Two pthreads [76] are created to check if messages are received by the TCP/IP server or by the RPMsg framework. If thread 1 detects a new message on the TCP/IP server, it transfers it to the Cortex-M4 using the file discriptor and the RPMsg framework. When the thread 2 detects a message in the file discriptor, this message is sent to the development computer via the TCP/IP server.

Figure 5.8 shows a schematic diagram of the application that passes the XCP messages.

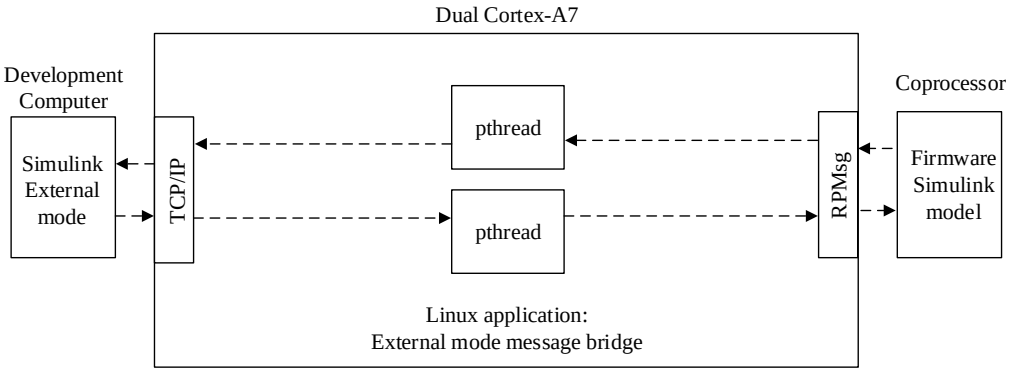


Figure 5.8: Linux application for bidirectional forwarding of XCP messages

This application, named “External_Mode”, is attached in the appendix under item A.1.2.

To use the external mode via XCP on TCP/IP, the “External mode” has to be selected in the Simulink “Hardware Settings” under “Interface”. The “Transport layer” has to be set to “XCP on TCP/IP”.

The “MEX-file arguments” consists of the IP address of the target, the port number and the path of the firmware. [77]

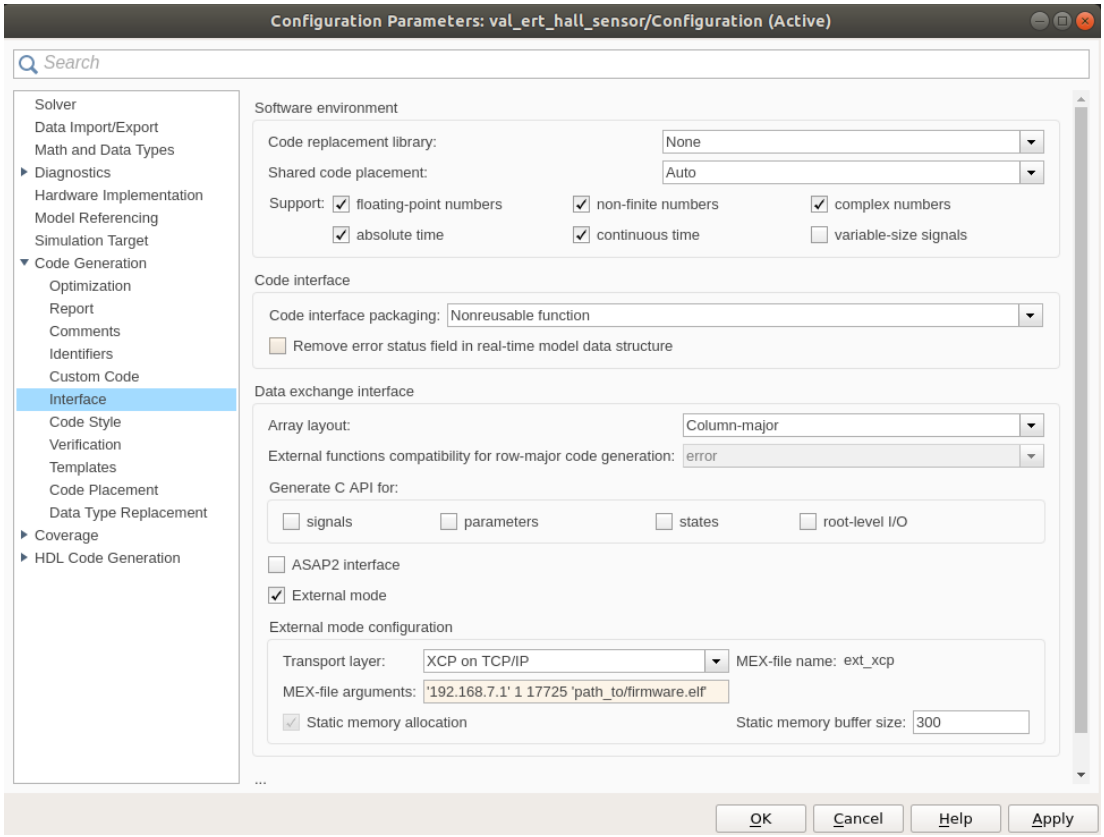


Figure 5.9: Setting the external mode via XCP on TCP/IP

Then first the generated firmware, and then the Linux application “External_mode” must be started on the STM32MP1.

Afterwards the external mode can be connected as described in [31].

5.3 Implementation of a MATLAB independent build process

As described in Req_03 A.1.1, a build process independent of MATLAB and independent of STM32CubeIDE [78] must be set up.
The build process implemented in [7] is based on a “Template Makefile”. A Makefile is generated from this during code generation. The Template Makefile is hardcoded. It knows only the C files, which are present at the point in time, when the Template Make-

file is created. The manual adjustment of the Template Makefile is time-consuming and inefficient.

The build process of STM32CubeIDE on the other hand is based on Eclipse C/C++ Development Toolkit (CDT). The CDT containing a C managed build, which generates a Makefile project. When using the STM32CubeIDE to generate a Makefile, the generated Makefile includes all files of the STM32CubeIDE Makefile project. [79]

This has the advantage that the Makefile does not have to be manually adjusted. The disadvantage is that the additional development tool ST32CubeIDE is necessary.

The implementation of CMake brings the advantage that a CMakeLists.txt is a blueprint for the generation of Makefiles, to compile the C project. This allows defining in which subdirectories code for creating objects, libraries, or applications are located. Thereby it is possible to build any source file after code generation from STM32CubeMX, which is located in the subdirectories described in the CMakeLists.txt. Another advantage is that it is possible to call the CMakeLists.txt from any directory path. [80]
This makes it possible to call the CMakeLists.txt file from the MATLAB project folder and to build the firmware inside the project folder, without having to copy the whole makefile project into the MATLAB project folder. The objects that are already compiled in the CMakeLists folder do not have to be recompiled when The CMakeLists.txt is called from a different path. In this CMakeList.txt project, the structure is divided into subdirectories. The build process of these subdirectories is described by their CMakeLists.txt files, which are included in the parent CMakeLists.txt file. This is mapped in figure 5.10.

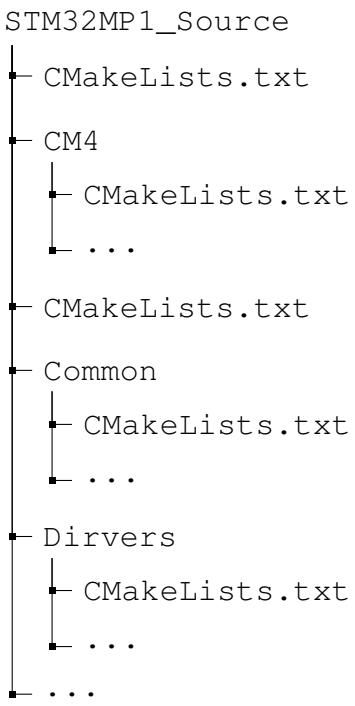


Figure 5.10: Structure of a STM32CubeMX prjectes

The build and deployment of the firmware on the STM32MP1, after the code generation by MATLAB Simulink, is done by the `build_and_deploy.bash` script A.1.2, that builds the firmware using the CMkaeLists.text and flashes the firmware to the SRAM memory of the Cortex-M4 by ssh and the help of the embedded Linux system.

5.4 Implementation of asynchronous Interrupts

The implementation of ISRs triggered by IRQs requires the configuration of the according interrupt, the configuration of the peripheral where the interrupt occurs and the programming of the according ISR. [81]
Since the generation of the C code for the Cortex-M4 has to be done by MATLAB/Simulink, all this has to be done by placing code in blocks.

An example code initialisation of an IRQ and an ISR for an DMA stream can be seen in listing 5.15 and listing 5.16.

```
main(){
    ...
    /* Configure NVIC */
    NVIC_SetPriority(DMA2_Stream2_IRQn, 0, 1, 0));
    NVIC_EnableIRQ(DMA2_Stream2_IRQn);
    /* Configure DMA peripheral */
    ...
    LL_DMA_EnableIT_TC(DMA2, LL_DMA_STREAM_2);
    ...
    while(1){
    }
}
```

Listing 5.15: Code example for the configuration of an interrupt

In listing 5.15 it can be seen that at the beginning the interrupt priority is set. Then the corresponding interrupt is enabled in the NVIC. Then the corresponding peripheral is configured and the interrupt is activated. From the activation of the interrupt enable flag in the register of the peripheral the interrupt is active and can lead to an ISR by the presence of an IRQ.

```
/* Interrupt Service Routine of DMA2_Stream2 */
void DMA2_Stream2_IRQHandler(void){
    /* start DMA Transfer */
    ...
    if (LL_DMA_IsActiveFlag_TC2(DMA2)) {
        LL_DMA_ClearFlag_TC2(DMA2);
        ...
    }
    NVIC_ClearPendingIRQ(DMA2_Stream2_IRQn);
    ...
}
```

Listing 5.16: Code example for the implementation of an ISR

Listing 5.16 shows the ISR, which is called by the IRQ. It is important that the corresponding register bits (peripheral and NVIC) that led to the call of the ISR are reset in the ISR. If these bits are not reset, then this can lead to a non-terminating call of the ISR.

MATLAB/Simulink supports the implementation of interrupt blocks. If the code generation is done by embedded coder, it is possible to include interrupts of the NVIC. If the board that is used is not supported by a board support package, a board specified interrupt block can be created by modifying the interrupt block contained in the "Support Package for ARM Cortex-M Processors". [82]

How to create an interrupts block for an ARM Cortex-M target is explained in the following:

Installation of the embedded coder Support Package for ARM Cortex-M processors. [82] The "Embedded Coder Support Package for ARM Cortex-M Processors" can be downloaded from the MathWorks file exchange.[83]

Creating an xml interrupt description file based on the numbers and the names of the interrupts of the interrupt vector table of the silicon vendor. An example xml interrupt description file can be viewed after installing "Embedded Coder Support Package for ARM Cortex-M Processors" by the commands shown in listing 5.17. [82]

```
1 cd(fullfile(codertarget.arm_cortex_m.internal.getSpPkgRootDir,
      'registry', 'interrupts'));
2 edit('arm_cortex_m_interrupts.xml');
```

Listing 5.17: Open the example xml interrupt description file by entering the commands shown in the MATLAB console

Within the xml interrupt description file the interrupts can be divided into groups. The IrqName, as well as the IrqNumber must match the name of the interrupt and the position in the interrupt vector table of the board. These two parameters are used for code generation. The parameter NumberOfPriorityBits must be adapted to the specification of the silicon vendor. It is possible to disable the "Disable interrupt pre-emption" checkbox by the ShowPreemptionOption, this should be done if the board does not support interrupt pre-emption. [82]

Figure 5.11 shows the relationship between the xml interrupt description file and the interrupt block mask.

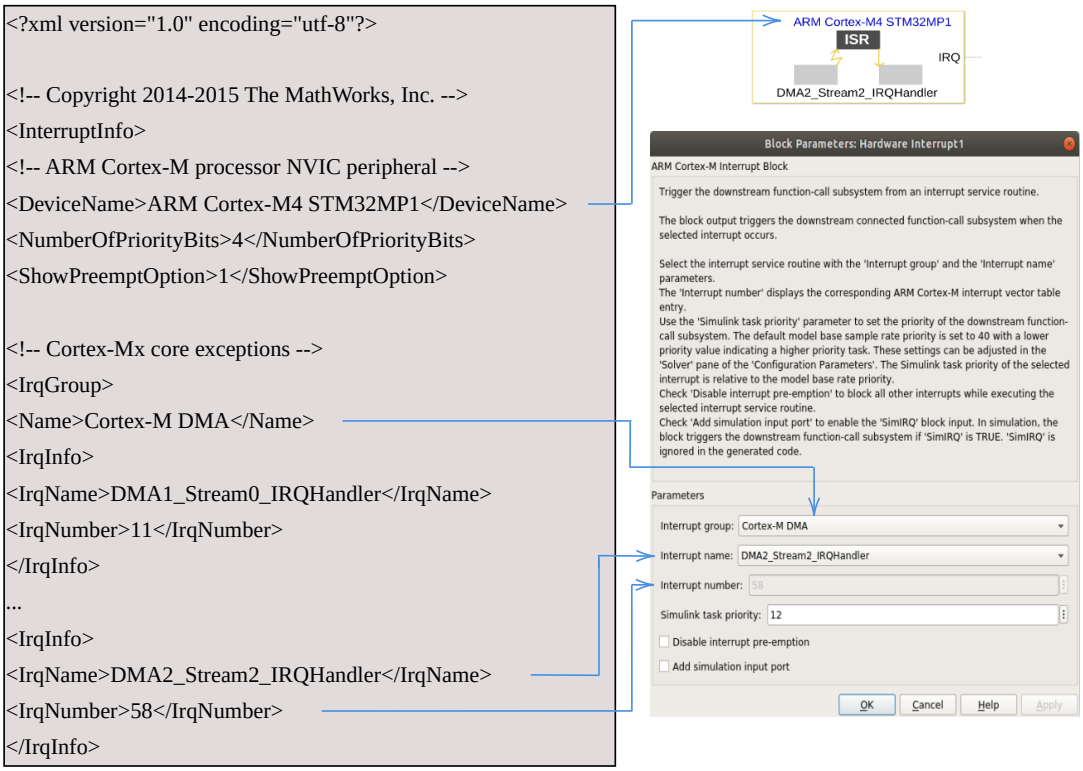


Figure 5.11: The xml interrupt description file displayed next to the interrupt block mask

Figure 5.11 shows a part of the xml interrupt description file on the left side, on the right side the interrupt block can be seen on top, below this the interrupt block mask is shown. The blue arrows symbolize the relationship between the xml interrupt description file and the block mask, as well as the block. The NumberOfPriorityBits is set to 4 for the Cortex-M processor of the STM32MP157. This can be read from the vendor specification in [24, p. 1266]. In the xml interrupt description file you can see that “ARM Cortex-M4 STM32MP1” was specified as `DeviceName`. This name is taken over together with the selected interrupt name in the block diagram. In the interrupt block mask the interrupt group “Cortex-M DMA” is currently selected, within this group the interrupt “DMA2_Stream2_IRQHandler” was selected under Interrupt name. In the interrupt block mask it can be seen that the interrupt number is grayed out. The interrupt number and the interrupt name were defined in the xml interrupt description file. For the interrupt “DMA2_Stream2_IRQHandler” the position was specified from the interrupt vector table. The interrupt position and the interrupt name can be

found for the Cortex-M4 of the STM32MP157 board in [24, p. 1252]. In the xml interrupt description file you can see that the `IrqName` "DMA2_Stream2_IRQHandler" was assigned the `IrqNumber` 58, which corresponds to the interrupt position in the interrupt vector table.

After the xml interrupt description file has been created, it must be registered in the interrupt block. To do this, the Arm Cortex-M interrupt block is copied to a library model and then registered using the command below in listing 5.18. [82]

The xml interrupt description file is registered in the interrupt block by the command shown in listing 5.18.

```
1 set_param(hardware interrupt block, InterruptsXMLPath,
    interrupt descritpion file)
```

Listing 5.18: Registration of the xml interrupt descrtiption file into the interrupt block

The library model containing the hardware interrupt block is then saved. From the library model the hardware interrupt block can be copied into the application models. [82]

Deployment of an interrupt block in a Simulink model

When using interrupt blocks within Simulink models, two more block types are interesting.

Function-Call Subsystems

A function-call subsystems is a subsystem that is executed when a function-call event is pending at the control port. An aperiodically triggered function-call subsystems can be called zero times, multiple times or once during a model step. In order for the blocks within the function-call subsystems to be called aperiodically, the sample time of these blocks must be set to -1.[31]

An example function-call subsystem, named "DMA2_Stream2_IRQHandler Interrupt Service Routine" is shown in figure 5.12.

Rate Transition blocks

The Rate Transition block is designed to connect periodic and asynchronous signal paths. If the Rate Transition block is inserted between two blocks with

different sampling rates, the block automatically configures its input and output for the transition.[31]

An example rate transition block named “Rate_Transition” is shown in figure 5.12.

The modified interrupt block can be used in the Simulink model as shown in figure 5.12.

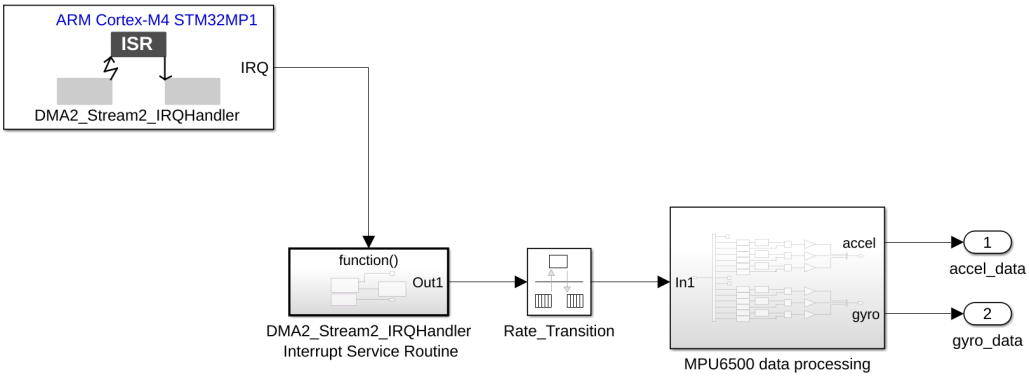


Figure 5.12: Deployment of an interrupt block in a Simulink model

The model shown in figure 5.12 shows the interrupt-based call to read the DMA buffer and process its data. The DMA buffer is read asynchronously within the function-call subsystem as soon as the DMA2 Stream2 IRQ is triggered. The read-in data is passed on to the “MPU6500 data processing subsystem” via the rate transition block. This subsystem is executed periodically.

If IRQs and ISRs are implemented by MATLAB/Simulink as described, configuration codes are generated for programming the NVIC, programming the ISR (interrupt handler). For a complete functionality of an interrupt the configuration of the respective peripherals and the resetting of the interrupt flags of the peripherals is still missing. These steps are done by custom Simulink blocks. These are described individually in section 5.5.

If the ISR (interrupt handler) are generated by the embedded coder of MATLAB/Simulink, a problem occurs during linking because two interrupt handlers with the same name are now available in the project.

To instruct the linker which of the existing ISRs has to be linked into the project, the function that has to be excluded has to be marked with __weak.

The keyword __weak tells the compiler to export a symbol as weak. It can be applied to variable, as well as function declarations and function definitions. If __weak is applied to a function definition, it behaves like a normally defined function, unless this function is present in the image with the same name in non-weak. In that case only the non-weak function is linked. [84]

To fix these linker errors all interrupt handlers generated by *STM32CubeMX* are marked with __weak. Thus the interrupt handlers generated by the embedded coder have precedence when linking. All interrupt handlers marked with __weak but not appearing in the MATLAB/Simulink generated code are treated as normal declared functions. To avoid having to declare all interrupt handlers manually with __weak after each pin muxing change in *STM32CubeMX*, a patch is created that is able to do that.

In listing 5.19 it can be seen how this patch is applied.

```
1 cd STM32CubeMXProjectDirectory
2 patch ./CM4/Src/stm32mp1xx_it.c ./patch/irq_weak.patch
```

Listing 5.19: Patching the interrupt handlers of the *STM32CubeMX* project to __weak

5.5 Implementation of hardware-related Simulink blocks

There are different ways to create custom Simulink blocks. [85] Not all methods are equally suitable for generating hardware specific C code. Level-2 MATLAB S-Functions are particularly suitable for generating hardware-related blocks, since they support the implementation of TLC files. The implementation of TLC files allows inlining. Level-2 MATLAB S-functions achieve lower execution speeds compared to C S-functions, since their code must be executed via the MATLAB engine and is not available in compiled form as for the C S-function. [86]

MATLAB S-Functions and Level-2 MATLAB S-Functions differ in their API. The API of the Level-2 MATLAB S-Functions is more extensive, and it is recommended to develop Level-2 MATLAB S-Functions blocks for newer developments. [87]

In this thesis, the developed hardware blocks do not perform any simulation, so it does not matter that by choosing the implementation by Level-2 MATLAB S-Functions, the method with the lower simulation time is chosen. The blocks are included in the Simulink model through a `.m` files. This file tells how many inputs and outputs a block has, and which configuration parameters are passed to the TLC during the code generation.

The block target file methods implemented in the `.tlc` file are also created within the `.m` file. These methods determine which TLC code blocks are inlined during code generation. There are, for example, block target file methods which are inserted at the start of the model, or which are iserted at each model step, as well as methods which serve the setup or the termination of the block. [88]

The two objects, block and system, are passed to the block target file methods. In the object block configuration parameters of the block are stored, some of them are generated by the `.m` files. The object system contains to the Simulink sub or root system information. [88]

The following block target file methods are summarized below:

BlockInstanceSetup The method `BlockInstanceSetup(block, system)` is a setup method that insertes TLC code for each block implemented in the Simulink model. [88]

BlockTypeSetup The method `BlockTypeSetup(block, system)` is a setup method that insertes TLC code for each block type implemented in the Simulink model. The application can occur, for example, when a lookup table is used multiple times, but only needs to be initialized once. [88]

Enable The TLC code of the `Enable(block, system)` method is then inserted if the block is in a Simulink subsystem that contains an enable function. [88]

Disable The TLC code of the `Disable(block, system)` method is then inserted if the block is in a Simulink subsystem that contains an disable function. [88]

Start The method `Start(block, system)` is used to insert TLC code into the start function. The start function is executed only at the beginning of the model. [88]

InitializeConditions The `InitializeConditions(block, system)` method inserts TLC code in two places. It inserts into subsystems that are con-

figured to reset back to their initial state when activated. And the method inserts TLC code into the start function when the blocks are inside the Simulink root system. The difference between the InitializeConditions and Start methods is that the block that performs its initialization with the InitializeConditions method is reinserted within a subsystem when reactivated. [88]

Outputs The method `Outputs(block, system)` inserts TLC code in two places. If the corresponding block is in the Simulink root, then the TLC code is inserted into the output function of the Simulink model. If the block is in a Simulink subsystem, the TLC code of the block is inserted into the output function of this subsystem. The output functions of the Simulink model can then be called, for example, at each model step. [88]

Update The method `Update(Block, System)` inserts TLC code into the update function of the model. It can be used for example to change an array index. [88]

Derivatives The `Derivatives(block, system)` method is used for inserting TLC code into the derivatives function of the model. This method allows the calculation of continuous block states. [88]

Terminate The `Terminate(block, system)` method is used to insert TLC code into the termination function of the model. The method can be used for example to reset initialized hardware or to free allocated memory. [88]

After an `.m` file and an `.tlc` file are created, a mask is created for the block. The main purpose of this mask is to allow entering the block parameters.

For example, the mask of an example block may look like in figure 5.13.

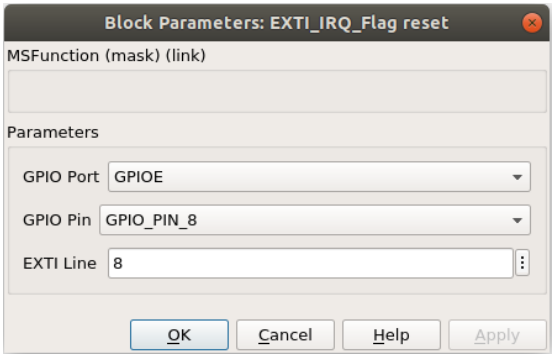


Figure 5.13: Block mask of the EXTI IRQ handler block

The mask creation is performed by the Mask editor. As seen in figure 5.14.

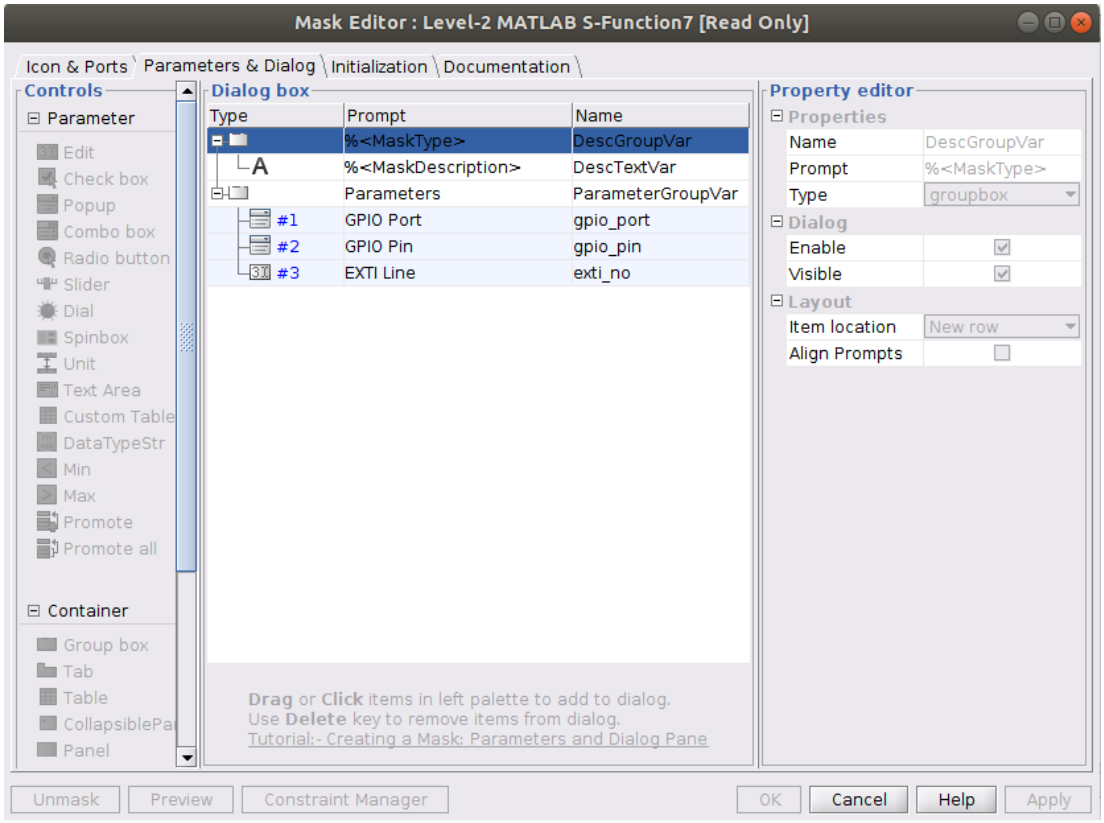


Figure 5.14: Mask editor, creating the block mask of the EXTI IRQ handler block

Next, the implementation of the developed Simulink blocks is described.

TIM_PWM_Config block

For the TIM_PWM_Config block, the block target file methods Start and InitializeConditions are implemented. The TLC code of the method Start is inserted inside the model before the TLC code of the method InitializeConditions. By using hardware configuration TLC code within the Start method, and the start commands within the InitializeConditions method, multiple timer channels of the same timer can be started. Following example describes how to set register bits for a timer PWM configuration for timer 1 and channel 1. For the configuration of a PWM the following bits in following registers must be programmed: [89]

1. Set the prescaler value in the prescaler register, shown in figure A.1.

2. Set the Auto-Reload value in the Auto-Reload register mapped in figure A.2.
3. Resetting the CC1S bit in Capture/Compare mode register, shown in figure A.3.If the bit is not set, the channel is configured as output.
4. Set PWM mode through data field OC1M in Timer Capture/Compare mode register, shown in figure A.3.
5. Set the duty cycle through the Capture/Compare register, shown in figure A.4.
6. Enable the timer channels by setting the CC1E bit in the register Capture/Compare Enable register, shown in figure A.5.
7. Starting the timer by setting the CEN bit in Timer control register 1, shown in figure A.6.

With the equation (A.1) and equation (A.2) the PWM frequency f_{Period} is set. The frequencies descriptions are listed here:

$f_{CK_{PSC}} = Prescaler\ input\ clock\ frequency$

$f_{CK_{CNT}} = Counter\ input\ clock\ frequency$

$f_{Period} = PWM\ frequency$

The created TIM_PWM_Config block and the corresponding block mask are shown in figure 5.15.

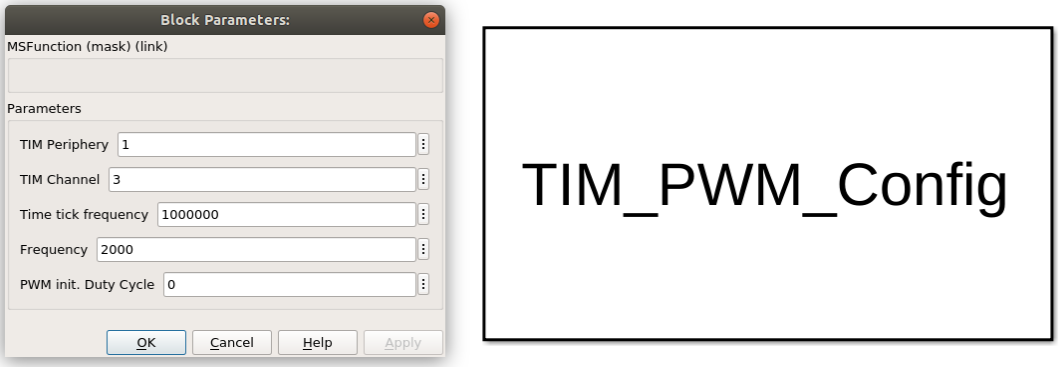


Figure 5.15: TIM_PWM_Config block and its block mask

If the block is used for synchronization of a PWM output, the user must enter the TIM peripheral, the channel, the timer tick frequency, the frequency and the initialization value of the duty cycle.

The setting of the frequency is not done fully automatically, because some timers have

a 16 bit and others a 32 bit counter register. This must be noted by the user before using the block so that the timer tick frequency (counter input clock frequency) can be set correctly. The value for the Prescaler register, the value for the Auto-Relode register and the value for the Capture/Compare register is determined within the TLC code.

TIM_Set_DC block

To set the duty cycle of a configured PWM it is only required to adjust the value of the Capture/Compare register. Shown in figure A.4. This is done by TLC code that is inserted by the block target file method Outputs.
The created TIM_Set_DC block and the corresponding block mask are shown in figure 5.16.

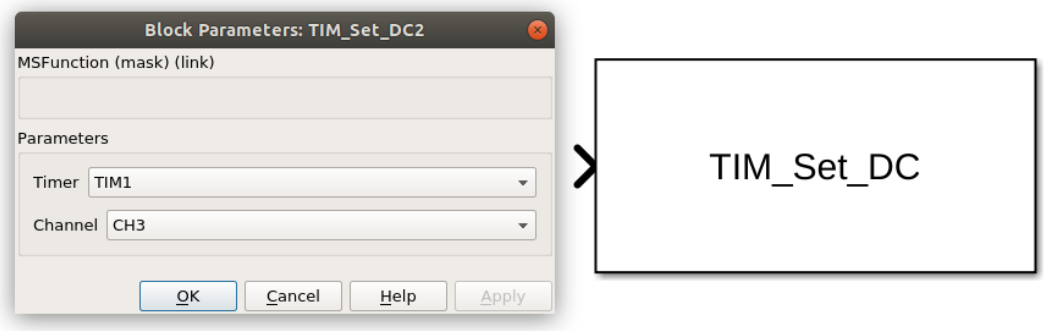


Figure 5.16: TIM_Set_DC block and its block mask

As seen in figure A.4, the user has to select the timer and the channel within the block mask.
The Simulink input type of the block is defined as `double`. The expected input values are between 0 and 100. Within the TLC code, a value is formed which is calculated from the Auto-Reload registers value and the Simulink input value. This value is written to the capture/compare register.

$$CC_{register} = (uint32_T)\frac{(ARR+1)*Input}{100} \tag{5.1}$$

At this point, it must be paid attention to the fact that the calculated value may not have a higher value than 65535 for 16 bit timers.

TIM_CC_Interrupt_Config_Flag_Reset block

The TIM_CC_Interrupt_Config_Flag_Reset block is implemented by the block target file method InitializeConditions and Outputs. For example, if a Capture/Compare interrupt is to be configured for channel 2 of timer 1, the bits are set as follows. Inside the InitializeConditions method, TLC code is inserted to set the CC2E bit in the Capture/Compare Enable register, to enable the channel, shown in figure A.5. And the bit CC2IE bit is set in the TIM1 DMA/Interrupt Enable register, shown in figure A.7.
The moment when the interrupt is triggered is determined by the third input field shown in figure 5.17. The value that is written into the timer Capture/Compare Register is calculated as described in equation (5.1).

To ensure that the interrupt can be triggered by the ARM-Cortex-M interrupt block, the Capture/Compare interrupt for the NVIC must be enabled in the STM32CubeMX configuration. The STM32CubeMX setting can be found under the selected timer peripheral, under NVIC, as shown in figure 5.17.

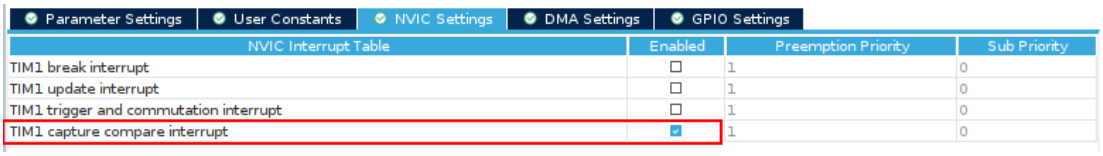


Figure 5.17: STM32CubeMX configuration to enable the interrupt call by the ARM-Cortex-M interrupt block

Within the block target file method Outputs, the TLC code checks whether the CCxIF bit of the TIMx status register is set. In this case, it is cleared. If the CCxIF bit is set when entering the TLC code of the Outputs method, a 1 is returned at the Simulink block output. If the bit was not set, a 0 is returned. The Simulink block output is configured as an `uint8_t` datatype. Returning whether the bit has been set enables the detection of multiple Capture/Compare channels within a timer Capture/Compare ISR.

The created TIM_CC_Interrupt_Config_Flag_Reset block and the corresponding block mask are shown in figure 5.18.

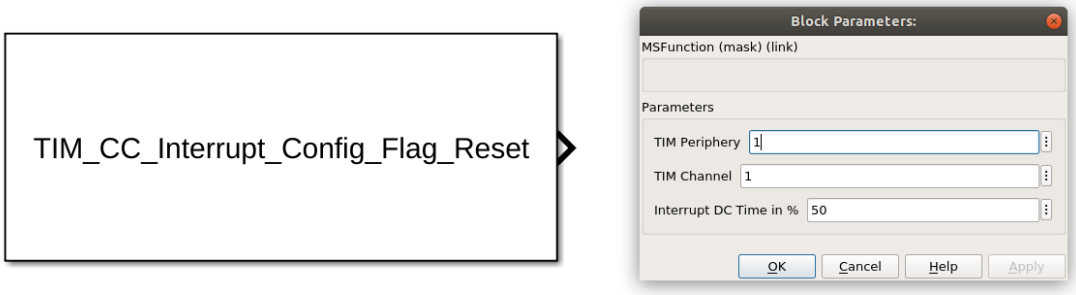


Figure 5.18: TIM_CC_Interrupt_Config_Flag_Reset block and its block mask

The user must select the desired timer and the timer channel within the block mask. This can be seen in figure 5.18.

TIM_Get_Counter

The block TIM_Get_Counter is very similar to the block TIM_CC_Interrupt_Config_Flag_Reset. Within the TLC code of the block target file method Outputs, the value of the timer counter register, seen in figure A.6, is returned to the block output. Then, in TLC code of the Outputs method, the value of the timer counter register is set to 0. The block output is assigned the datatype `uint32_t`, so that the block can be used for timers with 16 bit or 32 bit counter register.

The created TIM_Get_Counter block and the corresponding block mask are shown in figure 5.19.

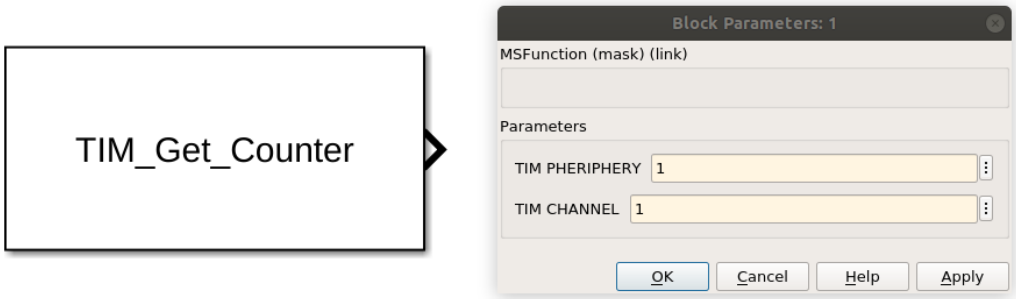


Figure 5.19: TIM_Get_Counter block and its block mask

This block is designed to be used in input Capture/Compare applications. The task of the block is to read the timer counter register and reset the interrupt flag of the timer. The user sets the timer peripheral and the configured timer channel as shown in figure 5.19 and positions this block in the Function-Caller subsystem that is called by the ARM Cortex-M interrupt block, which is configured as the corresponding Capture/-Compare interrupt.This structure is shown in figure 5.20.

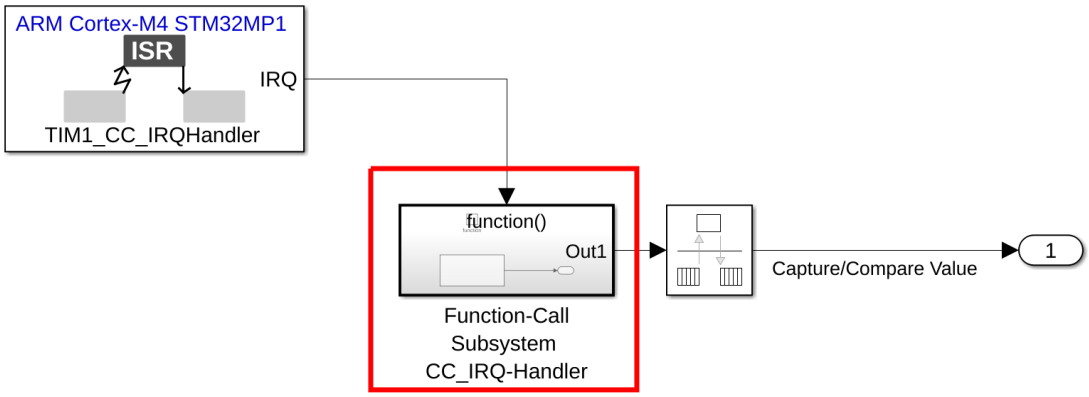


Figure 5.20: Location of operation of the TIM_Get_Counter block

The subsystem framed in red contains the TIM_Get_Counter block.

TIM_CC_Start_IR (only for the use of the external mode)

SPI_DMA_Transmit

The SPI_DMA_Transmit block is implemented by the block target file methods BlockTypeSetup, Start, Outputs, and Terminate.
Using the method BlockTypeSetup creates an `spi_dma_transmit.h` and an `spi_dma_transmit.c` file. The `spi_dma_transmit.c` file includes two functions. One is to initialize the chip select pin, which is selected by the block mask and the second function initializes the selected SPI DMA transmit peripheral. The following steps are performed within the initialization function:

1. Setting the bits: DIR, CIRC, PINC, MINC, PSIZE, MSIZE, PL, and PFCTRL in the DMA stream x configuration register, as seen in figure A.9.
2. Setting the base address of the memory through the DMA stream x memory 0 address register, as seen in figure A.10.
3. Setting the base address of the peripheral data register through the DMA stream x peripheral address register, as seen in figure A.11.
4. Setting the number of data items to transfer using the DMA stream x number of data register, as seen in figure A.12.
5. Selecting the input DMA request through the DMAMUX request line multiplexer channel x configuration register, as seen in figure A.13.
6. Enabling the DMA transmit stream through the SPI configuration register 1, as seen in figure A.16.
7. Setting the bits: TCIE, and TEIE in the DMA stream x configuration register, as seen in figure A.9.
8. Enabling the alternate function GPIOs controle through the SPI configuration register 2, as seen in figure A.17.
9. Enabling the serial peripheral through the SPI/I2S control register 1, as seen in figure A.18.

The block target file method Start initializes the SPI peripherals and the chip select pin, by calling the functions generated by the BlockTypeSetup method. Within the TLC code of the block target file method Outputs, the data of the input of the Simulink block (`uint8_t` vector) is copied to the configured base memory address. Afterward the CSTART bit of the SPI/I2S control register 1, seen in figure A.18, is enabled to start the DMA master transfer. The block target file method Terminate deinitializes the

configured DMA and SPI peripheral.
The SPI_DMA_Transmit block and its block mask is seen in figure 5.21.

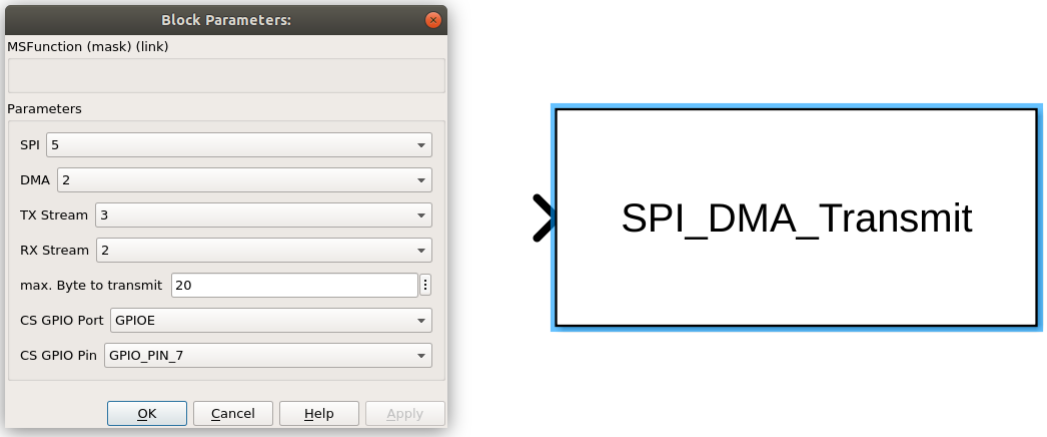


Figure 5.21: SPI_DMA_Transmit block and its block mask

In the block mask is defined which SPI peripheral, which DMA controller, which DMA stream is used for sending and which for receiving, as well as the maximum number of bytes that are transmitted. Additionally, a chip select pin can be selected.

SPI_DMA_Receive

When explaining the SPI_DMA_Receive block it is useful to refer to the SPI_DMA_Transmit block.
The differences are that during the BlockTypeSetup block target file method the two created files are called `spi_dma_receive.h` and `spi_dma_receive.c`. Their content is similar to the files of the SPI_DMA_Transmit.
One difference is for example that during initialization the the RXDMAEN instead of the TXDMAEN bit of the SPI configuration register 1 A.17 is set.
The block target file method Outputs copies the data received by the SPI peripheral using the DMA controller to the (`uint8_t` vector) output of the Simulink block.
The SPI_DMA_Transmit block and its block mask is seen in figure 5.22.

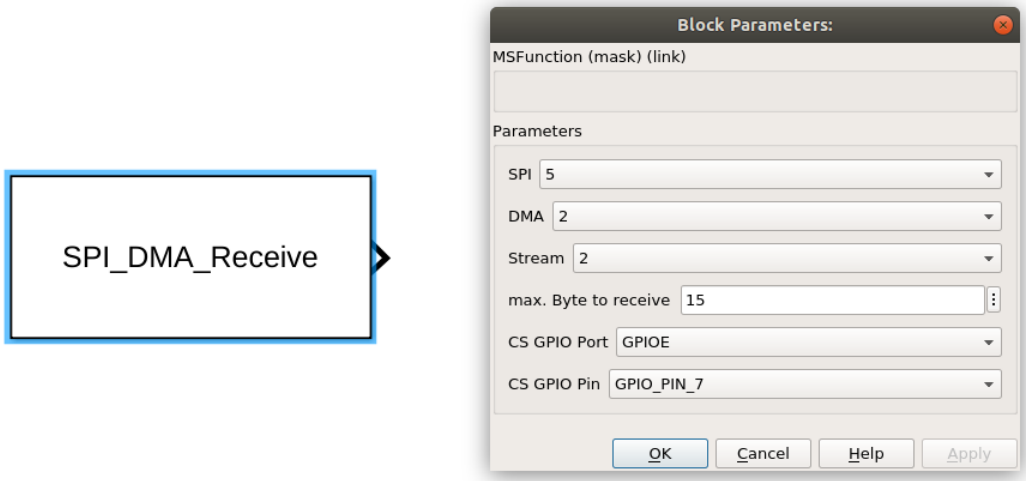


Figure 5.22: SPI_DMA_Receive block and its block mask

In the block mask is defined which SPI peripheral, which DMA controller, which DMA stream is used for receiving, as well as the maximum number of bytes that are received. Additionally, a chip select pin can be selected.

DMA_flag_handler

The block DMA_flag_handler is implemented to reset the TCIF bit in the DMA low interrupt status register, as seen in figure A.14. This is needed for the DMA interrupt implementation.

The block implementation uses only the Outputs block target file method. In the TLC code of the Outputs method, the interrupt flags transfer complete and transfer error are reset by the CTCIF and the CTEIF bits, as seen in figure A.15.

The Simulink block has two one-dimensional outputs of type `uint8_t`. The first output returns 1 if the transfer complete is set, the second output returns 1 if the transfer error is set. If the flags are not set, the outputs return 0.

The DMA_flag_handler block and its block mask is seen in figure 5.23.

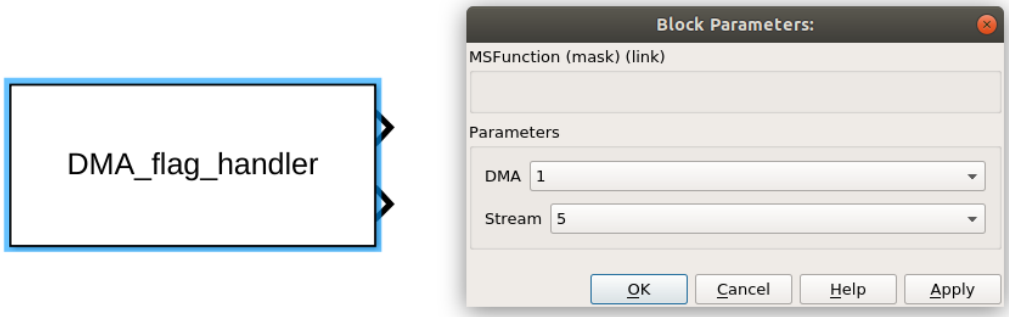


Figure 5.23: DMA_flag_handler block and its block mask

In the DMA_flag_handler block, the DMA controller and the DMA stream are specified.

The use of the DMA_flag_handler block is especially useful within a Function-Caller subsystem that is triggered by the corresponding DMA stream interrupt. The inside of such a Function-Caller is shown in figure 5.24.

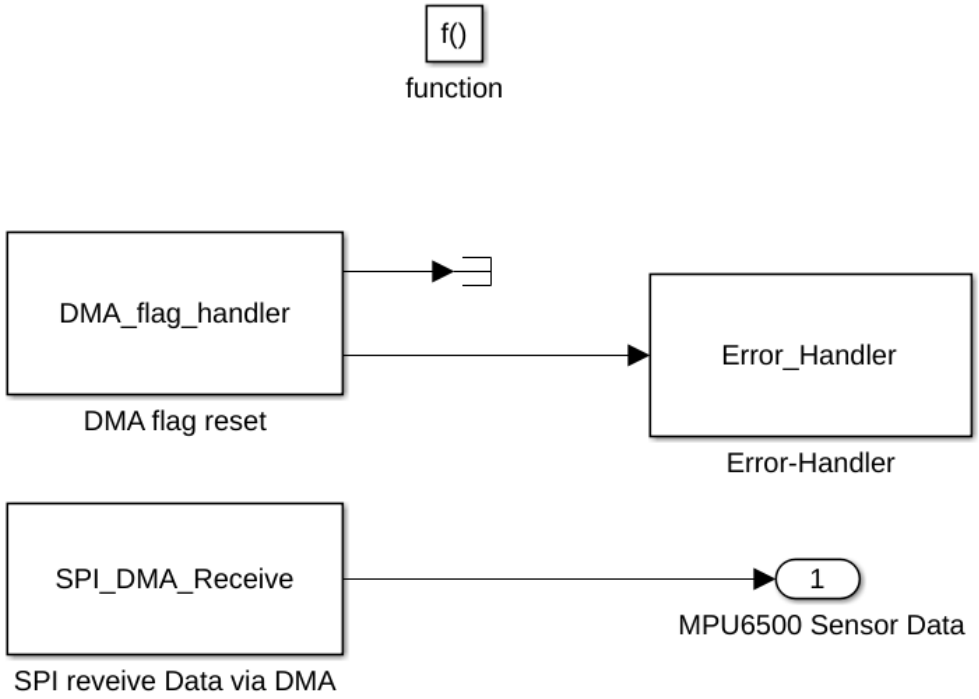


Figure 5.24: DMA_flag_handler block used inside of a Function-Caller

The second output returns if the transfer error flag is detected. The output is routed to the Error_Handler block. This is done to stop the firmware if a transfer error is detected.

Error_Handler

The Error_Handler block implementation uses only the Outputs block target file method. If the input of the Simulink block is not equal to 0, the function calls the C-project error function. The block does not have a mask, because there are no parameters that have to be set. The Error_Handler block can be seen in figure 5.24.

EXTI_flag_handler

The EXTI_flag_handler block implements the block target file methods Start and Outputs. The code generated by the Start method configures the input pin, selected by the mask, as a GPIO input, using no pull-resistor. Afterward, it configures the corresponding EXTI interrupt to the selected GPIO pin. In the Outputs method, the corresponding interrupt flag is reset. The EXTI_flag_handler block, similarly to the DMA_flag_handler block, is placed in the Function-Caller subsystem triggered by the EXTI interrupt.
The EXTI_flag_handler block and its block mask is seen in figure 5.25.

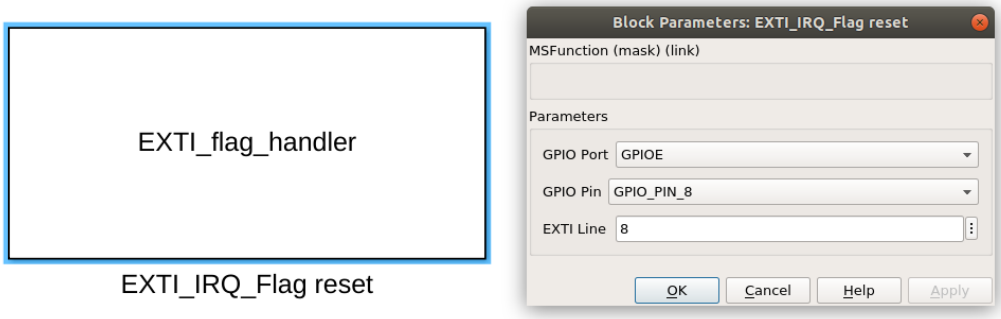


Figure 5.25: EXTI_flag_handler block and its block mask

In the block mask the GPIO port, the GPIO pin, and the EXTI line number is defined.

GPIO_Get_Input

The GPIO_Get_Input block implements the block target file methods Start and Outputs. In the method Start, the configuration of the GPIO pin is done.
The Start method reads the value of the IDR bit of the GPIO port input data register, seen in figure A.19. This bit has the value of the corresponding pin. The value is returned by the Simulink output.
The GPIO_Get_Input block and its block mask is seen in figure A.19.

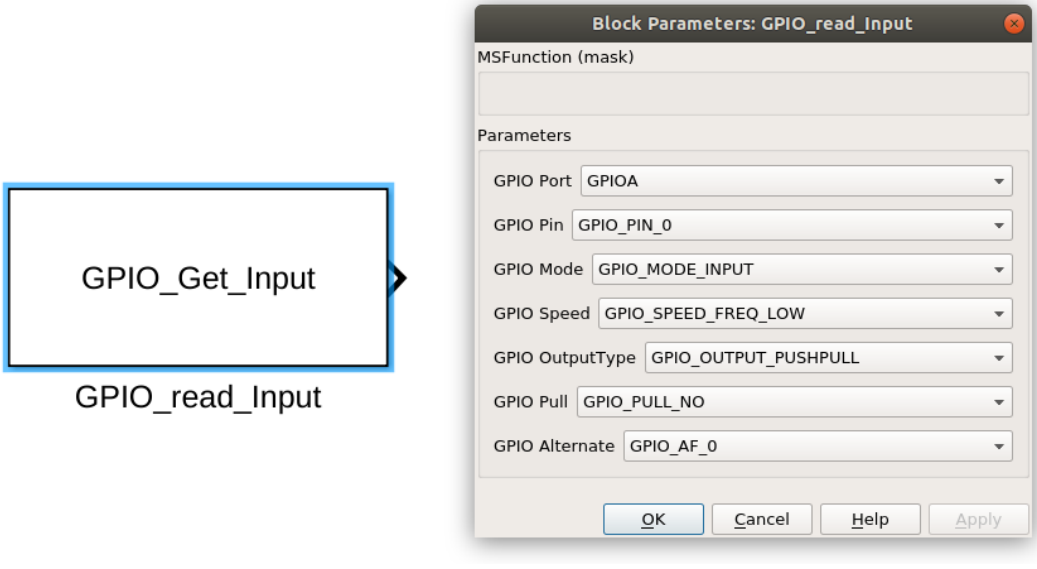


Figure 5.26: GPIO_Get_Input block and its block mask

Within the block mask the configuration of the GPIO pin can be done.

ADC_DMA_Data_request

The ADC_DMA_Data_request block is used to start the ADC DMA transfer. The block implementation is based on the example ADC DMA implementation [90].

The block is implemented by the block target file methods BlockTypeSetup, InitializeConditions, Outputs, and Terminate. A `adc_dma_request.h` and a `adc_dma_request.c` are created in the BlockTypeSetup method. The header includes headers required for the ADC implementation, such as the `adc.h` header, that is generated by the STM32CubeMX project.. A global array is also defined, the length of the array can be set by the input mask of the block. In the `adc_dma_request.c` file the callback functions of the ADC DMA IRQ handler are implemented. During the InitializeConditions method, an ADC calibration is performed as in [90]. The Outputs method implements the ADC DMA one shot, also performed as in [90]. The Terminate method deinitializes the ADC peripheral.
Figure 5.27 shows the ADC_DMA_Data_request block and the corresponding block mask.

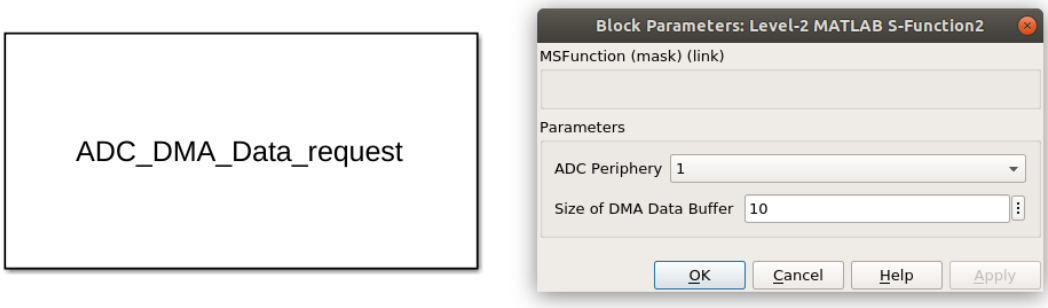


Figure 5.27: ADC_DMA_Data_request block and its block mask

Within the block mask the ADC peripheral and the data size of the DMA transfer are specified.

ADC_DMA_ISR

The ADC_DMA_ISR block is used to read the recorded DMA values of the ADC peripheral inside the Function-Caller subsystem, which is triggered by the DMA, which is configured to the ADC. The block is implemented using the Outputs method. In it, the ADC DMA interrupt handler is called, as in the example ADC DMA implementation [90]. If the DMA transfer is completed, the recorded ADC values are copied from the DMA memory to the Simulink output.

Figure 5.28 shows the ADC_DMA_ISR block and the corresponding block mask.

Figure 5.28: ADC_DMA_ISR block and its block mask

Within the block mask the ADC peripheral and the data size of the DMA transfer are specified.

MPU6500_DATA_REQUEST

The MPU6500_DATA_REQUEST block optimizes the time performance of the sensor data requeset of the MPU6500. The block is a lightweight version of the SPI_DMA_Transmit block 5.21. For the MPU6500_DATA_REQUEST block only the target file method Outputs is implemented. In this method, a constant `uint8_t` array of 15 B is transmitted via SPI unsing the DMA. The implementation of the Outputs method is like the Outputs method of the SPI_DMA_Transmit block. Except that the data that are sent via DMA do not have to be copied from the Simulink block input into the configured DMA memory area. The data to transmit for the sensor data request are already in the configured DMA memory at each call. That reduces C_1 of τ_1 by 42 μs.
Figure 5.29 shows the MPU6500_DATA_REQUEST block and the corresponding block mask.

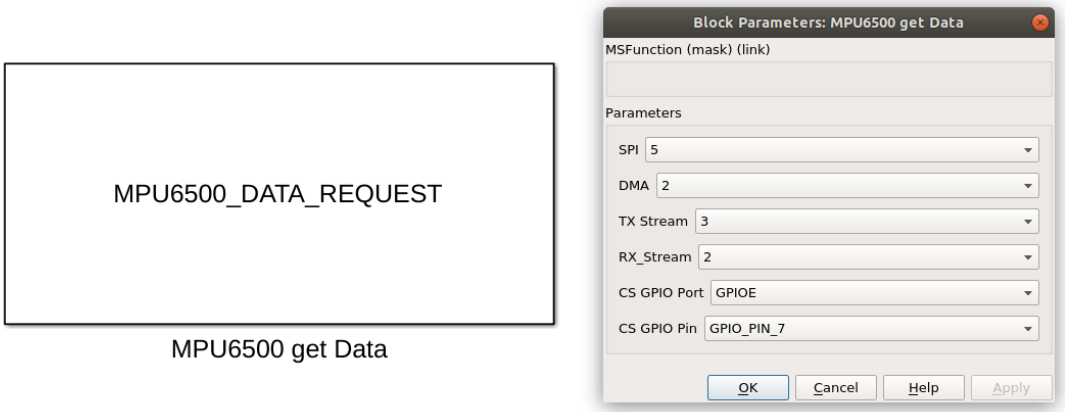


Figure 5.29: MPU6500_DATA_REQUEST block and its block mask

The block is configured like the SPI_DMA_Transmit block 5.21.

5.6 Mapping of the peripheral devices

The basic peripheral configuration of the STM32MP1 is done by the configuration tool STM32CubeMX.

STM32CubeMX is a microcontroller and microprocessor configuration tool, that uses a graphical user interface. After the graphical configuration the corresponding initialization codes for the Cortex-M processor are generated. The graphical interface allows the configuration of peripherals like GPIOs or Universal Synchronous/Asynchronous Receiver Transmitter (USART) interfaces, the system clock, memory configuration and middleware stacks like Universal Serial Bus (USB) or TCP/IP. Additionally STM32CubeMX generates a partial Linux devicetree for the Cortex-A during generation. [91]

Required peripherals are derived from the requirements of the Simulink target and requirements of the example application.

The Simulink target requires a timer. This triggers an interrupt after a configured time, which calls the model. To select this timer, the existing timers of the STM32MP1 are inspected first.

The STM32MP1 MPU has 14 timer units, if the low power timer and the realtime timer are not considered. These timers are divided into three categories, seen in table 5.1. [24]

Advanced-control timers	General-purpose timers	Basic timers
TIM1, TIM8	TIM2, TIM3, TIM4, TIM5, TIM12, TIM13, TIM14, TIM15, TIM16, TIM17	TIM6, TIM7

Table 5.1: Timers available on the STM32MP1 MPU

Advanced-control timers are equipped with a 16 bit counter and a 16 bit programmable prescaler. The maximal number of independent channels is 6. These independent channels can be driven in Input Capture, Output Capture, edge and centering aligned PWM generation and One-pulse mode. The timers can be used with Interrupts or DMA for the events: counter overflow/underflow, counter initialization,counter start, counter stop, Input capture, Output Capture and break inputs. [24]

General-purpose timers are equipped with a 16 bit or 32 bit counter and a 16 bit programmable prescaler. The maximal number of independent channels is 4. These independent channels can be driven in Input Capture, Output Capture, edge and centering aligned PWM generation and One-pulse mode. The timers can be used with Interrupts or DMA for the events: counter overflow/underflow, counter initialization,counter start, counter stop, Input capture, Output Capture and break inputs. [24]

Basic timers are equipped with a 16 bit counter and a 16 bit programmable prescaler. These timers have only one channel. These timers can be used to trigger a Digital-to-Analog Converters (DAC)s. The timers can be used with the Interrupt event: counter overflow. [24]

To leave as many options open as possible for the firmware generated by Simulink, one of the basic timers is selected to call the model step. The basic timers are sufficient for this task. Since the basic TIM6 in the default STM32CubeMX project [92] is mapped to the Cortex-A7 processor, the unused TIM7 is selected.

The default configuration file is used as the base configuration. This has the advantage that the configuration of, for example, the DDR, HSEM, IPCC, Independent WatchDoG (IWDG)s, system clock and much more are already done.

Figure 4 shows how many pins are occupied by the default configuration.

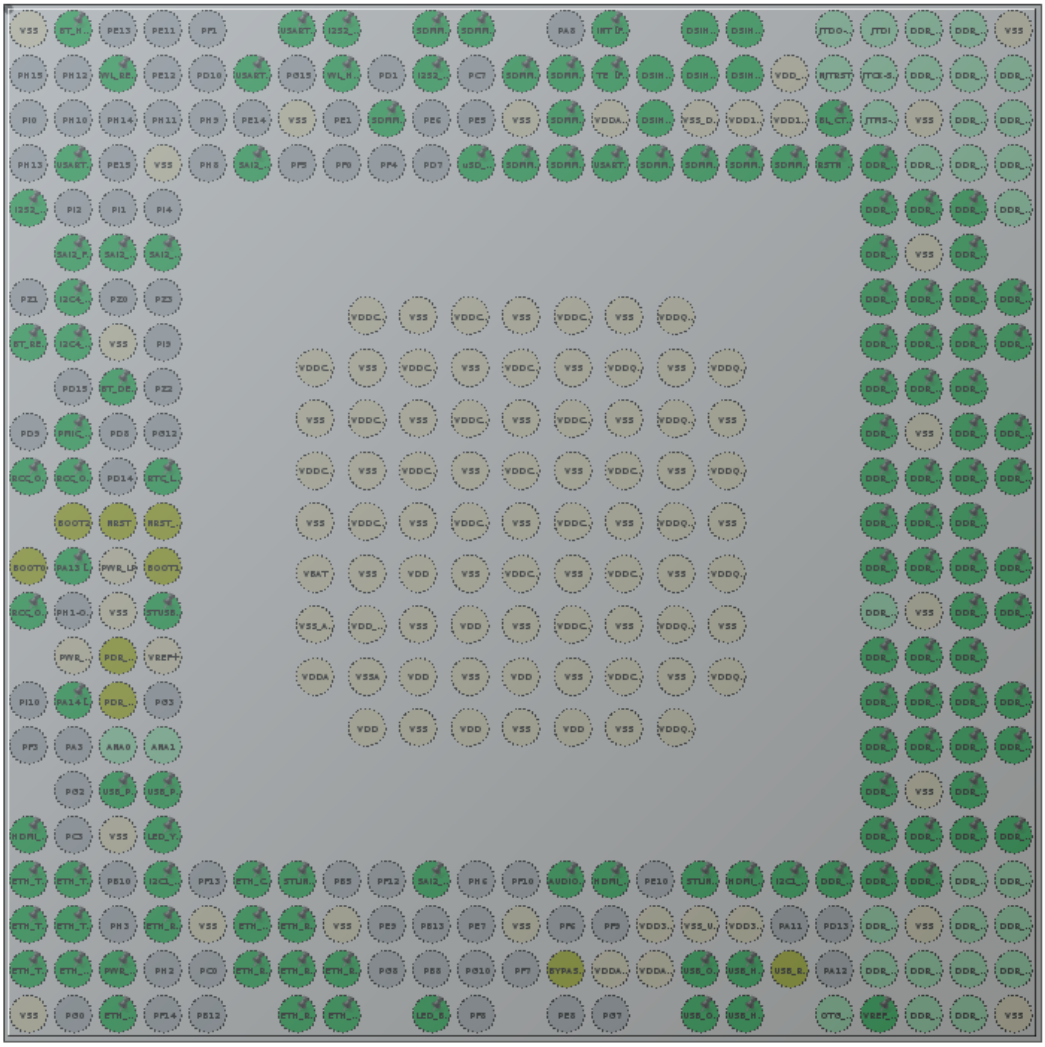


Figure 5.30: Default STM32CubeMX Configuration of the STM32MP157C-DK2

All pins that are grayed out can still be configured.

The peripheries used in the example application can be derived from the of the existing “Balance Car daughterboard” [39].

To give an overview of the existing hardware components that need to be connected to the STM32MP1 by the “Balance Car daughterboard”, a summary of the hardware components used in this project follows.

Gyroscope and accelerometer

The 3-axis gyroscope and the 3-axis accelerometer sit together in one chip. The used sensor type is the MPU6500. Gyroscope sensors and acceleration sensors can be programmed for different scale ranges. The sensor can communicate via an Inter-Integrated Circuit (I2C) or an SPI communication interface. [93]

Hall encoder

The Hall effect describes a voltage generated in a magnetic field, perpendicular to a current flow. Hall effect proximity sensors detect magnetic field changes caused by the movement of a metallic object. [94]
In figure 5.31 the measured hall signals are shown in a clockwise direction. In figure 5.32 the measured signals are shown counterclockwise. The rotation speed is calculated from the periodic duration of one signal. It is seen that the direction of rotation can be taken from the second sensor signal. If the edge of the blue signal rises while the pink signal has a low level, the direction of rotation is clockwise, if the pink signal has a high level at the time of the rise of the edge of the blue signal, the motor rotates counterclockwise.

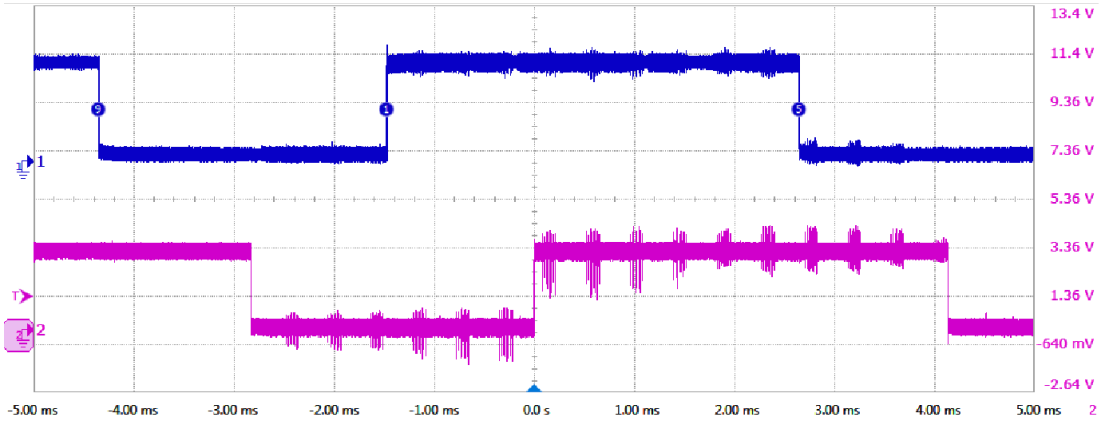


Figure 5.31: Hall signal clockwise

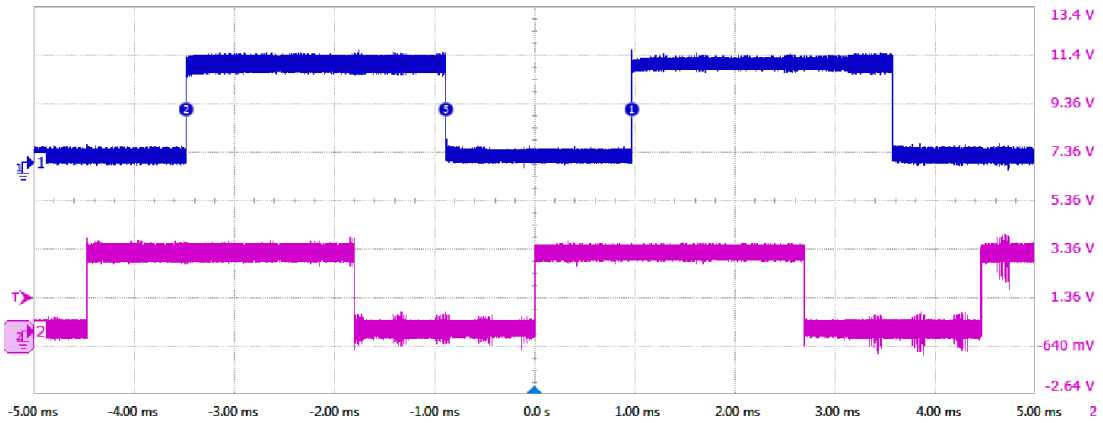


Figure 5.32: Hall signal counterclockwise

Motor driver

The TB6612FNG motor driver is mounted on the board and is suitable for supplying two Direct Current (DC) motors with energy. The TB6612FNG is controlled by two input lines, a standby line, and a PWM signal. The two input lines enable the direction settings (clockwise and counterclockwise), as well as short braking and a stop mode. The standby line enables to put the motor controller in a standby mode. [95]

Motor

The motors installed in the self-balancing robot are 3 W DC motors. Their drive torque is amplified by a gearbox with a ratio of 1:30. [96]
A image of the motor with its gearbox is seen figure 5.33.

Figure 5.33: DC motor with speed reducer [96, p. 29]

Ultrasonic sensor

The ultrasonic sensor type HC-SR04, shown in figure 5.34, is a contactless distance sensor, which has a measuring range from 2 cm to 4 m. [97]

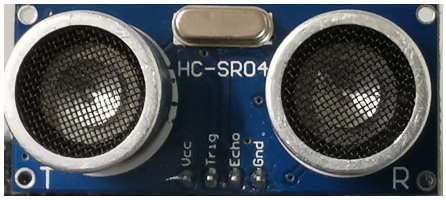


Figure 5.34: Ultrasonic sensor

Balance Car daughterboard connectors

Table 5.2 shows connectors of the “Balance Car daughterboard”, these are mapped to the peripheries of the STM32MP1 in the following steps.
In red, the table shows pins that belong to the power supply. In light green are the pins that belong to the motor interface. In orange are shown Speed encoder interface pins. In purple, the pin of the infrared receiver is shown. In dark gray are the pins of the Wireless Local Area Network (WLAN) and Bluetooth interface. The blue pins are the connection to the ultrasonic sensor. The VIN_ADC pin is shown in dark green. Light gray are the pins that are not connected.

Some of the pins from table 5.2 must be assigned to fixed pins of the STM32MP1.

Power supply				MPU_INT	1	2	ESP32_EN
Motor interface				IR_RXD	3	4	MTR_PWMA
Speed encoder interface				MTRR_N	5	6	MTRR_P
Gyro & Accel interface				MTRL_P	7	8	MTRL_N
Infrared receiver				ESP32_CMD8	9	10	MTR_STBY
WLAN & Bluetooth interface				VCC5_GPIO	11	12	GND
Ultrasonic sensor				MTRR_A	13	14	MTRR_B
VIN_ADC				MTRL_A	15	16	MTRL_B
				MPU_CS_n	17	18	MTR_PWMB
				MPU_SCL_SCLK	19	20	MPU_SDA_SDI
				MPU_AD0_SDO	21	22	MPU_FSYNC
				TRIG0	23	24	ECHO0
				TRIG1	25	26	ECHO1
				ESP32_UART0_TX	27	28	ESP32_UART0_RX
				VCC3P3	29	30	GND
VCC5_GPIO	1	2		ESP32_UART0_CTS	31	32	ESP32_UART0_RTS
VIN_ADC	3	4		ESP32_CMD0	33	34	ESP32_CMD4
	5	6		ESP32_CMD1	35	36	ESP32_CMD5
	7	8		ESP32_CMD2	37	38	ESP32_CMD6
	9	10	GND	ESP32_CMD3	39	40	ESP32_CMD7

Table 5.2: Pin assignment of the Balance Car Daughterboard (cf. [39])

These are the powersupply and the ground pins. Some of the units available on the “Balance Car daugtherboard” are not to be implemented in the sample application. These devices are the infrared receiver and the WLAN and Bluetooth interface. The pins belonging to these units are therefore not mapped to the STM32MP1 and remain not connected.

During the mapping, it is tried to assign the pins to the Arduino connectors if it is possible because they are the main connection of the MPU to the connection board. Other connections that cannot be assigned to the Arduino connectors are mapped to GPIO connectors CN2 [98].

Mapping the motor driver

The advanced timer TIM1 is selected as timer peripheral for the tow PWMs. The selected PWM channels 3 and 4 are connected to the Arduino connectors. The remaining inputs of the motor driver are connected to GPIO pins that are configured as outputs. These GPIO pins are also located on the Arduino connectors.

The assignment can be seen in table 5.3.

Balance car daughterboard	Peripheral MP1	Function	Pin MP1
MTR_PWMA	TIM1 Channel 3	PWM	PE13
MTRR_N	GPIO	Output	PD14
MTRR_P	GPIO	Output	PE10
MTR_PWMB	TIM1 Channel 4	PWM	PE14
MTRL_P	GPIO	Output	PE9
MTRL_N	GPIO	Output	PD1
MTR_STBY	GPIO	Output	PA12

Table 5.3: Mapping the pins of the TB6612FNG to the peripheries of the MP1

Mapping the gyro and acceleration sensor

For the MPU6500 the SPI interface SPI5 is selected. The pins of SPI5 are located on CN2. The remaining GPIOs are mapped to GPIO pins on the Arduino connector. The assignment can be seen in table 5.4.

Balance car daughterboard	Peripheral MP1	Function	Pin MP1
MPU_INT	GPIO	Input	PE8
MPU_CS_N	GPIO	Output	PE7
MPU_SCL_SCLK	SPI5	SCK	PF7
MPU_SDA_SDI	SPI5	MISO	PF9
MPU_AD0_SDO	SPI5	MOSI	PF8
MPU_FSYNC	GPIO	Output	PE1

Table 5.4: Mapping the pins of the MPU6500 to the peripheries of the MP1

Mapping the hall sensors

Two timers of the type general-purpose timer are selected for the two hall sensors. For the hall sensor on the right motor the timer peripheral, TIM4 channel 4 is selected. This is located on the Arduino connector. For the hall sensor on the right motor, timer peripheral TIM2 channel 2 is selected. The corresponding pin is located on CM2. The

second outputs belonging to the respective sensors are assigned to the GPIOs of the Arduino connector configured as inputs. The assignment can be seen in table 5.5.

Balance car daughterboard	Peripheral MP1	Function	Pin MP1
MTRR_A	TIM4 Channel 4	Input Capture	PD15
MTRR_B	GPIO	Input	PG3
MTRL_A	TIM2 Channel 1	Input Capture	PG8
MTRL_B	GPIO	Input	PH6

Table 5.5: Mapping the pins of the hall sensors to the peripheries of the MP1

Mapping the ultrasonic sensor

For the ECHO0 pin of the ultrasonic sensor, the general-purpose timer TIM5 is selected. A pin on the CN2 belongs to the assigned channel 2 of TIM5. The GPIO pin needed for TRIG0 is assigned to a pin on the Arduino connector. This GPIO pin is configured as an output. The assignment can be seen in table 5.6.

Balance car daughterboard	Peripheral MP1	Function	Pin MP1
ECHO0	TIM5 Channel2	Input Capture	PH11
TRIG0	GPIO	Output	PE11

Table 5.6: Mapping the pins of the ultrasonic sensor to the peripheries of the MP1

Mapping the voltage measurement

The pin where the battery voltage can be measured is assigned to the ADC1 peripheral. The selected input of the channel is the single-ended input 6, which is located on the Arduino connector. The assignment can be seen in table 5.7.

Balance car daughterboard	Peripheral MP1	Function	Pin MP1
VIN_ADC	ADC1 IN6	Single-ended	PF12

Table 5.7: Mapping the pin of the voltage mesurement to the peripheries of the MP1

Device tree

Since the peripherals mapped in section 5.6, except the GPIO peripherals, must be assigned according to [14], it is necessary to compile the device tree [99] generated by STM32CubeMX according to [100] and load it into the Linux OS.

5.7 Implementation of real-time firmware

In this section, the implementation of real-time software for the controller of the self-balancing robot, running on the Cortex-M4, is described using the embedded coder in MATLAB Simulink.

Model based implementation

This section describes the development of the real-time firmware for the control of the self-balancing robot. The development is done by using model-based design. For this purpose, a Simulink model is created. The embedded coder, the developed file customization template, and the developed Simulink blocks are used. In addition, some of the Simulink blocks from [6] are used.
In the following, the developed Simulink model is described. The overall view of the Simulink model is shown in figure 5.36.

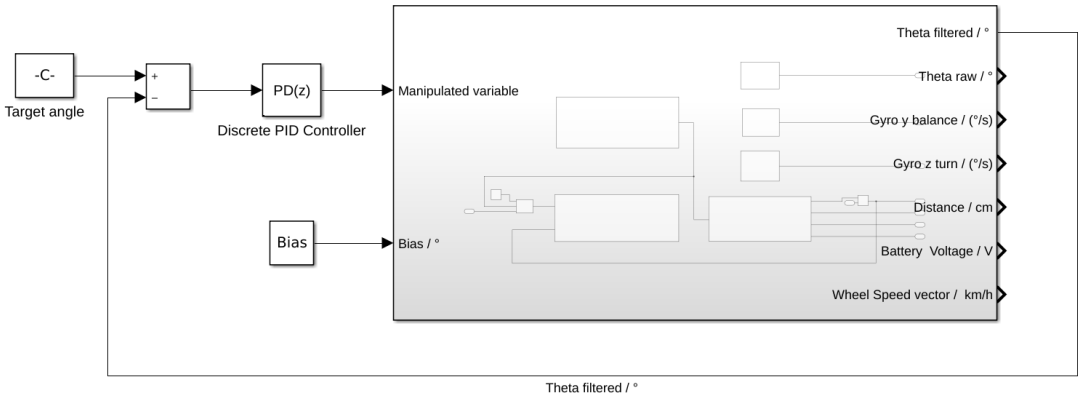


Figure 5.35: Overall view of the self-balancing robot Simulink model

The Simulink model shown in figure 5.36 was designed based on figure 5.62.
The target angle represents the reference variable w. After the controller block the manipulated variable can be seen. The controlled variable of the subsystem that is feedback is the theta filtered signal. Figure 5.36 shows the contents of the main subsystem from figure 5.36.

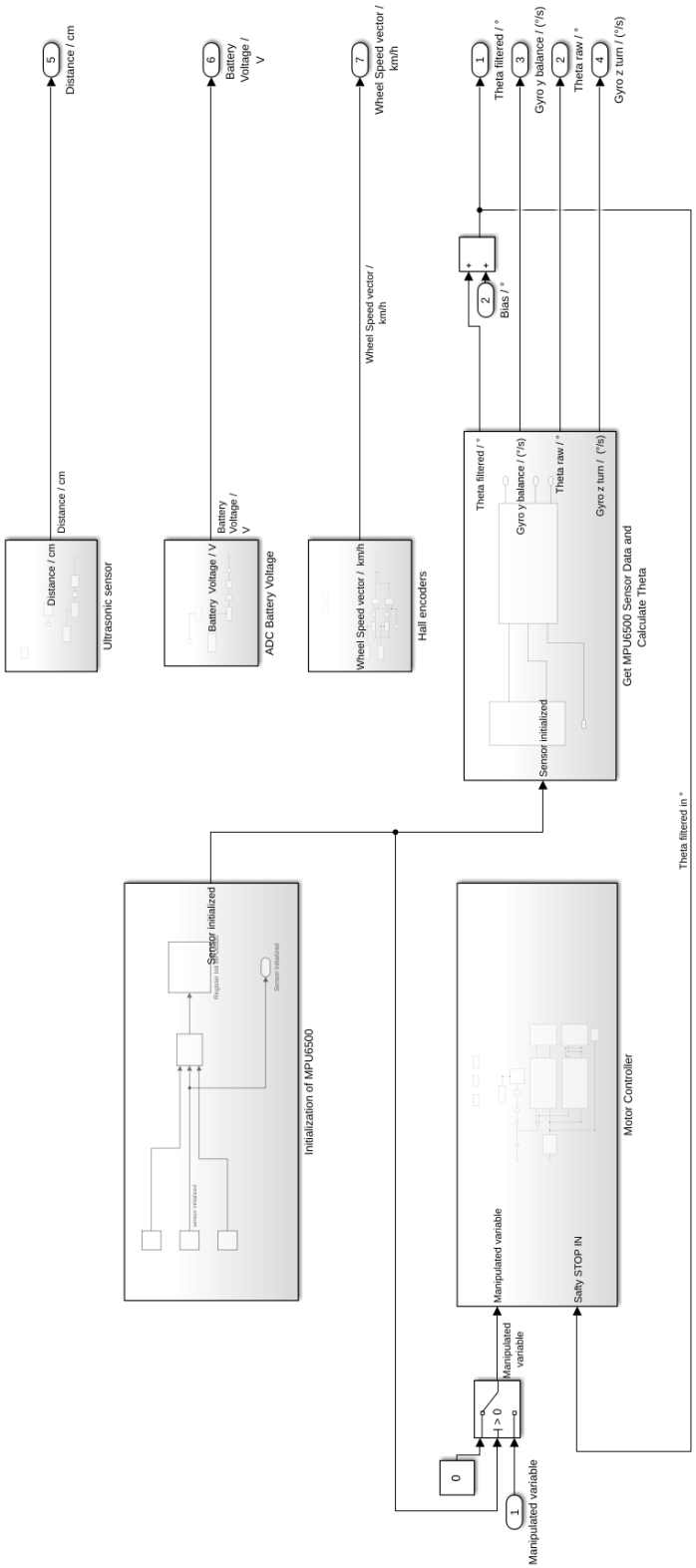


Figure 5.36: Content of the main simulink subsystem

The content of the main subsystem is divided into 6 subsystems. To give a general overview, the “Motor controller” subsystem forms the actuator and the “Get MPU6500 Sensor Data and Calculate Theta” subsystem the measuring device.

Inizialisation of MPU6500

The content of the “Inizialisation of MPU6500” subsystem, seen in figure 5.37, has the task to initialize the MPU6500.

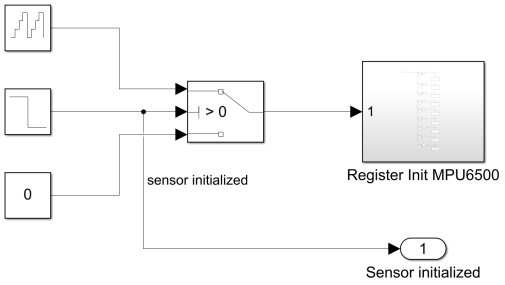


Figure 5.37: Subsystem: “Inizialisation of MPU6500”

An increasing sequence of numbers is generated by a counter, which is used to write register settings in the “Register Init MPU6500” subsystem at specified points in time. The counter increments every 10 ms. After 1.5 s the initialization is finished by the step function. The “Register Init MPU6500” subsystem is shown in figure 5.38. In this subsystem, the input signal of the counter is compared by the equal block with the mapped values. If the value and the counter signal are equal, the triggered subsystem connected to the equal block is executed. In the triggered subsystems, the registers of the MPU6500 are configured, according to [101]. This is done by transmitting data using SPI. The SPI transmission is done by using the SPI_DMA_Transmit block 5.21.

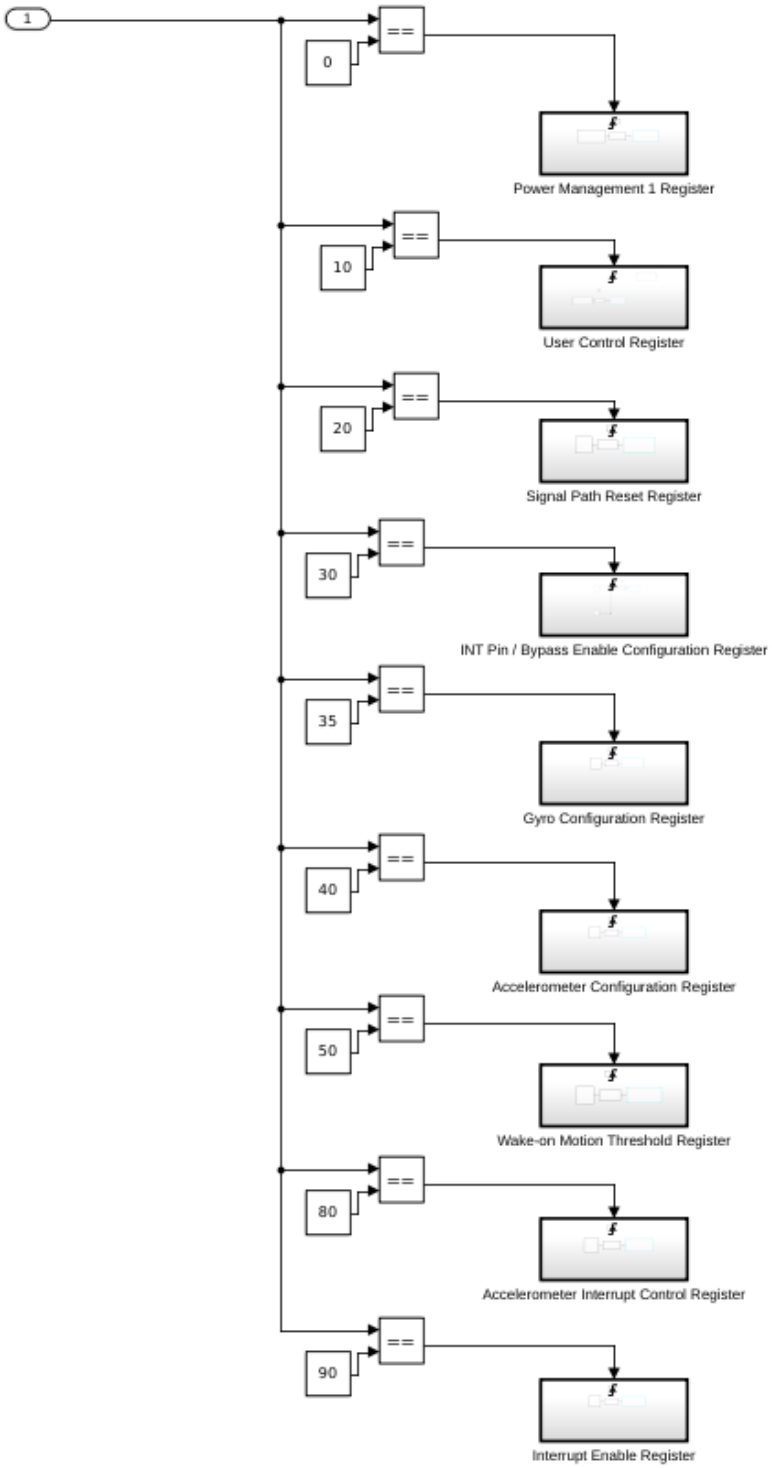


Figure 5.38: Subsystem: “Register Init MPU6500”

Get MPU6500 Sensor Data and Calculate Theta

The subsystem “Get MPU6500 Sensor Data and Calculate Theta” is divided into the subsystems “Receive MPU6500 Data” and “Calculate robot angle”, seen in figure 5.39. The task of the subsystem “Receive MPU6500 Data” is to read out the data of the MPU6500 as they are available. The task of the subsystem “Calculate robot angle” is to calculate the tilt angle θ_x of the robot.

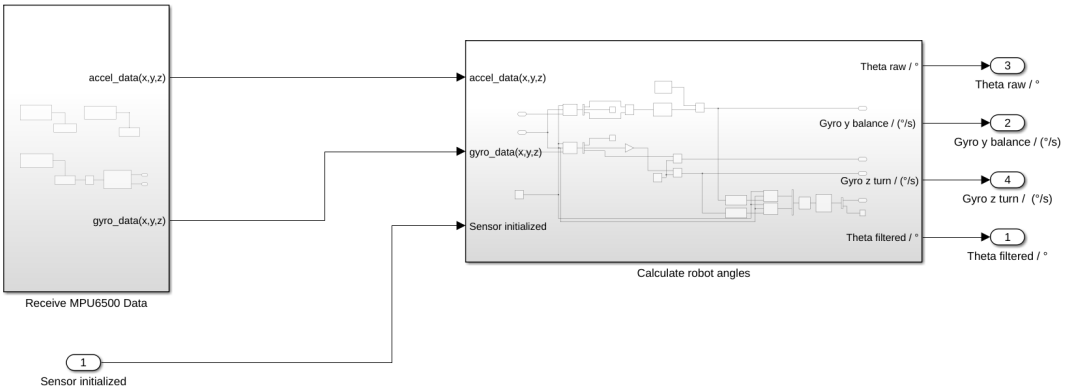


Figure 5.39: Subsystem: “Get MPU6500 Sensor Data and Calculate Theta”

Figure 5.40 shows the contents of subsystem “Receive MPU6500 Data”. To receive the sensor data 3 interrupts are used.

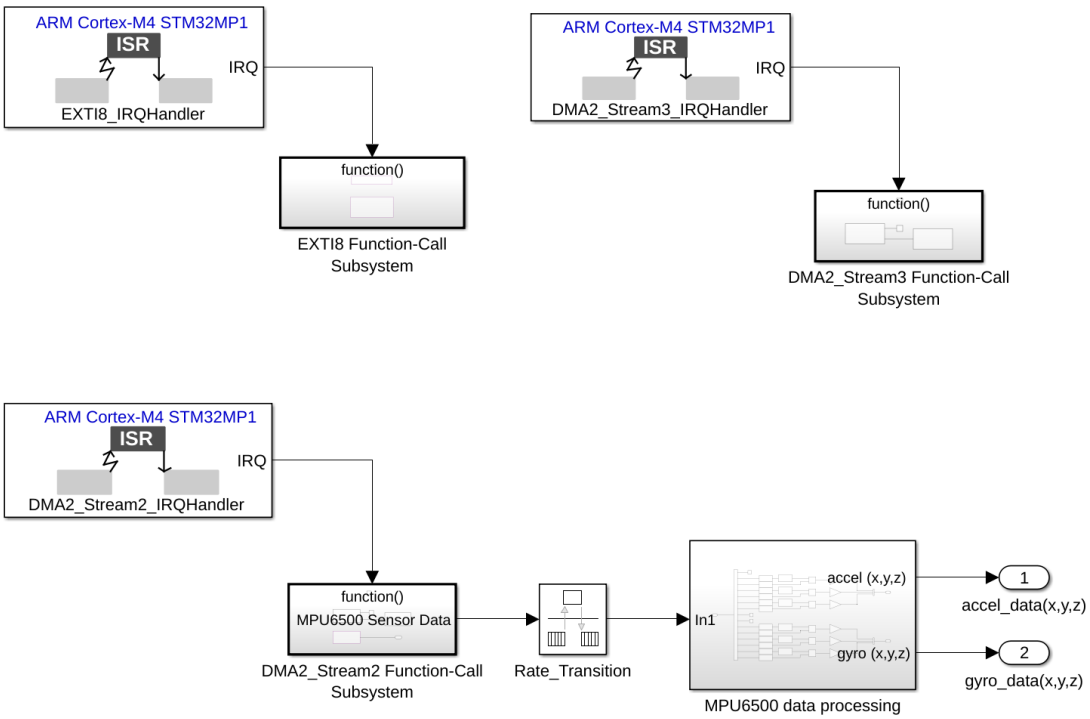


Figure 5.40: Subsystem: “Receive MPU6500 Data”

The EXTI interrupt detects, when data is available in the MPU6500 sensor, by detecting the MPU_INT pin signal of the sensor.
Inside the EXTI Function-Call subsystem, the EXTI IRQ status flag is reset using the EXTI_flag_handler block 5.25, and a data request is transmitted via SPI using the MPU6500_DATA_REQUEST block 5.29. The transmission of the data request triggers a DMA2 Stream3 transmission complete IRQ. The corresponding interrupt status bit is reset by the DMA_flag_handler block 5.23 in the Function-Call subsystem of the DMA2 Stream3 IRQ. After the SPI data, sent by the MPU6500 sensor, is received by the DMA2 stream2, the corresponding IRQ is triggered.
In the Function-Call subsystem of the DMA2 Stream2 IRQ, the corresponding interrupt status bit is reset by the DMA_flag_handler block 5.23 and the received sensor data is returned by the SPI_DMA_Receive block 5.22. The sensor data is fed into the “MPU6500 data processing” subsystem via the rate transition block. The content of the “MPU6500 data processing” subsystem is seen in figure 5.41.

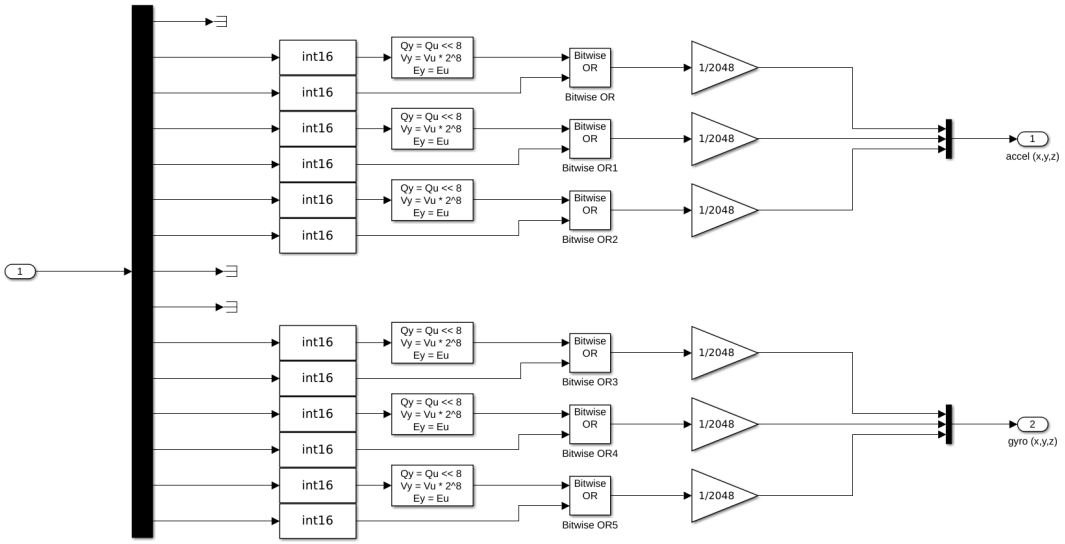


Figure 5.41: Subsystem: “MPU6500 data processing”

The figure shows how the raw data of the sensor is converted into data type `int`. According to [101] every acceleration and gyro values is stored in a 16 bit register. To transfer these values through a SPI running with a 8 bit data width, the values are seperated in a high byte and a low byte. The high byte of the value is shifted 8 bits to the left and is then combined bitwise by a or operation with the lower byte.

In the upper half of figure 5.41 the acceleration values are formed, in the lower half the gyro values. Then a scaling factor is applied to each value. The bus signals accel(x,y,z) and gyro(x,y,z) are fed into the subsystem “Calculate robot angle”,seen in figure 5.42.

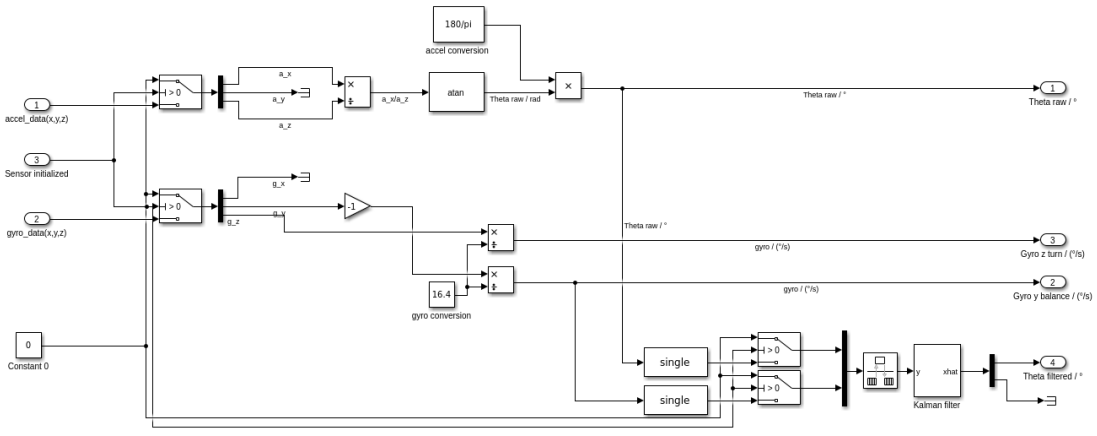


Figure 5.42: Subsystem: “Calculate robot angle”

In the subsystem “Calculate robot angle”, the robot tilt angle is calculated from the values of the bus signals accel(x,y,z) and gyro(x,y,z). The third subsystem input Sensor init is used to feed the data processing with a constant 0 value during the initialization of the MPU6500. In the upper area, the tilt angle θ_x of the robot is calculated from the acceleration value of the x axis a_x and the acceleration value of the z axis a_z.

$$\theta_x = \arctan\left(\frac{a_x}{a_z}\right) \tag{5.2}$$

In the middle, the gyro values $gyro_y$ and $gyro_z$ are calculated. The Sensitivity Scale Factor $K_{Sensitivity} = 16.4$ taken from [93] is used for this. Due to the geometric arrangement, the g_y value must be multiplied by -1.

$$gyro_y = \frac{g_y}{K_{Sensitivity}} \cdot (-1) \tag{5.3a}$$

$$gyro_z = \frac{g_z}{K_{Sensitivity}} \tag{5.3b}$$

In the lower half of figure 5.42, θ_x and $gyro_y$ are routed into the Kalman filter.

Implementation of a Kalman filter

Since the acceleration values are strongly influenced by impacts, it is not possible to control the robot without filtering the values of the acceleration and gyro sensor. The measurement of the acceleration values is strongly influenced by strong changes in the rotation speed, especially by directional changes. The angle θ_x calculated from the acceleration values is so strongly influenced by these impacts that deviations from the actual angular position occur.
In this physical system, the redundancy of the measured values acceleration, resulting in an angular position θ_x, and gyro value $gyro_y$ can be used to apply the Kalman filter described in section 2.11. The interrelated variables are shown in equation (5.4).

$$gyro_y(t) = \dot{\theta}_x(t) \tag{5.4}$$

The state vector $\underline{x}(t)$, the derived state vector $\underline{\dot{x}}(t)$ and the input vector $\underline{y}(t)$ for the state differential equation (2.8) are determined in equation (5.5a).

$$\underline{x}(t) = \begin{bmatrix} \theta_x(t) \\ gyro_y(t) \end{bmatrix} \tag{5.5a}$$

$$\underline{\dot{x}}(t) = \begin{bmatrix} \dot{\theta}_x(t) \\ \dot{gyro}_y(t) \end{bmatrix} = \begin{bmatrix} gyro_y(t) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \underline{z}(t) \tag{5.5b}$$

System matrix $\underline{A}$, input matrix $\underline{B}$, output matrix $\underline{C}$, feedthrough matrix $\underline{D}$, and the matrix of the system noise $\underline{G}$ are set up and inserted into the state-space equations, seen in equation (5.6).

$$\underline{\dot{x}}(t) = \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}}_{A} \cdot \underline{x}(t) + \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{B} \cdot u(t) + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{G} \cdot z(t) \tag{5.6a}$$

$$\underline{y}(t) = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{C} \cdot \underline{x}(t) + \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{D} \cdot u(t) \tag{5.6b}$$

The state-space model is discretized. To do this, the matrices $\underline{A}$, $\underline{B}$, and $\underline{G}$ are transformed into $\underline{A}_d$, $\underline{B}_d$ and $\underline{G}_d$ as seen in equation (2.11). The results are shown in equation (5.7).

$$\underline{A}_d = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix}, \qquad \underline{B}_d = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad \underline{G}_d = \begin{bmatrix} T_s \\ 1 \end{bmatrix} \tag{5.7}$$

The observability of the system is verified as described in section 2.11.
The rank of the observation matrix for the discretized system $\underline{S}_B^*$ is calculated.

To determine if the system is observable the rank of the ,observation matrix for the discretized system $\underline{S}_B^*$ must be equal to the order of the system. [34]

The order of this system is $n = 2$. The result of this calculation is seen in equation (5.9).

$$\underline{S}_B^* = \begin{bmatrix} \underline{C} \\ \underline{C} \cdot \underline{A}_d \end{bmatrix} \tag{5.8}$$

$$\text{Rang}(\underline{S}_B^*) = \text{Rang}\left(\begin{bmatrix} \underline{C} \\ \underline{C} \cdot \underline{A}_d \end{bmatrix} \right) = 2 \tag{5.9}$$

Since $\text{Rang}(\underline{S}_B^*) = n$, the system is observable, and the Kalman filter can be applied.

To apply the Kalman filter the values of the covariance matrix of the system noise $\underline{Q}$ and the covariance matrix of the measurement noise $\underline{R}$ have to be determined. [34]

It is assumed that the system/process noise vector $\underline{z}(k)$ is a scalar quantity.
Therefore the covariance matrix of the system noise $\underline{Q}$ will also be a scalar value.
It is assumed that the angular velocity changes by a maximum of $\frac{3^\circ}{1\,\text{ms}} \approx 3\,\text{ms}^{-1}$ within a sample period. It is assumed that the change of the angular velocity is normally distributed. It is assumed that the maximum velocity change is approximately to value $1 \cdot \sigma$.
From these assumptions, the system noise $Q(k)$ can be calculated as follows:

$$Q(k) = \sigma_v^2 = \left(\frac{3\,\text{ms}^{-1}}{1} \right)^2 \approx 9\,\mu/\text{s}^{-2} \tag{5.10}$$

For the calculation of the covariance matrix of the measurement noise $\underline{R}$ the assumptions are met, that the noise variables do not influence each other ($\text{Cov}(\underline{v}_{\theta_x}(k), \underline{v}_{gyro_y}(k)) = 0$). It is also assumed that the noise does not change over time.
From the measurement signals recorded while the robot is in the resting position, the noise variances $\sigma_{\theta_x}^2$ and $\sigma_{gyro_y}^2$ can be estimated.
Figure 5.43 shows the resting position measurement signal of θ_x.

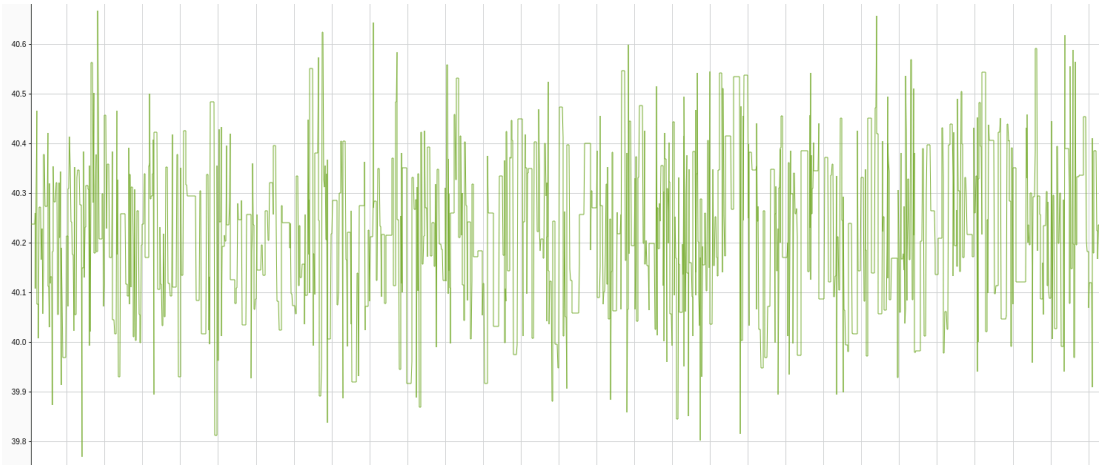


Figure 5.43: Measured noise signal of the tilt angle θ_x

From the signal it is estimated that $\sigma^2_{\theta_x} \approx 0.8\,(^\circ)^2$.
Figure 5.44 shows the resting position measurement signal of $gyro_y$.

Figure 5.44: Measured noise signal of $gyro_y$

From the signal it is estimated that $\sigma^2_{gyro_y} \approx 330\,\mu(^\circ/\mathrm{s})^2$.
From the estimated noise variances, the covariance matrix of the measurement noise $\underline{R}(k)$ is determined.

$$\underline{R}(k) = \mathrm{Var}(v(k)) = \begin{pmatrix} \mathrm{Var}(v_{\theta_x}(k)) & \mathrm{Cov}(v_{\theta_x}(k), v_{\omega_x}(k)) \\ \mathrm{Cov}(v_{\theta_x}(k), v_{\omega_x}(k)) & \mathrm{Var}(v_{\omega_x}(k)) \end{pmatrix} \tag{5.11a}$$

$$= \begin{pmatrix} \sigma^2_{\theta_x} & 0 \\ 0 & \sigma^2_{gyro_y} \end{pmatrix} \approx \begin{pmatrix} 0.8\,(^\circ)^2 & 0 \\ 0 & 330\,\mu(^\circ/\mathrm{s})^2 \end{pmatrix} \tag{5.11b}$$

The parameters $\underline{A}_d$, $\underline{C}$, Q, and $\underline{R}$ are entered into the Kalman filter block [102] in Simulink, as shown in figure 5.45.

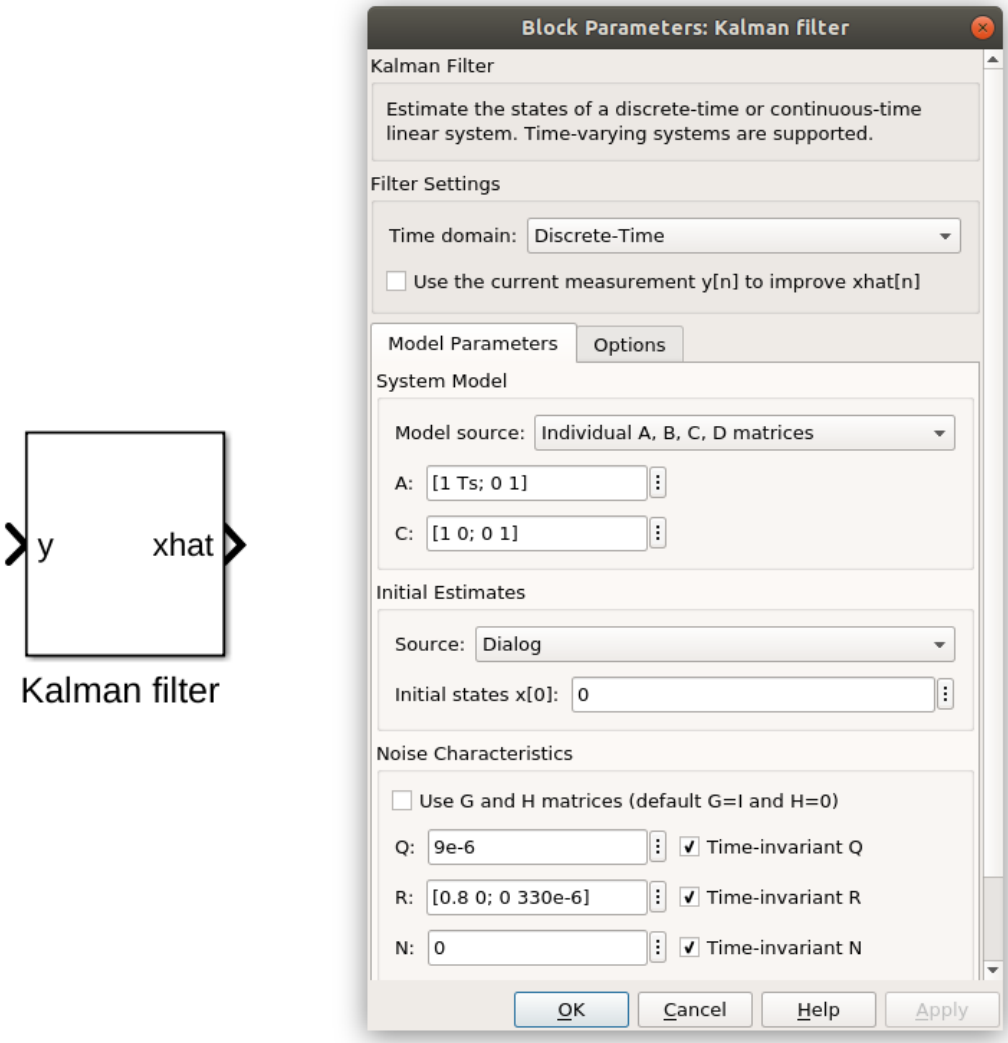


Figure 5.45: Setting up the kalman filter block

Figure 5.46 shows the response (orange) of the Kalman filter to the noisy and disturbed input signal (green).

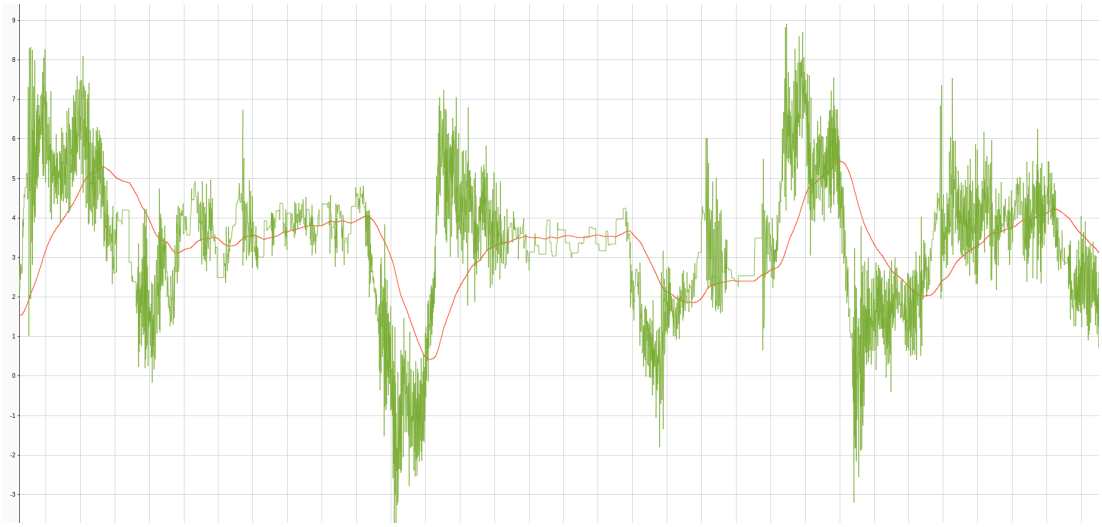


Figure 5.46: Demonstration of the applied Kalman filter

Motor Controller

Figure 5.47 shows the contents of the "Motor Controller" subsystem. The task of the"Motor Controller" subsystem is to control the motor driver.

The subsystem has the two inputs variables, the manipulated variable, and the robot tilt angle θ_x. In the upper third of figure 5.47, the interrupt block of TIM1 is shown. The Function-Call subsystem triggered by the TIM1 interrupt block resets the status interrupt flag by using the TIM_CC_Interrupt_Config_Flag_Reset 5.18 block, and sets the duty cycle of the two PWM outputs by using the TIM_Set_DC 5.16 blocks. The duty cycles of the PWM are set to the amount of the manipulated variable, which is scaled with the factor 100 before. The "Saftey STOP" subsystem checks whether the angle θ_x is greater than 60. If this is the case, a stop signal is passed to the "Motor Driver Logic" subsystems. In addition, the MTR_STBY input of the motor controller is switched to high level, which causes the motor controller to be put into standby mode. The GPIO-write blocks shown on the right are taken from [6].

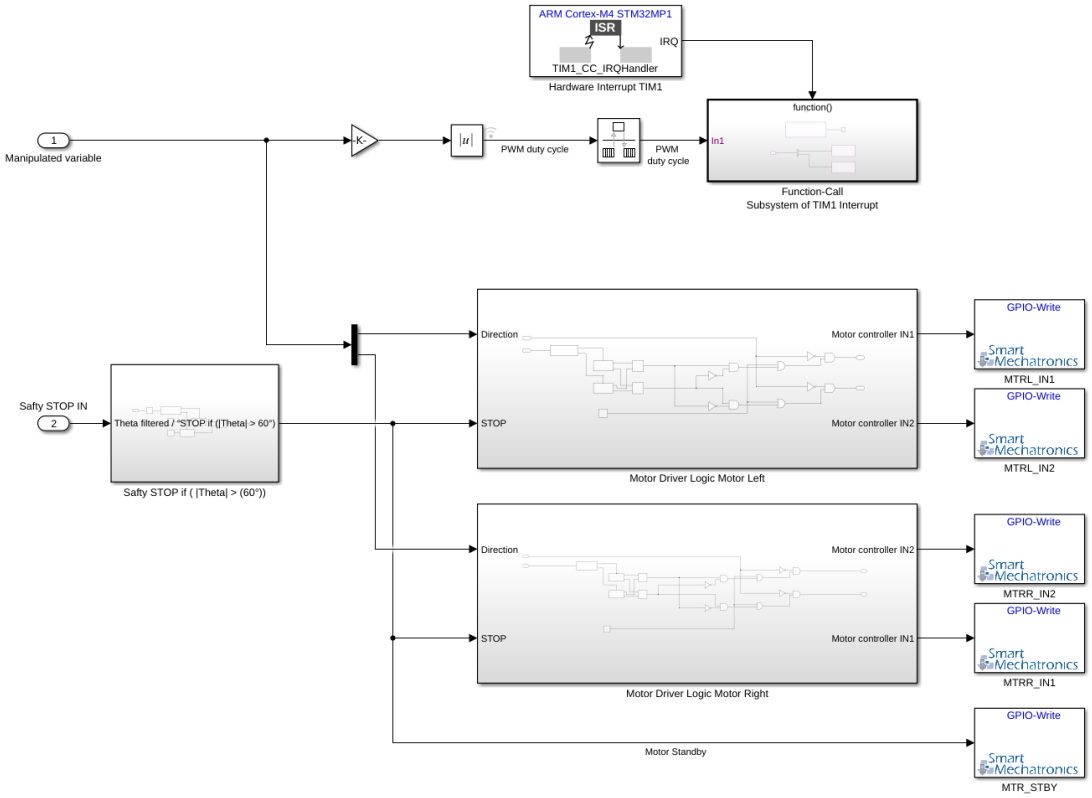


Figure 5.47: Subsystem: “Motor Controller”

The content of the “Motor Driver Logic” subsystems is shown in figure 5.48.

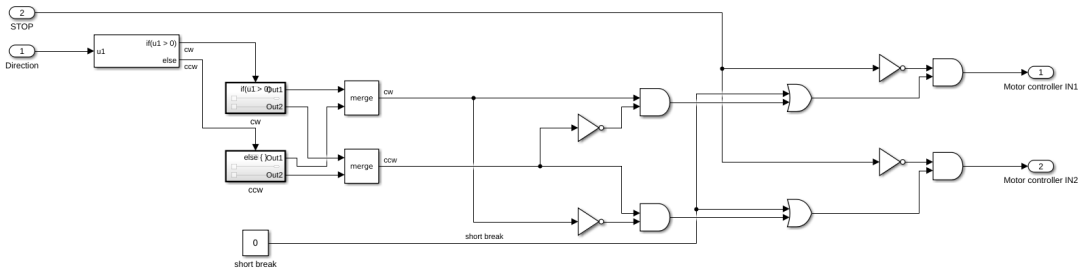


Figure 5.48: Subsystem: “Motor Driver Logic”

The input signals of the “Motor Driver Logic” subsystem are the manipulated variable, which carries the information about the direction of rotation, and the STOP signal. If the direction is greater than zero, the motor rotates clockwise. Otherwise, it rotates counter-clockwise.

The logic shown in figure 5.48 is derived from table 5.8.

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Table 5.8: Hardware-Software Control Function, taken from [95, p. 4]

The subsystems “Motor Driver Logic Motor Left” and “Motor Driver Logic Motor Right” are identical except for the outputs. The outputs of the “Motor Driver Logic Motor Right” subsystem are interchanged so that oppositely mounted motors drive in the same direction.

Ultrasonic sensor

The content of the "Ultrasonic sensor" subsystem is seen in figure 5.49.

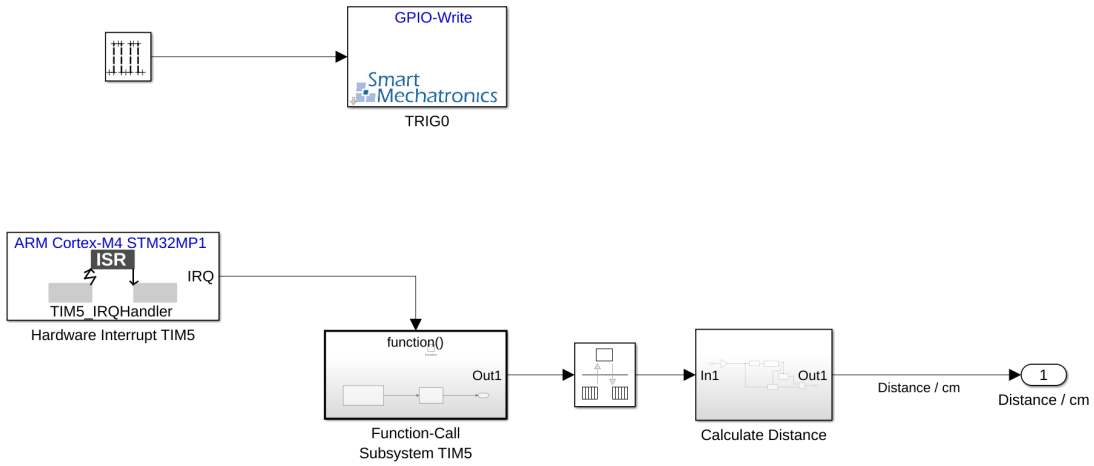


Figure 5.49: Subsystem: "Ultrasonic sensor"

In the upper half, the GPIO-Write block [6] is toggled by a function generator. The configured output of the GPIO-Write block is connected to the TRIG0 pin of the ultrasonic sensor. According to [97], the ultrasonic sensor returns a high-level pulse after the sensor was triggered. The measured distance is calculated from the time taken by the high-level pulse. This is done by equation (5.12):

$$d = \frac{t_{hl} \cdot v_{sound}}{2} \tag{5.12}$$

The distance is represented by d, time of the high-level pulse is represented by t_{hl}, and the sound velocity is represented by v_{sound} (according to [97] $v_{sound} \approx 340\,\mathrm{m\,s^{-1}}$).
The ECHO0 pin, at which the high-level pulse occurs, is connected to the configured input of the TIM5. The interrupt block, corresponding to TIM5, is seen in the lower half of figure 5.49. Inside the Function-Call subsystem of the TIM5 hardware interrupt block, the TIM_Get_Counter block 5.19 is used to reset the interrupt status flag and to return the counter value. The counter value, returned by the TIM_Get_Counter block, is passed to a MATLAB function, which allows every second value to pass. This is necessary because the first of each two interrupts ist triggered by the rising edge of the high-level pule and therefore has no information about the length of the high

level pulse. The output of this MATLAB function is the output of the Function-Caller subsystem. The signal that has the information of the counter value is converted in the subsystem “Calculate Distance” into distance in cm.

To convert the counter value into a distance, the frequency of a counter tick $f_{CK_{CNT}}$ must be known. The prescaler is used to specify that $f_{CK_{CNT}}$ is 1 MHz. Since the clock frequency in front of the prescaler $f_{CK_{PSC}}$ is 208.87 MHz, the prescaler is set to $208-1$ according to equation (A.1). Now, according to equation (5.12), the factor K_{conv} for the conversion from counter value to distance in cm can be determined. The result is $K_{conv} = \frac{1\,\text{cm}}{58}$.

Figure 5.50 shows the required STM32CubeMX configuration.

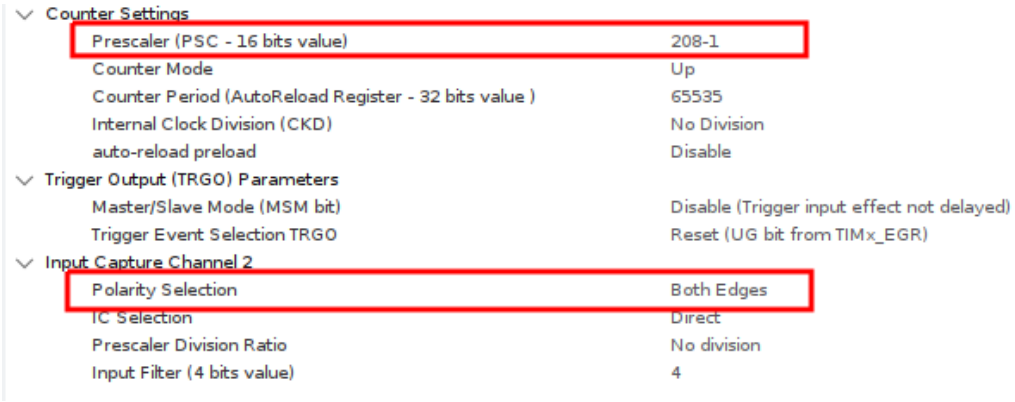


Figure 5.50: STM32CubeMX configuration of TIM5

The selected settings are framed in red.

ADC Battery Voltage

Figure 5.51 shows the contents of the "ADC Battery Voltage" subsystem.

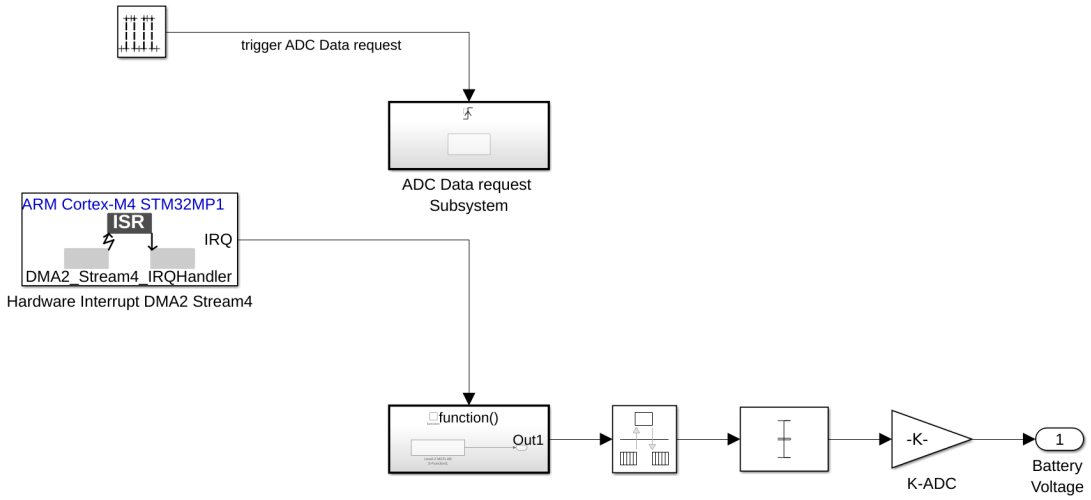


Figure 5.51: Subsystem: "ADC Battery Voltage"

The ADC_DMA_Data_Reqest block 5.27 is a triggered subsystem. This triggered subsystem gets triggered at the rising edge of the function generator. Inside the Function-Caller subsystem of the DMA triggered interrupt, the ADC_DMA_ISR block 5.28 calls the ADC DMA interrupt handler and returns the recorded ADC values to the model. After the rate transitions block a mean value of the recorded ADC values is calculated.

K_{ADC} is calculated from the resistance values R_{46} and R_{47} of the voltage dividers on the self-balancing daughter board [39], the ADC resolution ADC_{res}, which is configured to 16 bits, and the maximum ADC voltage $V_{REF} = 3.3\,\mathrm{V}$ [24]. The calculation is based on Kirchhoff's mesh analysis [103]. Resulting from this the factor K_{ADC} is calculated, seen in equation (5.13).

$$K_{ADC} = \frac{V_{REF}}{ADC_{res}} \cdot \frac{R_{46} + R_{47}}{R_{46}} \approx 192.85\,\mu\mathrm{V} \tag{5.13}$$

The battery voltage V_{DC_IN} is calculated as follows:

$$V_{DC\_IN} = K_{ADC} \cdot ADC\_value \tag{5.14}$$

Figure 5.52 shows the voltage divisor.

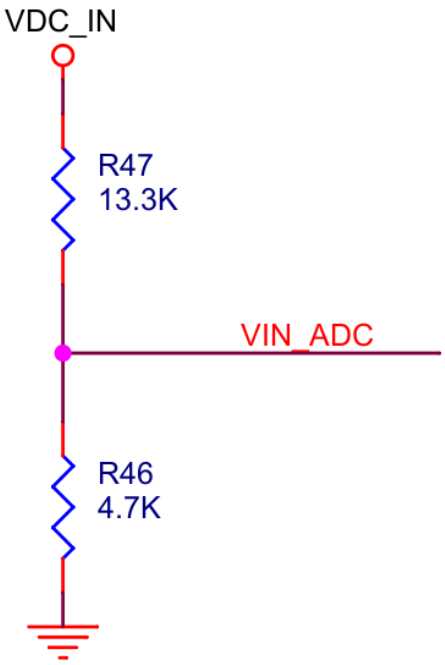


Figure 5.52: Voltage divisor [39, p. 6]

In figure 5.53 the STM32CubeMX setting for the configuration of the ADC is shown.

Figure 5.53: STM32CubeMX configuration of ADC1

In figure 5.54 the STM32CubeMX setting for the configuration of the DMA used with the ADC is shown.

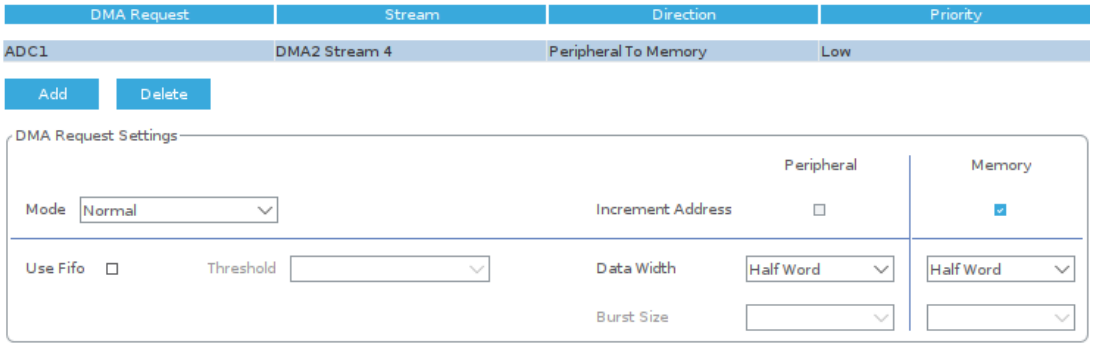


Figure 5.54: STM32CubeMX configuration of the DMA for ADC1

Hall encoders

Figure 5.55 shows the contents of the “Hall encoder” subsystem.

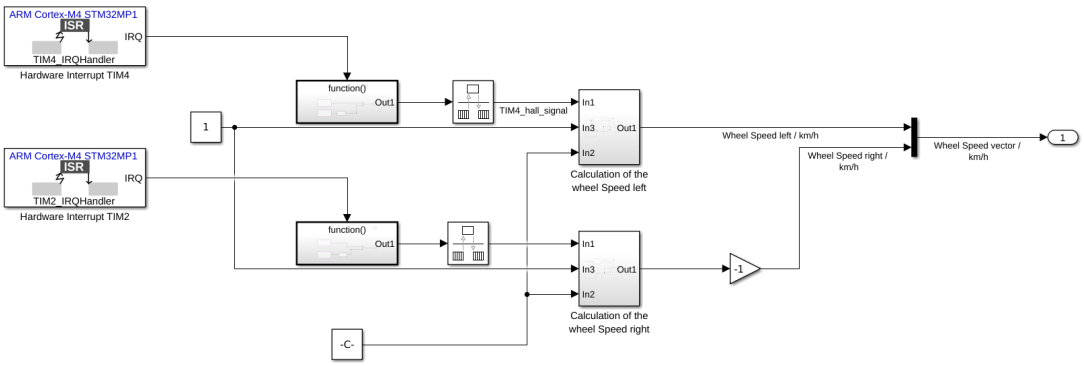


Figure 5.55: Subsystem: “Hall encoders”

In figure 5.55, on the left, you can see the interrupt blocks of the timers configured as input Capture/Compare. Within the Function-Caller subsystem, the TIM_Get_Conter block 5.19 is used to measure the pulse duration at hall encoder signal A. With the GPIO_Get_Input block A.19 the level of the hall encoder signal B is detected. The rotation speed is calculated from the hall encoder signal A. The direction is determined from the hall encoder signal B. The hall encoder signals are shown in figure 5.56.

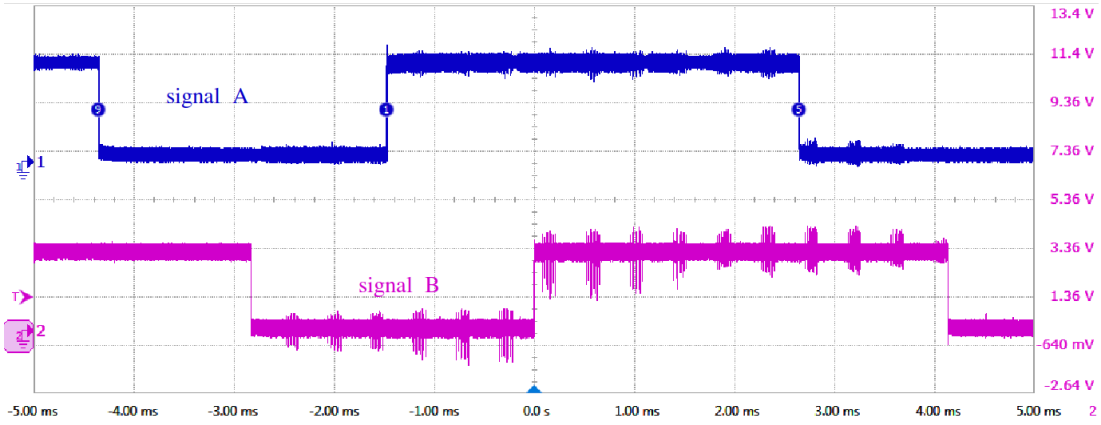


Figure 5.56: Hall encoder signal A and B

The rotation speed and the direction are processed in the “Calculation of the wheel Speed” subsystems. If the signal B is equal 1 a factor of -1 is added to the calculated speed.

The speed of the wheels is calculated by the measured counter value CNT, the maximum motor frequency $\hat{f}_{motor}$ ($\hat{f}_{motor} \approx 4.43\,\mathrm{s}^{-1}$ [96, p 29]), the minimum pulse

duration $\check{t}_{pd}$ of signal A, which is recorded during the maximum motor frequency ($\check{t}_{pd} = 501.57\,\mu$s 4.8), the radius of a wheel r_w ($r_w = 0.031$ m), and the time of a counter increment t_{CNT} must be known. t_{CNT} is set to 1 μs, as described in section 5.7. The timer configuration for this is seen in figure 5.57.
The speed of the wheel, when neglecting the slip, can be calculated as follows:

$$v_{wheel} = \frac{2 \cdot \pi \cdot r_w \cdot \hat{f}_{motor} \cdot \check{t}_{pd}}{(CNT+1) \cdot t_{CNT}} \tag{5.15}$$

The conversion is calculated by the dividend $D_{hall} = v_{wheel} \cdot (CNT+1)$. It is calculated in $\mathrm{km\,h^{-1}}$ by:

$$D_{hall} = \frac{2 \cdot \pi \cdot r_w \cdot \hat{f}_{motor} \cdot \check{t}_{pd}}{t_{CNT}} \approx 1558.04\,\mathrm{km\,h^{-1}} \tag{5.16}$$

The wheel speed right signal is multiplied by -1 because the motors are mounted in the opposite direction. The STM32CubeMX cunfiguration for the timers is shown in figure 5.57.

Figure 5.57: STM32CubeMX configuration of TIM2 and TIM4

Interprocessor communication

The interprocessor communikation is done by the Simulink blocks OpenAMP-Transmit and OpenAMP-Receive taken from [6]. Figure 5.58 shows the transmission

of the signals Theta, Battery Voltage and Distance.

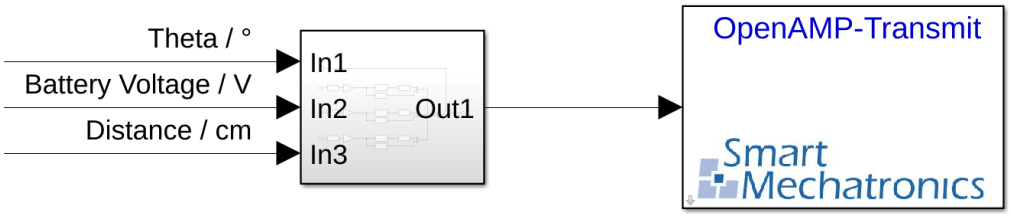


Figure 5.58: Transmitting IPC data

The signals are distributed within the subsystem to an array of the data type `(uint8_t)` and sent to the Cortex-A7 by the OpenAMP-Transmit block.
Figure 5.59 shows the receiving of the data send by the Cortex-A7.

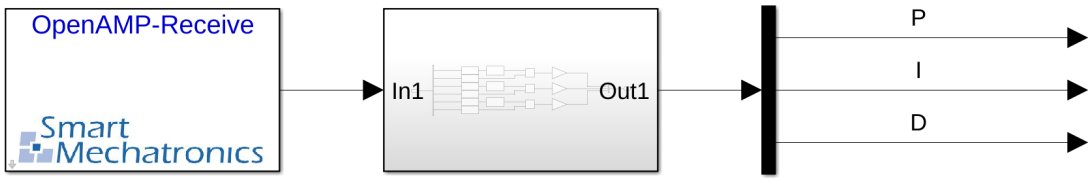


Figure 5.59: Receiving IPC data

The received data array is converted in the subsystem into the 3 mapped signals P, D, and I.

5.8 Control system for the inverted pendulum

In this chapter the main elements of the plant are analyzed. For this purpose, a mathematical description of the inverted pendulum is developed. Based on this mathematical description, the controller is implemented.
An inverted pendulum is a well-known problem in control engineering where the pendulum is held in the forced unstable upper position. [104]
From the physical model of the inverted pendulum, high real-time requirements for the system can be derived. Compared to a stable system, it requires fast and accurate angle

or position sensors. For this reason, the implementation of a controller for an inverted pendulum is suitable to demonstrate the real-time capability of a system. [105]
The model of an inverted pendulum consists of a stem, which stands upright in space, as shown in figure 5.60.

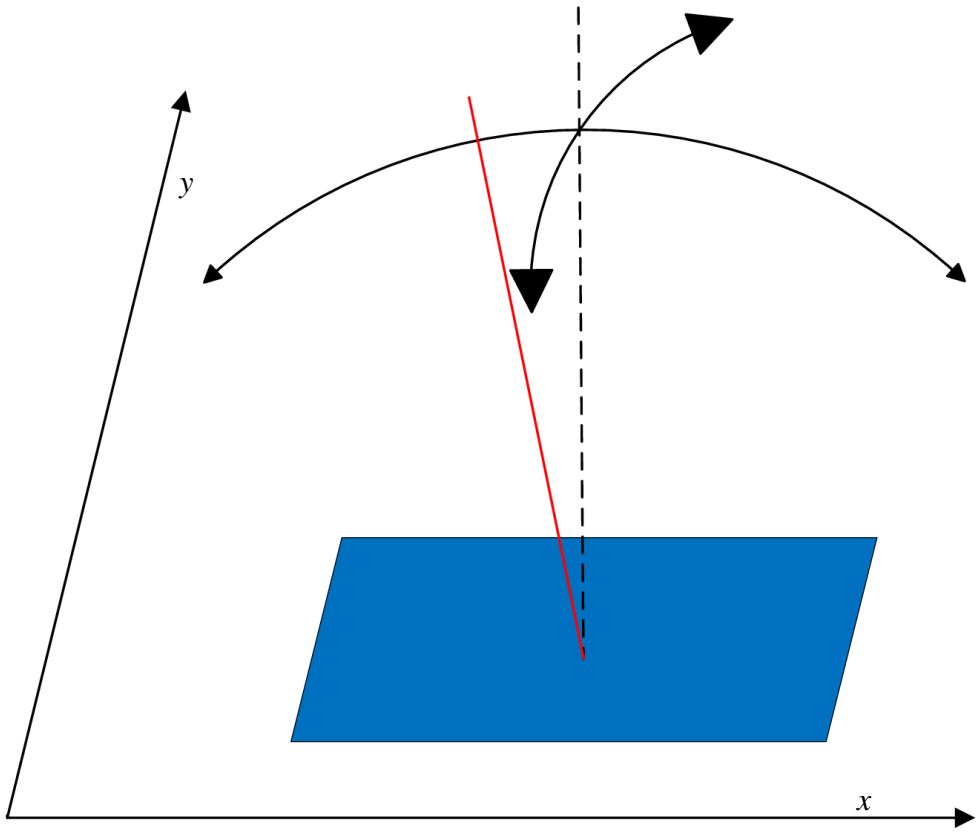


Figure 5.60: Schematic representation of an inverted pendulum in space

Because the bottom point of the stem is movable on the plane, the pendulum tilts when the system is affected by an external force. This can be prevented by a regulation at the installation point of the pendulum stem. [106]

Derivation of the mechanical controlled system

Using the model in figure 5.61, equations of motion for the pendulum are formulated. To determine the plant, the degree of freedom is restricted. This means that the left and right wheel accelerate the robot in a straight line alonge the x-axis. The mathematical description is done without linearization. The mass of the robot is described by m_p. The distance between the axis of the wheels and the center of mass is given as length l. In the figure 5.61, the center of gravity is symbolized as a sphere. In the real system, the mass moment of inertia is determined by the shape of the robot. This shape can be approximated as a cuboid.
Mass moment of inertia for a cuboid according to [107, p. 76]:

$$J_x = \frac{1}{12} m(b^2 + h^2) \tag{5.17}$$

with b as cuboid depth and h as cuboid height. The tilt angle of the robot is seen as θ_x. F_{Hx} and F_{Vx} are horizontal and vertical reaction forces.

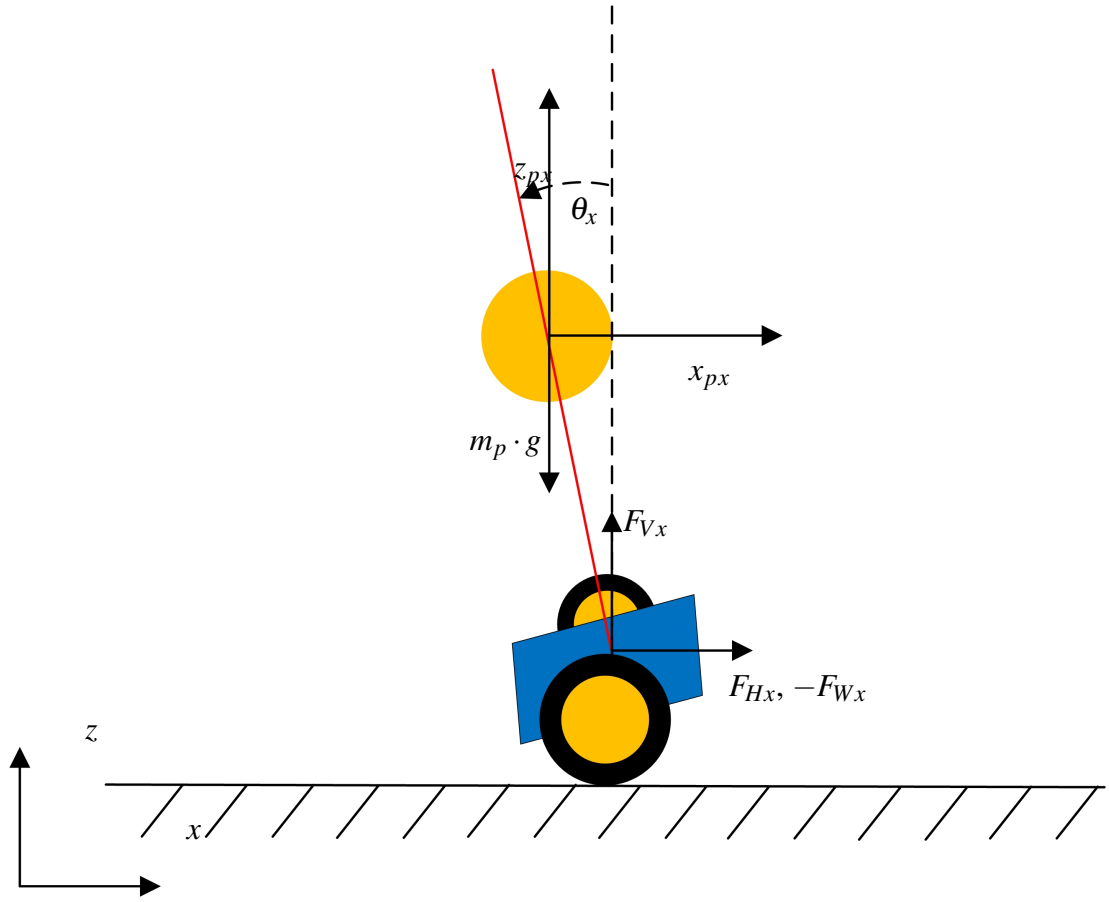


Figure 5.61: Model of the self-balancing robot

Translation:

$$m_w \cdot \ddot{x}_{wx} = F_{Wx} - F_{Hx} \tag{5.18a}$$

$$m_p \cdot \ddot{x}_{px} = F_{Hx} \tag{5.18b}$$

$$m_p \cdot \ddot{z}_{px} = F_{Vx} - m_p \cdot g \tag{5.18c}$$

Rotation:

$$J_x \cdot \ddot{\theta}_x = F_{Vx} \cdot l \cdot \sin\theta_x + F_{Hx} \cdot l \cdot \cos\theta_x \tag{5.19}$$

Geometry:

$$x_{px} = x_{wx} - l \cdot \sin\theta_x \tag{5.20a}$$

$$z_{px} = l \cdot \cos\theta_x \tag{5.20b}$$

In the following, the large-signal performance of the plant is derived.
Derivation of the time-variable function of θ_x:

$$f(t) = \theta_x \tag{5.21a}$$

$$f'(t) = \frac{d\theta_x}{dt} = \dot{\theta}_x \tag{5.21b}$$

$$f''(t) = \frac{d^2\theta_x}{dt^2} = \ddot{\theta}_x \tag{5.21c}$$

Derivation of the time-variable function of $\sin(\theta_x)$:

$$h(t) = \sin(\theta_x) \tag{5.22a}$$

$$h(t) = \sin(f(t)) \tag{5.22b}$$

$$h'(t) = \cos(f(t)) \cdot f'(t) \tag{5.22c}$$

$$h''(t) = \cos(f(t)) \cdot f''(t) + f'(t) \cdot f'(t) \cdot (-\sin(f(t))) \tag{5.22d}$$

$$h''(t) = \cos(\theta_x) \cdot \ddot{\theta}_x - \dot{\theta}_x^2 \cdot \sin(\theta_x) \tag{5.22e}$$

Derivation of the time-variable function of $\cos(\theta_x)$:

$$g(t) = \cos(\theta_x) \tag{5.23a}$$

$$g(t) = \cos(f(t)) \tag{5.23b}$$

$$g'(t) = -\sin(f(t)) \cdot f'(t) \tag{5.23c}$$

$$g''(t) = -\sin(f(t)) \cdot f''(t) + f'(t) \cdot f'(t) \cdot (-\cos(f(t)) \tag{5.23d}$$

$$g''(t) = -\sin(\theta_x) \cdot \ddot{\theta}_x - \dot{\theta}_x^2 \cdot \cos(\theta_x) \tag{5.23e}$$

The time derivatives are inserted into the reflection forces.

For F_{Hx} considering that the geometry equation (5.20a) and the derivation equation (5.22e) is used:

$$F_{Hx} = m_p \cdot \ddot{x}_{px} \tag{5.24a}$$

$$F_{Hx} = m_p \cdot \frac{d^2 x_{px}}{dt^2} \tag{5.24b}$$

$$F_{Hx} = m_p \cdot \frac{d^2}{dt^2}(x_{wx} - l \cdot \sin(\theta_x)) \tag{5.24c}$$

$$F_{Hx} = m_p \cdot \ddot{x}_{wx} - m_p \cdot l \cdot \ddot{\theta}_x \cdot \cos(\theta_x) + m_p \cdot l \cdot \dot{\theta}_x^2 \cdot \sin(\theta_x) \tag{5.24d}$$

For F_{Vx} considering that the geometry equation (5.20b) and the derivation equation (5.23e) is used:

$$F_{Vx} = m_p \cdot \ddot{z}_{px} + m_p \cdot g \tag{5.25a}$$

$$F_{Vx} = m_p \cdot \frac{d^2 z_{px}}{dt^2} + m_p \cdot g \tag{5.25b}$$

$$F_{Vx} = mp \cdot l \cdot \frac{d^2}{dt^2} \cos(\theta_x) + m_p \cdot g \tag{5.25c}$$

$$F_{Vx} = -m_p \cdot l \cdot \ddot{\theta}_x \cdot \sin(\theta_x) - m_p \cdot l \cdot \dot{\theta}_x^2 \cdot \cos(\theta_x) + m_p \cdot g \tag{5.25d}$$

Translational movement: The reaction forces equation (5.24d) is inserted into the translation equation (5.18a).

$$\ddot{x}_{wx}(m_w + m_p) = F_{wx} + m_p \cdot l \cdot \ddot{\theta}_x \cdot \cos(\theta_x) - m_p \cdot l \cdot \dot{\theta_x}^2 \cdot \sin(\theta_x) \tag{5.26}$$

Rotational movement: The reaction forces equation (5.24d) and equation (5.25d) are inserted into the rotation equation (5.19).

$$\begin{aligned} J_x \cdot \ddot{\theta}_x = (m_p \cdot l)(&-l \cdot \ddot{\theta}_x \cdot \sin^2(\theta_x) - l \cdot \dot{\theta_x}^2 \cdot \cos(\theta_x) \cdot \sin(\theta_x) + g \cdot \sin(\theta_x) + \\ &\ddot{x}_{wx} \cdot \cos(\theta_x) - l \cdot \ddot{\theta}_x \cdot \cos^2(\theta_x) + l \cdot \dot{\theta_x}^2 \cdot \sin(\theta_x) \cdot \cos(\theta_x)) \end{aligned} \tag{5.27}$$

Using the Trigonometric Pythagoras equation (5.28a) and the Addition Theorem equation (5.28b) [108, p. 94]

$$\sin^2(x) + \cos^2(x) = 1 \tag{5.28a}$$

$$\sin(x_1) \cdot \cos(x_2) - \cos(x_1) \cdot \sin(x2) = \sin(x_1 - x_2) \tag{5.28b}$$

equation (5.27) is solved according to equation (5.29).

$$\ddot{\theta}_x(m_p \cdot l^2 + J_x) = m_p \cdot g \cdot l \cdot \sin(\theta_x) + m_p \cdot \ddot{x}_{wx} \cdot l \cdot \cos(\theta_x) \tag{5.29}$$

After all, kinematic relations of the inverted pendulum are described, the functional equation $\theta_x = f(F_{wx})$ is tried to be set up. In this case, the equations of the translational and the rotational motion with their $\sin(\theta_x)$ and $\cos(\theta_x)$ terms show nonlinear behavior, so it is necessary to eliminate their $\sin(\theta_x)$ and $\cos(\theta_x)$ terms first.
For this purpose, a linearization is performed. The linearization proceeds as follows: A proportionality coefficient K_p for the operating point is formed as a result of the linearization. For this purpose, the non-linear element is converted into an equation of the line with the aid of the Taylor series. For this purpose, the Taylor series is terminated after the first theorem. The result is a linear equation through the operating point. [35]

Taylor series [108, p. 182]

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \tag{5.30}$$

For the therm $\sin(x)$ working at the operating point $x = 0$:

$$f(x) = \frac{\sin(0)}{0!}(x-0)^0 + \frac{\cos(0)}{1!}(x-0)^1 \tag{5.31a}$$

$$f(x) = x \tag{5.31b}$$

It results that the proportionality coefficient $K_{p_{\sin(\theta_x)}} = \theta_x$.

For the term $\cos(x)$ working at the operation point $x = 0$:

$$f(x) = \frac{\cos(0)}{0!}(x-0)^0 + \frac{-\sin(0)}{1!}(x-0)^1 \tag{5.32a}$$

$$f(x) = 1 \tag{5.32b}$$

It results that the proportionality coefficient $K_{p_{\cos(\theta_x)}} = 1$.

Moreover, the assumption is made that potencies of θ_x or the derivatives of θ_x with higher exponent than 1 can be set to 0, since their value, around the operating point $\theta_x \approx 0$ are negligibly small.

After eliminating the nonlinear terms and the negligible potencies, equation (5.26) and equation (5.29) yield the following equations:

$$\ddot{x}_{wx}(m_w + m_p) = F_{wx} + m_p \cdot l \cdot \ddot{\theta}_x \tag{5.33a}$$

$$\ddot{\theta}_x(m_p \cdot l^2 + J_x) = m_p \cdot g \cdot l \cdot \theta_x + m_p \cdot l \cdot \ddot{x}_{wx} \tag{5.33b}$$

Now the relationship between θ_x and the force F_{wx} is determined by inserting in $\ddot{x}_{wx}$.

Afterwards for $\ddot{\theta}_x$ the difference theorem for the second derivative is applied. [35, p. 67] Then the transfer function of the pendulum can be determined by substituting.

$$G_s(s) = \frac{b_0}{s^2 \cdot a_2 + s \cdot a_1 + a_0} \tag{5.34a}$$

$$b_0 = \frac{1}{g \cdot (m_w + mp)} \tag{5.34b}$$

$$a_0 = -1 \tag{5.34c}$$

$$a_1 = 0 \tag{5.34d}$$

$$a_2 = \frac{1}{g}(l + \frac{b^2}{12 \cdot l} + \frac{l}{12} - \frac{1}{m_w + m_p}) \tag{5.34e}$$

After the transfer function has been formulated, the stability of the system can be investigated.

For this purpose, the characteristic equation of the system is examined. The system has an unstable resting position if at least one zero of the characteristic equation does not have a negative real part. [35, p. 871]

$$s^2 \cdot \frac{1}{g}(l + \frac{b^2}{12 \cdot l} + \frac{l}{12} - \frac{1}{m_w + m_p}) - 1 = 0 \tag{5.35a}$$

$$s_1 = + \sqrt{\frac{1}{\frac{1}{g}(l + \frac{b^2}{12 \cdot l} + \frac{l}{12} - \frac{1}{m_w + m_p})}} \tag{5.35b}$$

$$s_2 = - \sqrt{\frac{1}{\frac{1}{g}(l + \frac{b^2}{12 \cdot l} + \frac{l}{12} - \frac{1}{m_w + m_p})}} \tag{5.35c}$$

Equation (5.35b) shows that one of the zeros of the characteristic equation has a positive real part. So the system has an unstable resting position.

Since G_s is only the transfer function of the mechanical tilt moment, many unknown transfer functions remain in the controlled system. The unknown transfer functions are highlighted in gray in figure 5.62. The reference variable w determines how the angle θ_x should be. It is compared with the measured angle θ_x at the addition point. From the control difference $(w - x)$ the manipulated variable y is calculated by the

controller. The manipulated variable y is superimposed by the disturbance variable $z1$. In figure 5.62 the controlled system is the multiplication of G_{Driver}, G_{Motor}, and G_s. The controlled system outputs the controlled variable x. The controlled variable is fed back via the measuring device G_{Sensor}. [35]

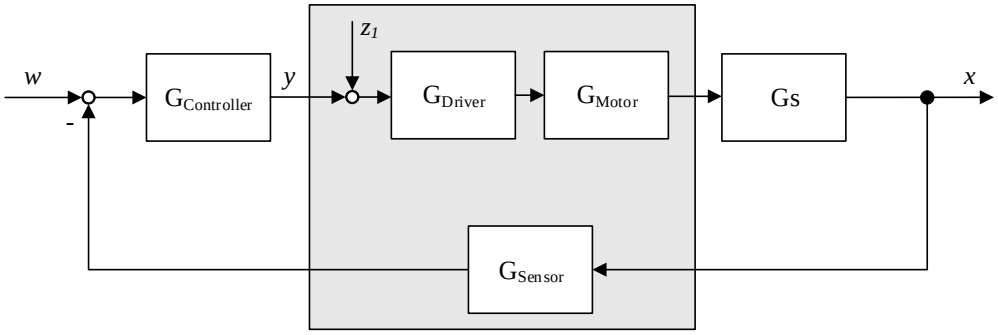


Figure 5.62: System overwiew of the self-balancing robot

Experimental analysis of the overall control system

An experimental analysis of the entire system should help to determine the transfer function of the whole system, in order to subsequently dimension a controller for this system. In the expirimental analysis of systems, a step function is applied to the system at rest, while the output function of the system is measured. Afterwards the non-parametric system can be transformed into a parametric system. The result is the transfer function of the entire system. [35]

Figure 5.63 shows a schematic diagram of how the experimental analysis is performed. The step function is applied to y, while the output function is recorded at $x_{measured}$.

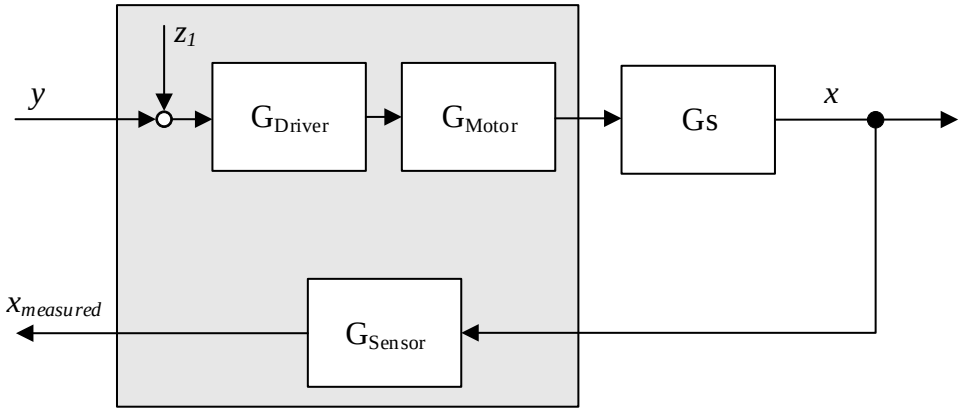


Figure 5.63: Set up to record the step response

The self-balancing robot is held in the unstable rest position while a step function with the peak value 0.1 is applied to the system as manipulated variable y. The manipulated variable 0.1 ensures that the PWM is set to a duty cycle of 10 % and the robot starts moving in one direction. Since the system is not controlled, the robot tips over. During this, the angle θ_x is recorded. The resulting response $x_{measured}$ = System_out and the step function = System_in are shown in figure 5.64. The step function is shown on a scale of 100:1.

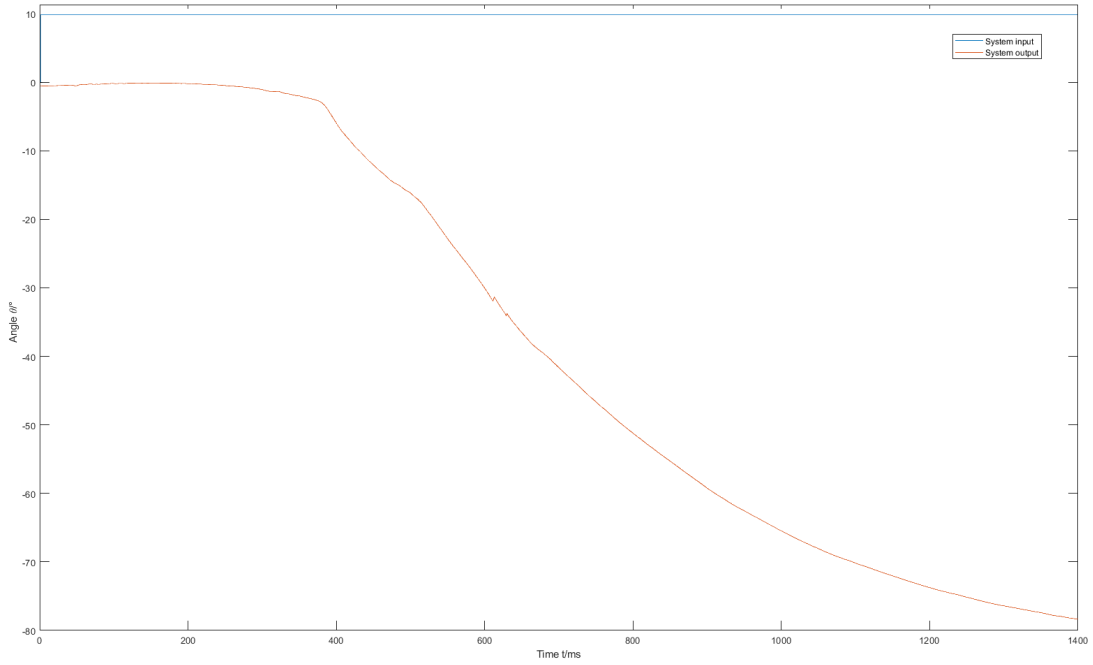


Figure 5.64: Step response of the System recorded at a step sie of 1 ms

The MATLAB “System Identification Toolbox” [109] is used to create a transfer function from the step response. The window of the toolbox is seen in figure 5.65.

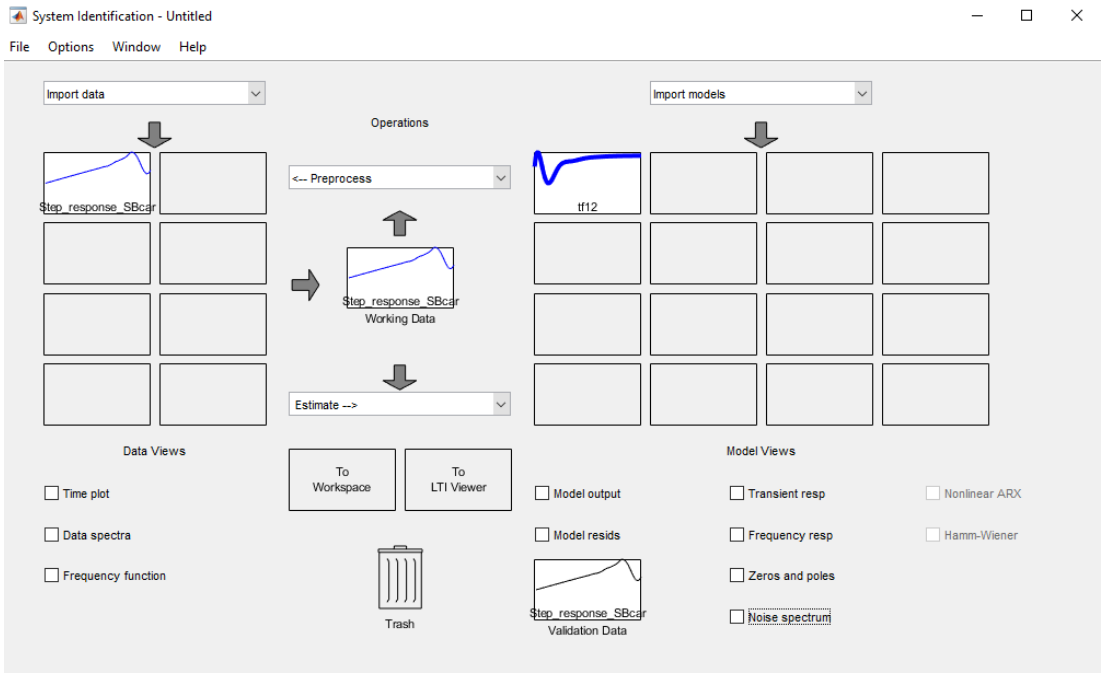


Figure 5.65: Window of the “System Identification Toolbox”

Creating the transfer function is done by importing the step function and the system response in the “System Identification Toolbox”, seen in figure 5.66.

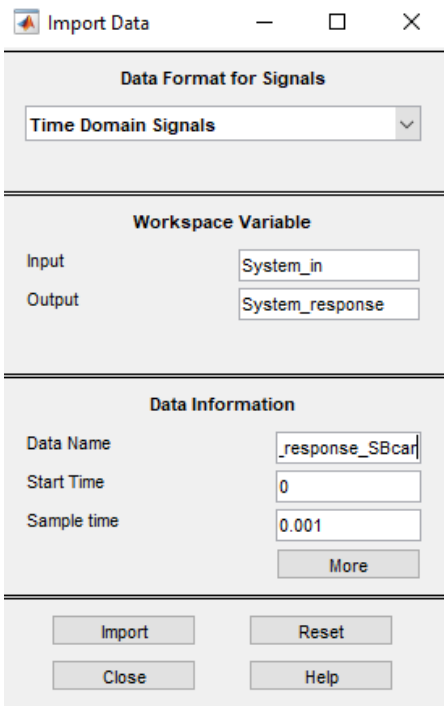


Figure 5.66: Importing data to the "System Identification Toolbox"

In the next step, it is necessary to specify how many poles and zeros are to be calculated by the "System Identification Toolbox". To determine this, the system from figure 5.63 is summed up as described in [35] to form a transfer function. This transfer function $G_{Response}$ can be seen in equation (5.38).

$$G_{Response}(s) = G_{Driver}(s) \cdot G_{Motor}(s) \cdot G_s(s) \cdot G_{Sensor}(s) \tag{5.36}$$

It is estimated that $G_{Response}(s)$ can be approximated by a transfer function with 3 poles and 2 zeros. This is specified in the window shown in figure 5.67.

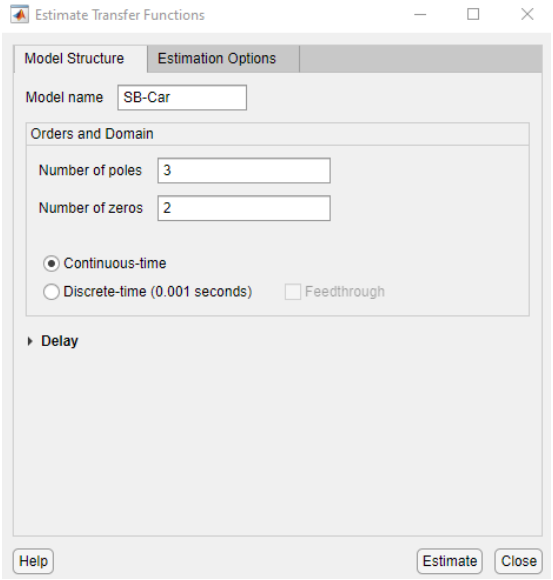


Figure 5.67: Estimation of the number of poles and zeros of the transfer function

The transfer function determined by the "System Identification Toolbox" can be seen in equation (5.38).

$$G_{Response}(s) = \frac{-493 \cdot s^2 + 8074 \cdot s - 5.9 \cdot 10^4}{s^3 + 7.232 \cdot s^2 + 50.97 \cdot s + 58.3} \tag{5.37}$$

In figure 5.68, the recorded step response (green) and the step response of the calculated transfer function (blue) can be seen.

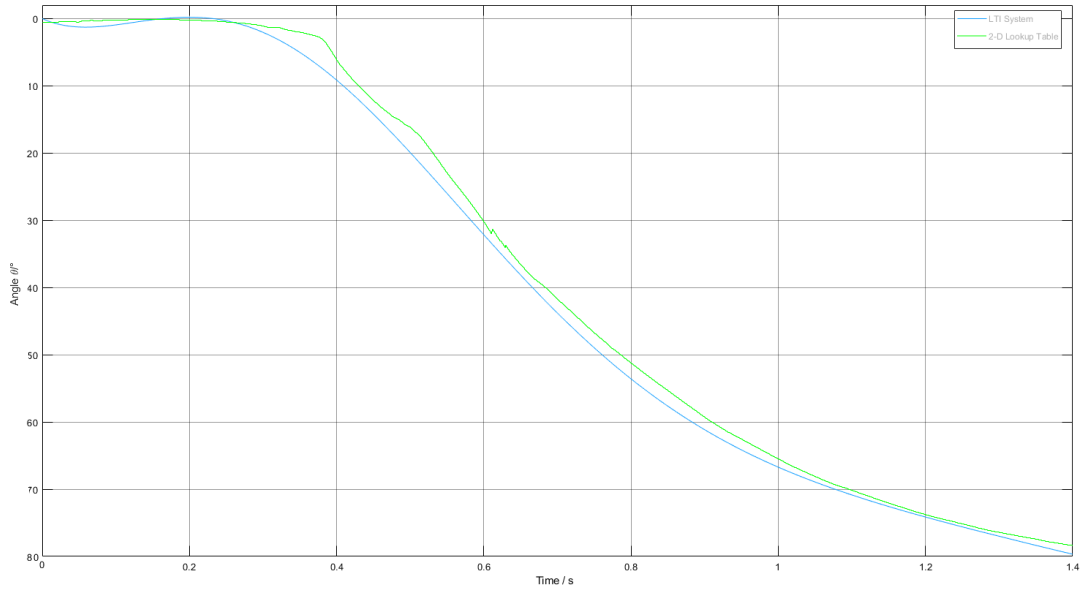


Figure 5.68: Comparison between the recorded step response of the real system (green) and the one calculated by the "System Identification Toolbox" (blue)

To compare the step responses of the derived transfer function with the step responses shown in figure 5.68, the gravity, the geometry parameters, and the masses of the self-balancing robot are inserted into the equation (5.34a). To make a comparison, the derived function must be multiplied by a factor $K_{RAD\_to\_DEG} = \frac{180}{\pi}$ to convert radians into degree. The parameters insertet in the eqation are $g = -9.81\,\mathrm{m\,s^{-2}}$, $m_w = 0.090\,\mathrm{kg}$, $mp = 0.706\,\mathrm{kg}$, $b = 0.070\,\mathrm{m}$, and $l = 0.1485\,\mathrm{m}$.
The resulting transfer function is seen in equation (5.38).

$$G'_s(s) = K_{RAD\_to\_DEG} \cdot G_s(s) = \frac{-23.05}{0.3499 \cdot s^2 - 3.142} \tag{5.38}$$

If the transfer function $G'_s(s)$ is loaded with a step function that has a peak value of 2.118, it approaches the measured and the calculated step response. Seen in figure 5.69.
This comparison is shown in figure 5.69. The recorded step response is plotted in green, the step response of the transfer function calculated by the "System Identification Toolbox" in blue, and the step response of the derived transfer function in red.

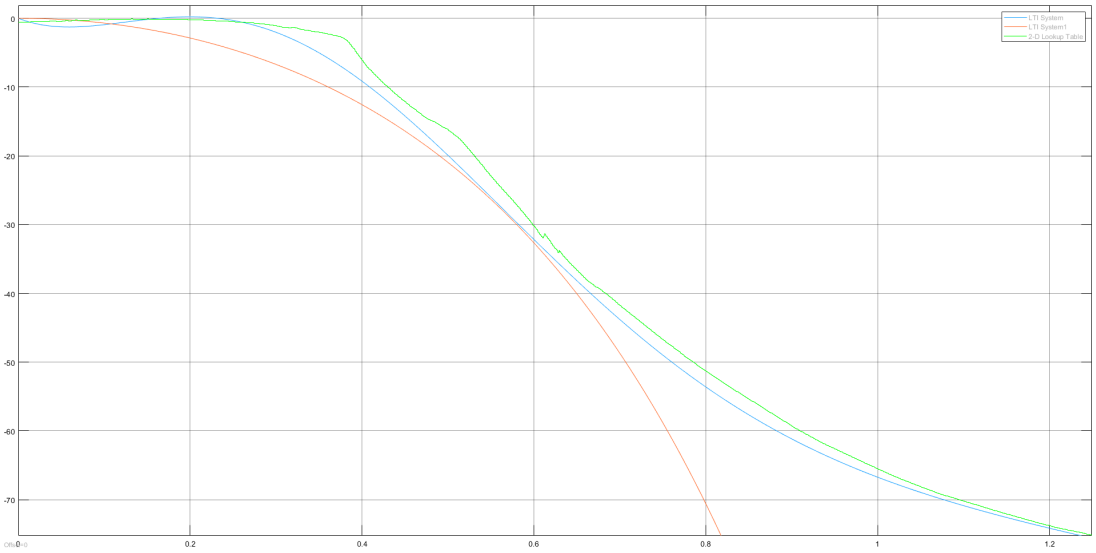


Figure 5.69: Comparison between the step response recorded of the real system (green), the one calculated by the "System Identification Toolbox" (blue), and the one derivated by hand for the mechanical tilting action (red)

Figure 5.69 verifies the recorded transfer function and the transfer function calculated by the "System Identification Toolbox" with the derived transfer function of the inverted pendulum.

To determine the minimum required sample time for quasi-analog system control, the step response of the transfer function has to be analyzed. [35]

For this purpose, the transfer function calculated with the "System Identification Toolbox", is used to determine the delay time T_u, the compensation time T_g and the settling time T_{95}. Then, the minimum required sampling time T is determined using the table [35, p. 498]. The line in the table [35, p. 498] where $T_g \geq 10$ applies to the analyzed system. It follows according to the table [35, p. 498] that $T \leq 0.1 * T_g$ must be selected. The step function and the determination of its characteristic values are shown in figure 5.70.

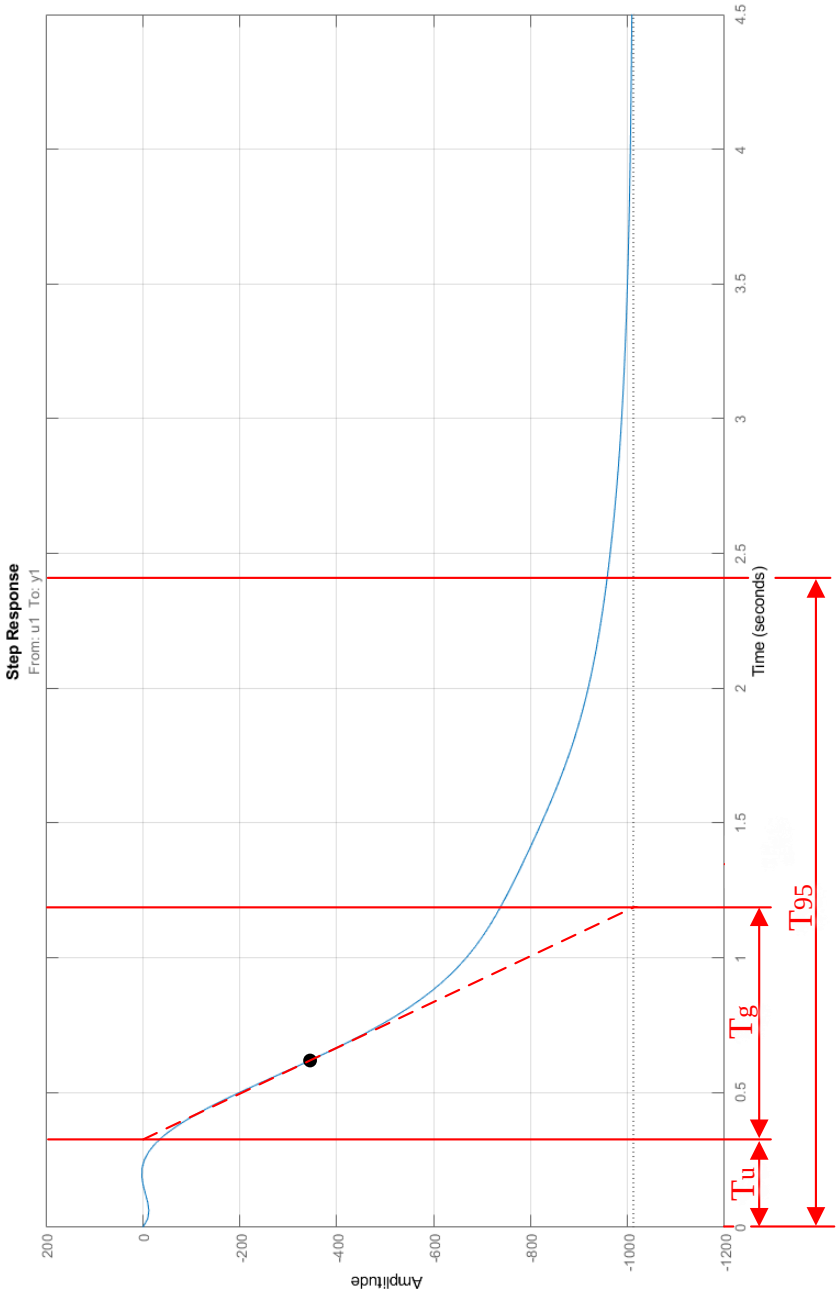


Figure 5.70: Determination of the sampling time, from characteristic values of the controlled system

The values $T_u \approx 0.3\,\text{s}$, $T_g \approx 0.9\,\text{s}$, and $T_{95} \approx 2.4\,\text{s}$ are read off.
It follows that the minimum required sample time T for quasi-analog control must be $T \leq 90\,\text{ms}$.
Once the minimum sample time is checked, it is considered in which period new gyro and acceleration values are recorded. From table 4.8, it is seen that the sensor values are read in a minimum period of $997.62\,\mu\text{s}$. To process each set of sensor values in about a separate model step the sample time is set to 90 times the minimum required sample time ($T = 1\,\text{ms}$).
The "Control System Toolbox" [110] is used to design a controller for the Simulink model shown in figure 5.71. In the model the transfer function calculated with the "System Identification Toolbox", is used as plant. The two step functions represent a 1 ms puls with size 1 to disturb the plant. The plant is sampled with $T = 1\,\text{ms}$ by the "Discretization" subsystems. The Controller C calculated by the "Control System Toolbox" must regulate the disturbing pulse. The transfer function of the controller used in the following figures can be seen in equation (5.39).

$$C = \frac{-0.0014829(s+0.6399)}{s} \tag{5.39}$$

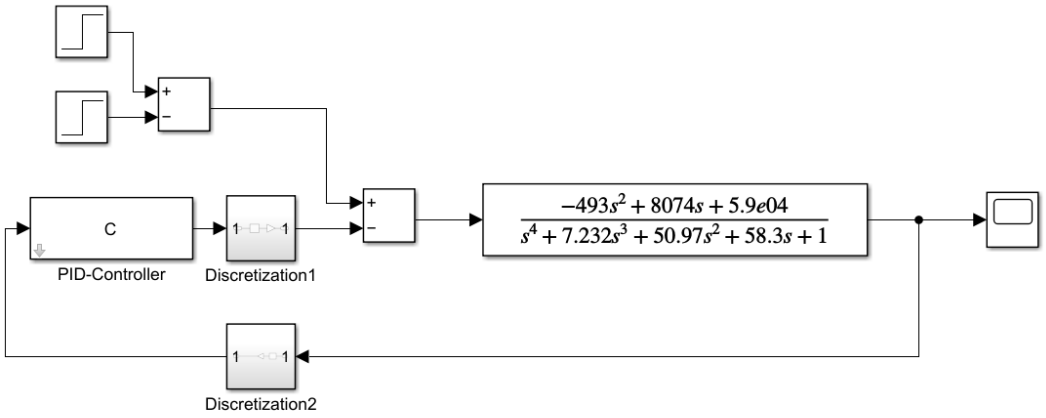


Figure 5.71: Controller design in MATLAB Simulink

Figure 5.72 shows the "Control System Toolbox" during the design of the Controller C.

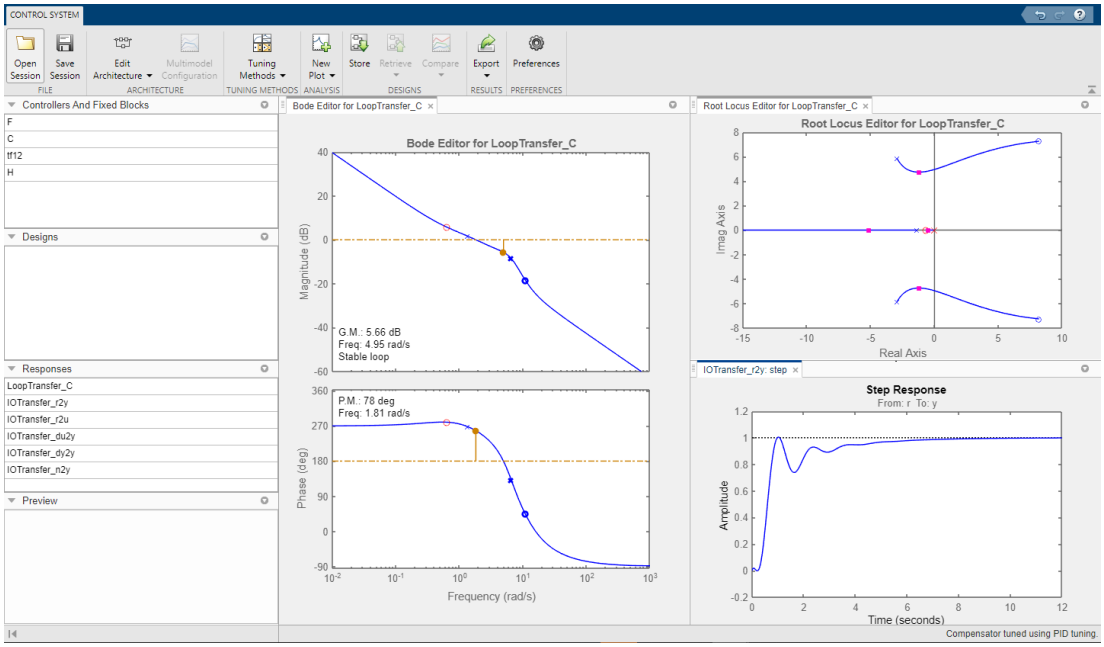


Figure 5.72: “Controll Syste Design Toolbox”

Figure 5.73 shows the system response to the disturbance pulse.

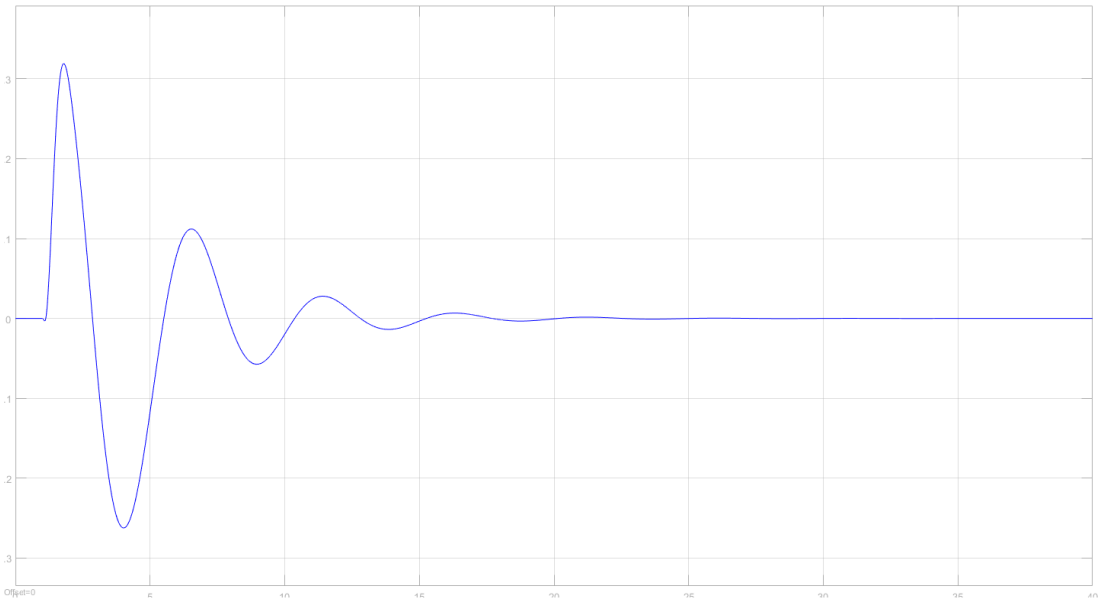


Figure 5.73: Controller design in MATLAB Simulink

It can be seen that controller C can regulate the disturbance pulse.

A PID controller for the self-balancing car can also be determined empirically, according to [111]. This can be done in the real system using the external mode. The PD controller parameters found in this process are $P = 0.0196$ $D = 0.0187$, and $N = 28.234$.

5.9 Implementation of non real-time application

Because the implementation of the non-real-time application is not the focus of this thesis, it will only be touched upon.

To continue using the external mode via XCP on TCP/IP, the application “External_mode” forms the fundament of the non real-time application. The implementation of the graphical components of the non real-time application are performed with the use of the Light Versantil Graphics Library (LVGL) [112]. LVGL is selected, because it has been used in a previous project. The LVGL supports the GIMP-Toolkit (GTK) driver [113] (GNU Image Manipulation Program (GIMP), GNU s Not UNIX (GNU)). The GTK driver is included in the “st-image-westone” [114] OS, which is running on the Cortex-A7.

The application created is divided into two pages. The page “Slider”, seen in figure 5.74, shows 3 sliders to tune “P”, “I”, and “D” parameter of the controller. The page “Diagram”, seen in figure 5.75, monitors “Theta”, “Battery Volage”, and “Distance”.

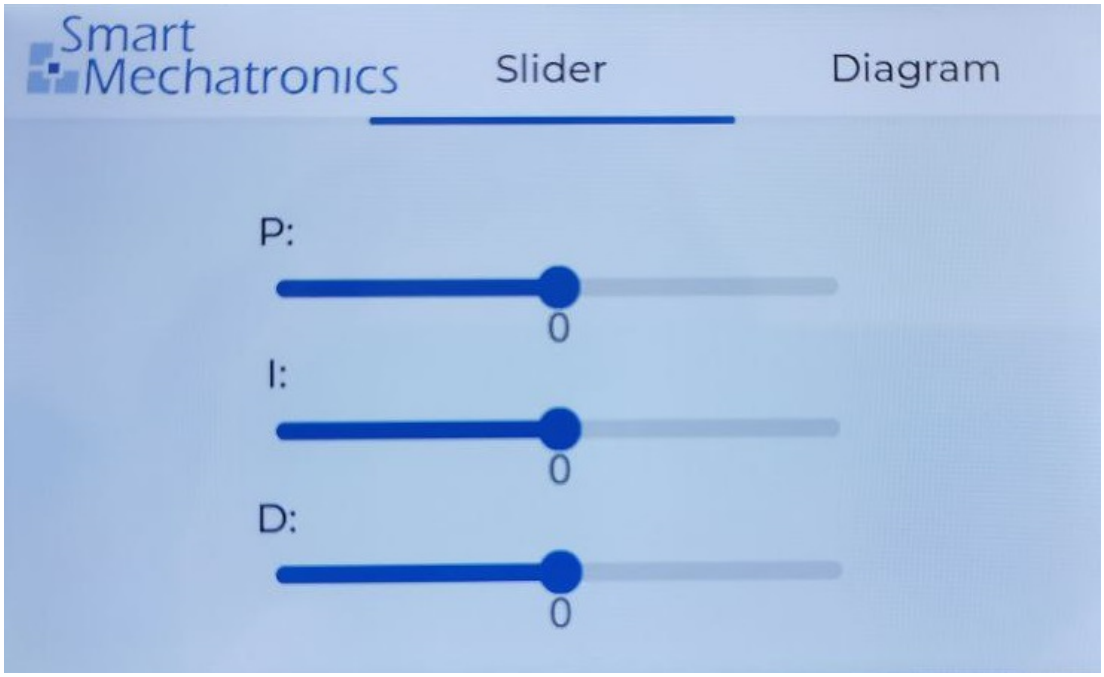


Figure 5.74: Non real-time application: “Slider” page

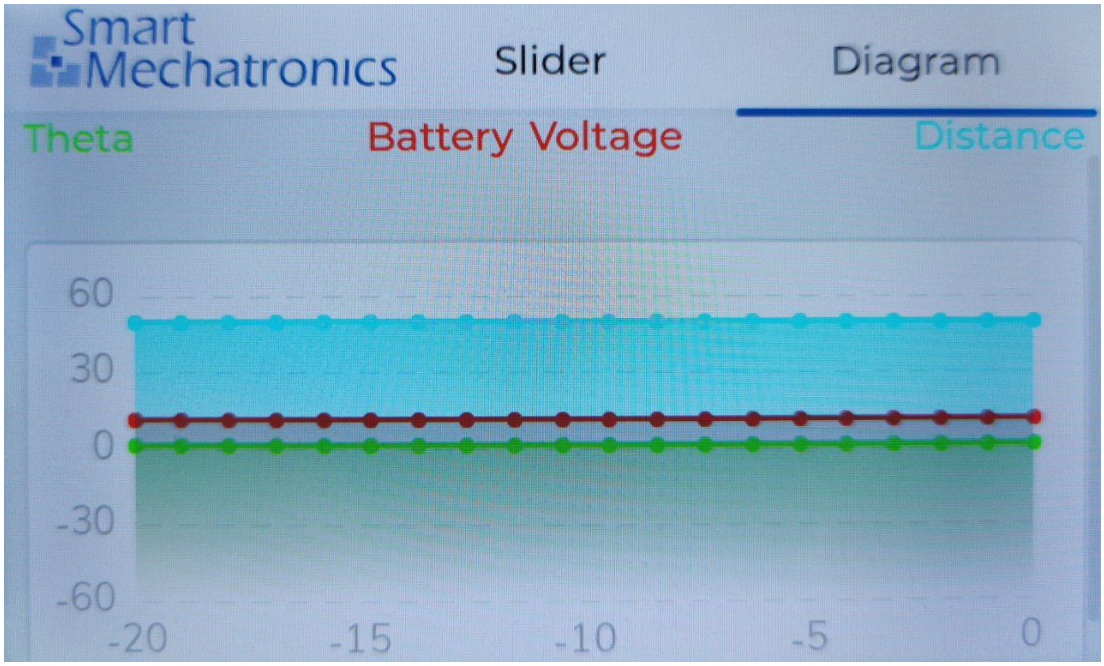


Figure 5.75: Non real-time application: “Diagram” page

6 Verification

The requirements specifications defined in item A.1.1 are checked by the following verification plan. The verification procedure with the corresponding tools is also described in the plan.

The instrument used for the time-critical measurements during verification is the oscilloscope shown in table 4.7.

The main measurements are described under table 6.1.

No./ID	Non-technical title	Verification of the requirement	Tools
Req_01	Code generation	It is verified, whether the generated code can be integrated into a STM32CubeMX project. It is verified whether the firmware compiled and linked from the code generated by using the developed coder target within Simulink, and the code of the STM32CubeMX project, can be executed on the Cortex-M4 core. It is verified whether the firmware can be operated with a sample time of 100 μs. It is verified whether the model running on the Cortex-M4 outputs the correct values corresponding to the Simulink model. It must be verified that the model step is called by the timer interrupt. Results: The generated code can be integrated into a STM32CubeMX project. The firmware compiled from the generated code and the STM32CubeMX can be executed on the Cortex-M4. The firmware can operate with a sample time of 100 μs. (Sample times of 20.83 μs = 48 kHz can be achieved) The model that is executed on the hardware returns the correct values. The model step is called by the TIM7 interrupt. **Tests passed.**	STM32MP1, Computer, MATLAB Simulink, Embedded Coder, Debug-tool, Build environment for cross-compiling, Oscilloscope, External mode
Req_02	External mode via XCP	It is verified whether the external mode via XCP provides a correct data transfer between the Cortex-M4 and the development computer. Result: The External mode via XCP transfers the data correctly. **Test passed.**	STM32MP1, Computer, MATLAB Simulink, Embedded Coder, Build environment for cross-compiling, Network communication

No./ID	Non-technical title	Verification of the requirement	Tools
Req_03	Set up project build environment	It is verified whether the set up build project can compile and link the generated code from MATLAB Simulink and STM32CuebMX. Result: The developed build project can compile and link an executable frimware from the available code sources (`*.c,*.h,*.s,*.ld`). **Test passed.**	Computer, arm-none-eabi-gcc cross-compiler, CMake
Req_04	Acceleration and gyro data	It is verified whether the MPU6500 can be implemented via Simulink blocks, using the codegeneration for the Cortex-M4 core. It is verified whether the register data of the MPU6500 are read out from the sensor via SPI. It is verified whether the acceleration and gyro measurement data of the MPU6500 is read out within 500 μs. It is verified whether the sensor outputs a tigger signal. It is verified that the Hardware Abstraction Layer (HAL) is not used. Results: The MPU6500 can be implemented by developed Simulink blocks, using the codegeneration for the Cortex-M4 core. The register data of the MPU6500 are read out via SPI. The acceleration and gyro measurement data of the MPU6500 are read out within 500 μs. The MPU6500 outputs a trigger signal. The HAL has not been used. **Tests passed**	STM32MP1, Computer, MATLAB Simulink, Embedded Coder, Build environment for cross-compiling, Oscilloscope, External mode, Self-balancing robot
Req_05	Motor control	It is verified whether the implementation of the motor control can be done by Simulink blocks, using the codegeneration for the Cortex-M4 core. It is verified whether the setting of the duty cycle of the PWM is done within 100 μs. It is verified whether the direction/stop logic is working. It is verified whether the HAL has not been used. Result: The implementation of the motor control can be done by Simulink blocks. Setting the duty cycle of the PWM is done within 100 μs. The direction/stop logic is working properly. The HAL has not been used. **Tests passed**	STM32MP1, Computer, MATLAB Simulink, Embedded Coder, Build environment for cross-compiling, External mode Self-balancing robot

No./ID	Non-technical title	Verification of the requirement	Tools
Req_06	Hall encoder	It is verified whether the implementation of the hall encoders can be done by Simulink blocks, using the codegeneration for the Cortex-M4 core. It is verified whether a measurement of the hall encoder value is performed within 50 μs. It is verified whether the direction of rotation of the motor to which the hall sensor is attached can be recorded. It is verified whether the HAL has not been used. Result: The implementation of the hall encoders can be done by Simulink blocks. The time measurement and the detection of the direction is performed within 50 μs. The direction of rotation of the motors to which the Hall sensors are attached is detected. The HAL has not been used. **Tests passed.**	STM32MP1, Computer, MATLAB Simulink, Embedded Coder, Build environment for cross-compiling, External mode Self-balancing robot
Req_07	Ultrasonic sensor	It is verified whether the ultrasonic sensor can be implemented via Simulink blocks, using the codegeneration for the Cortex-M4 core. It is verified whether it is possible to adjust when the measurement of the ultrasonic sensor takes place. It is verified that the acquisition of the signal returned by the ultrasonic sensor is performed within 100 μs. It is verified whether the HAL has not been used. Results: The ultrasonic sensor can be implemented using the developed simulink blocks. The point in time when the ultrasonic sensor measurement should take place can be adjusted The acquisition of the returned value of the ultrasonic sensor is performed within 100 μs. The HAL has not been used. **Tests passed**	STM32MP1, Computer, MATLAB Simulink, Embedded Coder, Build environment for cross-compiling, Oscilloscope, External mode, Self-balancing robot

No./ID	Non-technical title	Verification of the requirement	Tools
Req_08	Battery voltage	It is verified whether the implementation of the battery voltage measurement can be done by Simulink blocks, using the codegeneration for the Cortex-M4 core. It is verified whether it is possible to adjust when the measurement of the battery voltage takes place. It is verified whether the Battery voltage value consists of the mean of 100 values. It is verified whether the acquisition of the battery voltage is performed within 2 ms. Result: The implementation of the battery voltage measurement can be done by Simulink blocks. It is possible to adjust when the ADC measurement takes place. The battery voltage is measured using the average value of 100 values. The battery voltage is measured within 2 ms **Tests passed**	STM32MP1, Computer, MATLAB Simulink, Embedded Coder, Build environment for cross-compiling, External mode Self-balancing robot
Req_09	Tuning model parameters via Touch display	It is verified whethe parameters can be adjusted by using graphical sliders displayed by the touch display. It is verified whether model parameters can be plotted in a diagram shown by the touch display. Result: Model parameters can be adjusted by using graphical sliders displayed on the touch display. Model parameters are plotted via a diagram on the touch display. **Test passed.**	STM32MP1, Computer, arm-none-eabi-gcc cross-compiler, Debug-tool, CMake
Req_10	Real-time Control	It must be checked whether the real-time firmware complies with the real-time requirements. Result: The real-time conditions are fulfilled. **Tests passed**	STM32MP1, Computer, MATLAB Simulink, Embedded Coder, Build environment for cross-compiling, External mode Self-balancing robot

Table 6.1: Verification plan

Main measurements

Model sample time

According to Req_01, a model must be capable of operating with a sample time of 100 μs. The model shown in figure 6.1 is used to verify this. Inside the model, a function generator generates a rectangular function. One sample of this function has the value 1 and the next sample has the value 0. Connecting the output of this function generator to the GPIO-Write block allows measuring the sample time of the model using an oscilloscope. The measured periods correspond to two model steps. Table 6.2 shows which model step times have been tested and whether the test has been passed.

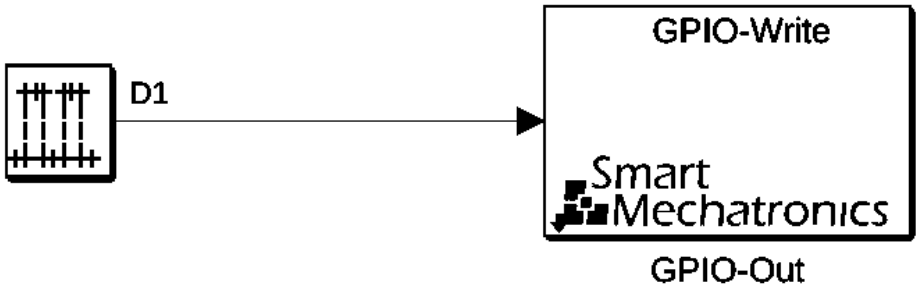


Figure 6.1: Step Time test

Step Time	Test result
1 s	passed
100 ms	passed
10 ms	passed
1 ms	passed
100 μs	passed
10 μs	failed

Table 6.2: Step time test result

It is seen, that the required sample time of 100 μs can be kept, seen in figure 6.2. A sample time of 10 μs fails. The reason for this has not been investigated, it may be due

to slow implementation of the GPIO-Write block, or the time needed to calculate the model step.

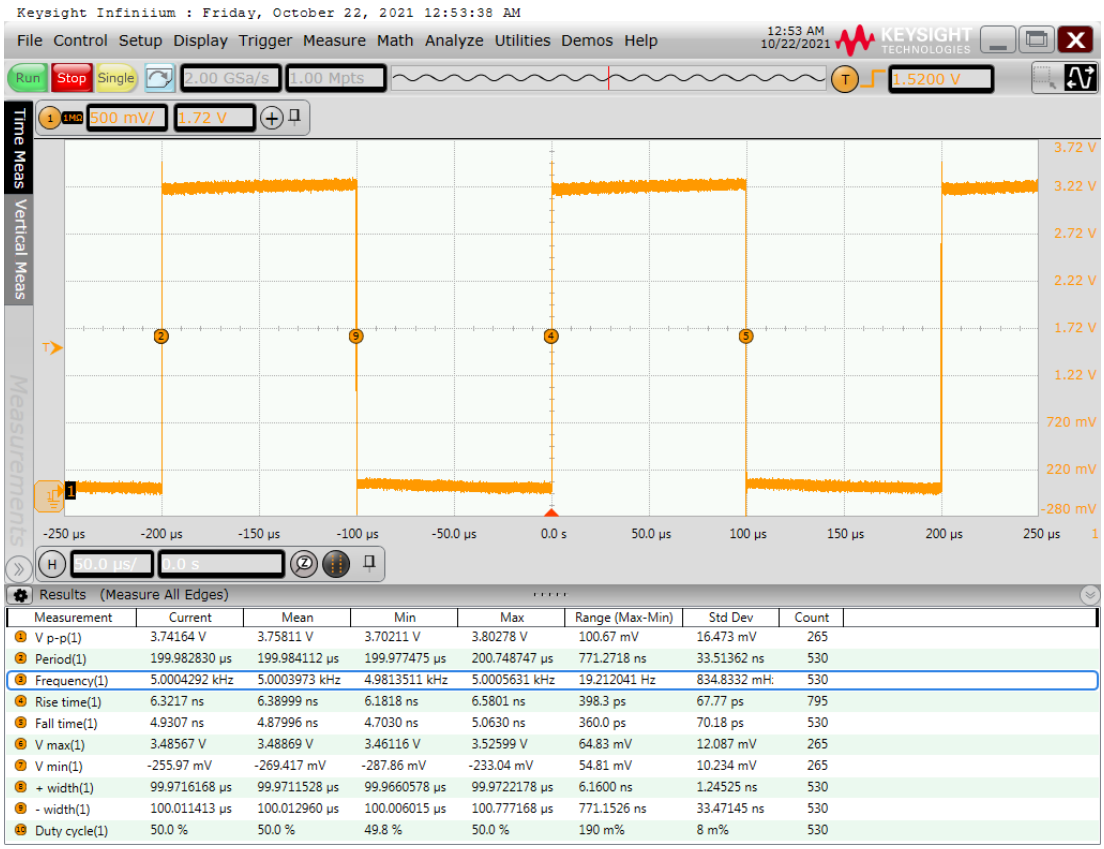
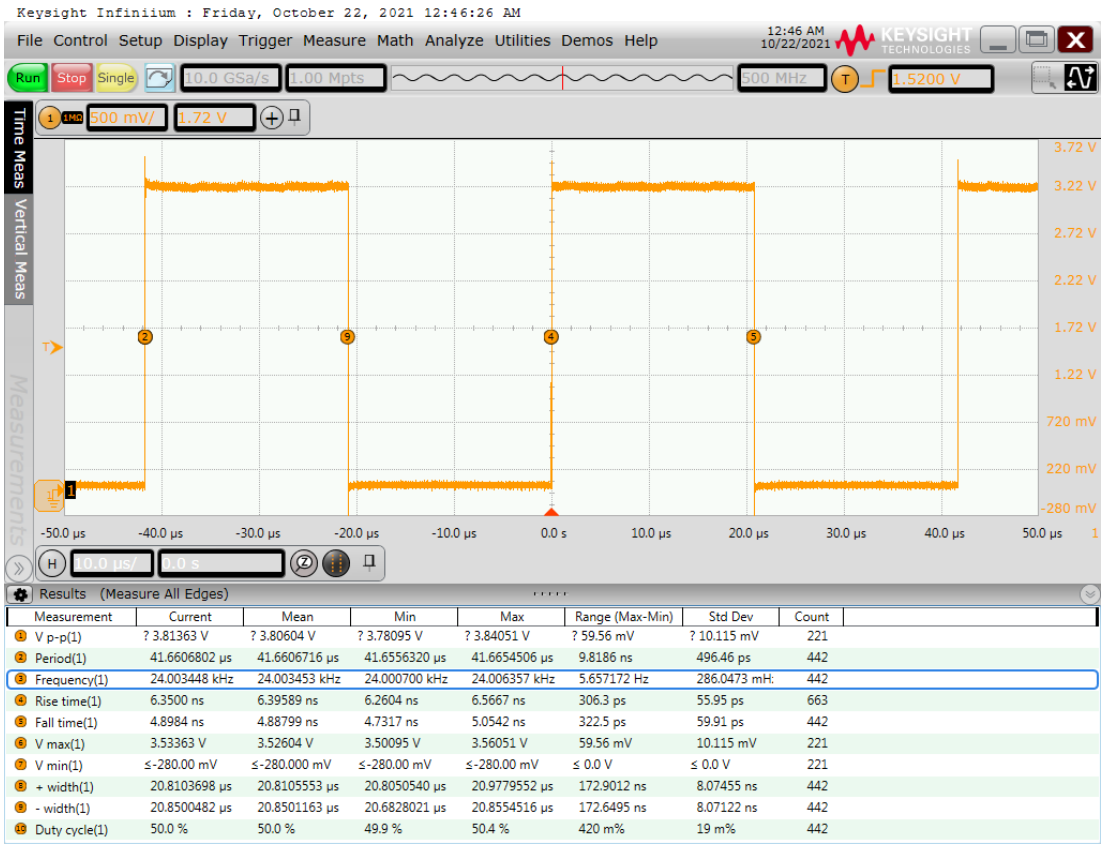


Figure 6.2: Model step time = 10 μs

As seen in figure 6.3, a period of 41.66 μs can also be measured. The resulting model sample time is 20.83 μs. That is equivalent to a frequency of 48 kHz, which could make the implementation interesting for audio processing.



Measurement	Current	Mean	Min	Max	Range (Max-Min)	Std Dev	Count
V p-p(1)	? 3.81363 V	? 3.80604 V	? 3.78095 V	? 3.84051 V	? 59.56 mV	? 10.115 mV	221
Period(1)	41.6606802 µs	41.6606716 µs	41.6556320 µs	41.6654506 µs	9.8186 ns	496.46 ps	442
Frequency(1)	24.003448 kHz	24.003453 kHz	24.000700 kHz	24.006357 kHz	5.657172 Hz	286.0473 mH:	442
Rise time(1)	6.3500 ns	6.39589 ns	6.2604 ns	6.5667 ns	306.3 ps	55.95 ps	663
Fall time(1)	4.8984 ns	4.88799 ns	4.7317 ns	5.0542 ns	322.5 ps	59.91 ps	442
V max(1)	3.53363 V	3.52604 V	3.50095 V	3.56051 V	59.56 mV	10.115 mV	221
V min(1)	≤-280.00 mV	≤-280.000 mV	≤-280.00 mV	≤-280.00 mV	≤ 0.0 V	≤ 0.0 V	221
+ width(1)	20.8103698 µs	20.8105553 µs	20.8050540 µs	20.9779552 µs	172.9012 ns	8.07455 ns	442
- width(1)	20.8500482 µs	20.8501163 µs	20.6828021 µs	20.8554516 µs	172.6495 ns	8.07122 ns	442
Duty cycle(1)	50.0 %	50.0 %	49.9 %	50.4 %	420 m%	19 m%	442

Figure 6.3: Model step time = 20.83 μs

External mode data transfer

Figure 6.4 shows a model, that is used to test the transmission of the external mode via XCP on TCP/IP. For this purpose, a known random `uint16_t` array consisting of 4096 elements is loaded into the look-up table. This test is performed with a sample time of 1 ms, resulting in a data transfer of $2\,\mathrm{kB\,s^{-1}}$, which is transferred from the Cortex-M4 via the Cortex-A7 to the development computer. After the runtime of the external mode, the values received from the target can be compared with the values of the known array. This can be seen in figure 6.5.

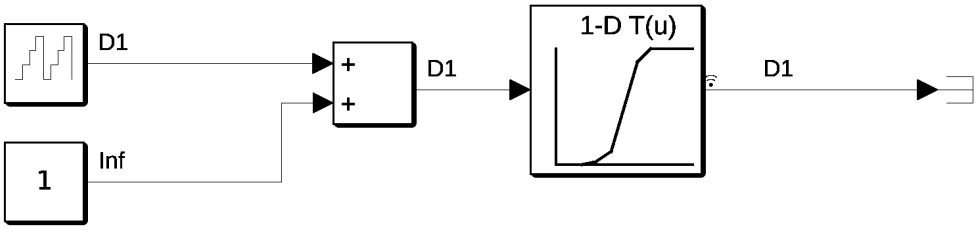


Figure 6.4: External mode via XCP communication test

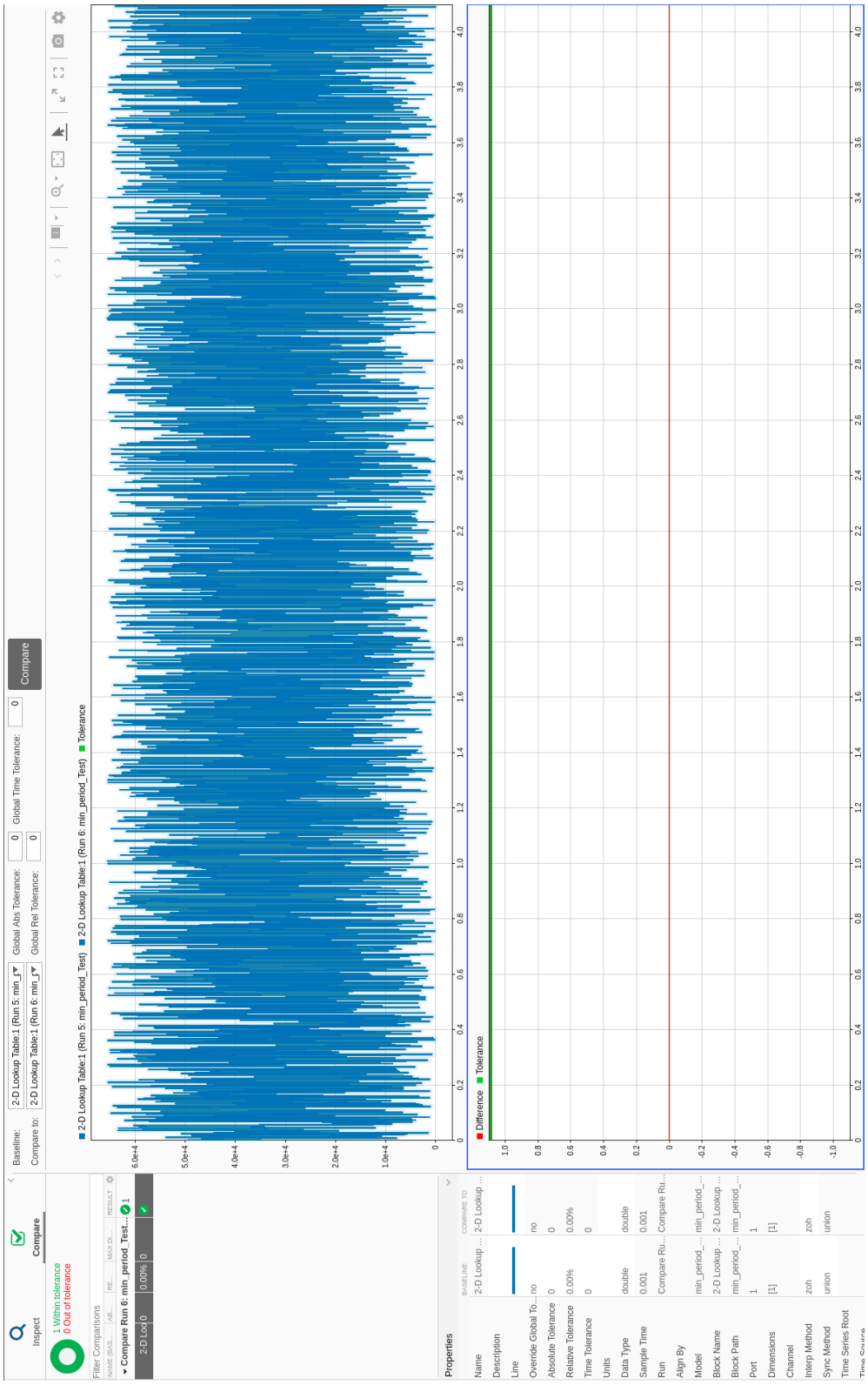


Figure 6.5: Comparison between the known array values and the values sent by the target via the external mode via XCP on TCP/IP.

Real-time control

During the verification of the real-time system on the Cortex-M4, the Cortex-A7 is intentionally stressed, to show that the Cortex-M4 and the Cortex-A7 work independently. For this purpose, several applications are started simultaneously on the Linux operating system of the Cortex-A7. An snapshot of the utilization of the Cortex-A7 during the real-time verification of the Cortex-M4 is seen in figure 6.6.

```
top - 13:53:48 up 20 min,  4 users,  load average: 8.61, 4.02, 2.28
Tasks: 145 total,   7 running, 138 sleeping,   0 stopped,   0 zombie
%Cpu(s): 44.4 us, 54.0 sy,  0.0 ni,  0.1 id,  0.0 wa,  0.0 hi,  1.4 si,  0.0 st
MiB Mem :    428.0 total,    154.5 free,    136.6 used,    136.9 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.    257.0 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 2313 root      20   0   69436  21484  15344 R 104.4   4.9   0:54.91 lvgl_ext_mode
 1991 root      20   0    2008   1396   1200 R  25.4   0.3   0:13.38 dropbear
 1957 root      20   0   47304  20140  16276 S  23.3   4.6   2:24.15 lvgl
  389 root      20   0  127312  21456  15792 S  17.2   4.9   3:35.44 weston
 1381 root      20   0  103448  14988   7576 R  12.2   3.4   1:57.03 weston-st-egl-c
```

Figure 6.6: Deliberate stressing of the Cortex-A7 during real-time verification of the Cortex-M4 firmware

To verify the real-time firmware on the Cortex-M4, the idle time calculated in section 4.2 must be kept. To observe this, a GPIO pin is set at each task start and reset at each task end. The maximum average voltage of the GPIO pin is used to derive the idle time. This measurement is seen in figure 6.7.

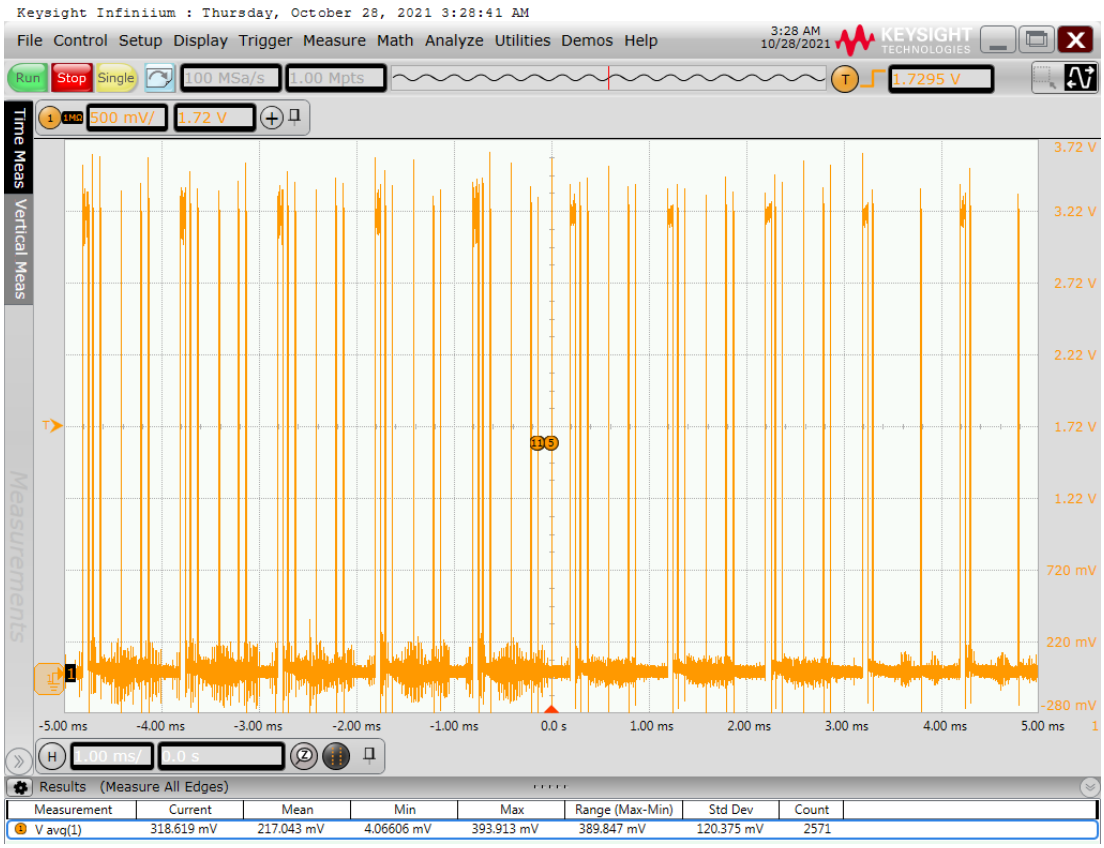


Figure 6.7: Measuring the idle time of the Cortex-M4 (preemption enabled)

The relative idle time T_{idle} of the processor is calculated as seen in equation (6.1).

$$T_{idle} = \frac{V_{REF} - V_{\hat{avg}}}{V_{REF}} \tag{6.1}$$

The result is $T_{idle} \approx 88\,\%$. The relative core utilization is $U_c \approx 12\,\%$.
This measurement has a systematic error. This always occurs when a task with a low priority is interrupted by a task with a higher priority. When the processor jumps back to the task of the lower priority, the GPIO pin is not set again, so the rest of the task processing is not covered by the measurement. To prevent this, the preemption of each task is deactivated for the measurement. This prevents the processor from switching to a task with a higher priority while a task is being processed. The measurement where the preemtion is disabeld is seen in figure 6.8.

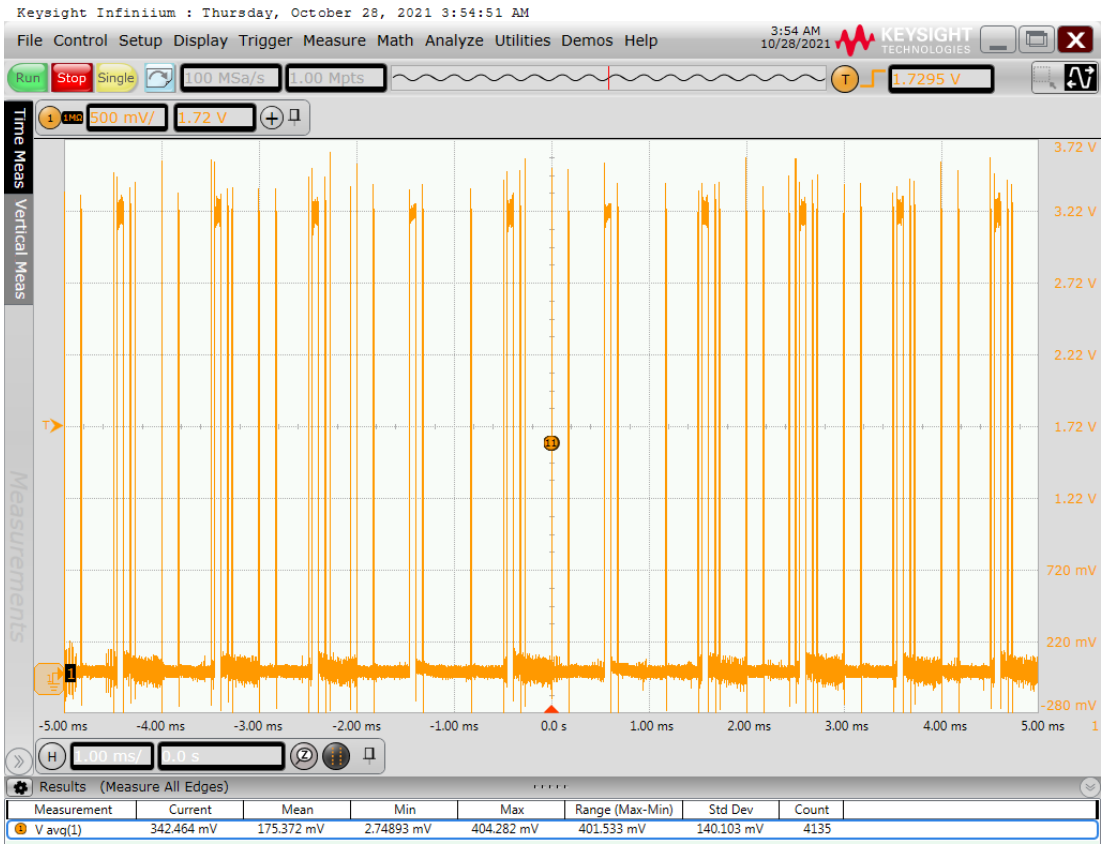


Figure 6.8: Measuring the idle time of the Cortex-M4 (preemption disabled)

This measurement results $T_{idle} \approx 87\,\%$ and $U_c \approx 13\,\%$. If the relative core utilization U_c and the maximum utilization U_{sum}, calculated in equation (4.1), are compared, it is seen that:

$U_c \approx 0.12 < 0.712 \approx U_{sum}$.

This means that the real-time condition is met.

7 Conclusion

The MATLAB Simulink coder target designed and implemented within this project creates the possibility to perform model-based design on the Cortex-M4 of the STM32MP1. With the implemented external mode via XCP on TCP/IP, parameters of the real-time firmware running on the Cortex-M4 can be observed and tuned. While the Cortex-M4 is executing the real-time firmware, a graphical application can be executed on the Cortex-A7, which can display and tune the parameters of the real-time process.

With the Simulink coder target and the Simulink blocks, both developed in this project, the example application control of the self-balancing robot can be implemented model-based.

During the development of the Simulink blocks, the focus was on a hardware-related and fast implementation. By using and modifying the ARM-Cortex-M interrupt block, the hardware interrupts of the NVIC can be integrated into the Simulink model. By using interrupts and DMAs within the real-time application, polling is avoided.

Thanks to existing drivers on the Linux OS running on the Cortex-A7, network and graphic touch applications can be developed rapidly. Even the utilization of the Linux processor does not affect the real-time application. The communication of the heterogeneous processors via the shared memory can be implemented as a communication interface for the external mode via XCP.

The setup CMake project resolves the dependencies of the build process from the template Makefile of MATLAB Simulink or the Makefile projet of the STM32CubeIDE.

The controller for the self-balancing robot, created by the model-based design, can keep the self-balancing robot in its unstable resting position. The acceleration components resulting from changes in direction, impacts, rapid acceleration, and deceleration can be estimated by applying the Kalman filter.

7.1 Outlook

In future projects, the communication of the heterogeneous processors can be implemented via the indirect buffer exchange mode. The data acquisition of the Hall encoders and the ultrasonic sensor could be further optimized by a DMA-based implementation. Furthermore, it would be possible to optimize the control of the self-balancing robot by implementing a linear-quadratic controller. Through model-based development, a control for the trajectory of the self-balancing robot can now be planned and implemented. A graphical Linux application could also be developed to match them.

List of Figures

List of Tables

Listings

Bibliography

[1] Geoffrey Blake, Ronald Dreslinski, and Trevor Mudge. “A survey of multicore processors.” In: *IEEE Signal Processing Magazine* 26.6 (2009), pp. 26–37. ISSN: 1053-5888. DOI: 10.1109/MSP.2009.934110.

[2] Hermann Kopetz. *Real-time systems: Design principles for distributed embedded applications*. 2nd edition. Real-time systems series. New York, Dordrecht, and Heidelberg: Springer, 2011. ISBN: 978-1-4419-8237-7.

[3] Marko Bertogna. “A View on Future Challenges for the Real-Time Community.” In: *RTNS 2019 Keynote*. 2019. URL: https://www.irit.fr/rtns2019/wp-content/uploads/2019/11/bertogna_keynote.pdf (visited on 10/28/2021).

[4] Youssef Zaki. “An Embedded Multi-Core Platform for Mixed-Criticality Systems.” Master of Science Thesis. School of Information and Communication Technology, Stockholm, 2016.

[5] Philip Geuchen. “Machbarkeitsstudie zu der Mikroprozessor-Plattform STM32MP1.” Bachelorarbeit. Hochschule Bochum - Bochum University of Applied Sciences, 2020.

[6] Philip Geuchen. “Adaptation of a MATLAB Simulink target to the Cortex-M4 processor of the microprocessor unit STM32MP157C-DK2.” Development project. Hochschule Bochum - Bochum University of Applied Sciences, 2021.

[7] Andreas Prösser. “Entwicklung und Evaluierung eines STM32F4Discovery Targets für MATLAB/Simulink.” Masterstudienarbeit. FH Dortmund, 2017.

[8] The MathWorks, Inc. *Simulation und Model-Based Design*. URL: https://de.mathworks.com/products/simulink.html (visited on 09/17/2021).

[9] STMicroelectronics. *Company presentation*. https://www.st.com/content/ccc/resource/corporate/company/company_presentation/8d/fc/ba/0b/41/0d/47/12/company_presentation.pdf/files/company_presentation.pdf/_jcr_content/translations/en.company_presentation.pdf, 2021. (Visited on 10/20/2021).

[10] STMicroelectronics. *STM32MP1 Series*. https://www.st.com/en/microcontrollers-microprocessors/stm32mp1-series.html, 2021. (Visited on 10/16/2021).

[11] STMicroelectronics. *STM32MP1 SWARCH: Embedded Software architecture Revision 1.0*. https://www.st.com/content/ccc/resource/training/technical/product_training/group0/79/99/30/94/a3/ec/4c/f4/STM32MP1-Software-Software_architecture/files/STM32MP1-Software-Software_architecture.pdf/jcr:content/translations/en.STM32MP1-Software-Software_architecture.pdf, 2019. (Visited on 10/18/2021).

[12] STMicroelectronics. *FreeRTOS_ThreadCreation FreeRTOS Thread Creation example*. https://github.com/STMicroelectronics/STM32CubeMP1/tree/master/Projects/STM32MP157C-DK2/Applications/FreeRTOS/FreeRTOS_ThreadCreation, 2019. (Visited on 10/20/2021).

[13] Weston Embedded Solutions. *Browse Example Projects for the µC/ Product Line: STM32H743ZI-Nucleo-144*. 2018. (Visited on 10/15/2021).

[14] STMicroelectronics. *STM32MP15 peripherals overview*. https://wiki.st.com/stm32mpu/wiki/STM32MP15_peripherals_overview, 2021. (Visited on 10/14/2021).

[15] A. Sangiovanni-Vincentelli and G. Martin. "Platform-based design and software design methodology for embedded systems." In: *IEEE Design & Test of Computers* 18.6 (2001), pp. 23–33. ISSN: 07407475. DOI: 10.1109/54.970421.

[16] Peter Marwedel. *Embedded System Design*. Cham: Springer International Publishing, 2021. ISBN: 978-3-030-60909-2. DOI: 10.1007/978-3-030-60910-8.

[17] L. Thiele. "Design Space Exploration." In: (2006). URL: http://www.artist-embedded.org/docs/Events/2006/ChinaSchool/4_DesignSpaceExploration.pdf (visited on 09/21/2021).

[18] J. Xu and D. L. Parnas. "On satisfying timing constraints in hard-real-time systems." In: *IEEE Transactions on Software Engineering* 19.1 (1993), pp. 70–84. ISSN: 00985589. DOI: 10.1109/32.210308.

[19] J. Jackson. "Scheduling a production line to minimize maximum tardiness. Management Science Research Project 43." PhD thesis. University of California, Los Angeles, 1955.

[20] C. L. Liu and J. W. Layland. "Scheduling algorithms for multi-programming in a hard real-time environment." In: (1973).

[21] Rüdiger R. Asche. *Embedded Controller*. Wiesbaden: Springer Fachmedien Wiesbaden, 2016. ISBN: 978-3-658-14849-2. DOI: 10.1007/978-3-658-14850-8.

[22] National Instruments. "DMA Fundamentals on Various PC Platforms - Application Note 011." In: (1991).

[23] Dhananjay V. Gadre and Sarthak Gupta. *Getting Started with Tiva ARM Cortex M4 Microcontrollers*. New Delhi: Springer India, 2018. ISBN: 978-81-322-3764-8. DOI: 10.1007/978-81-322-3766-2.

[24] STMicroelectronics. "STM32MP157 advanced Arm®-based 32-bit MPUs - Reference manual 0436." In: (2021).

[25] HEENE MARK R, HILL SUSAN C, and JELEMENSKY JOSEPH. "Queued serial peripheral interface for use in a data processing system." US4816996A. 1987.

[26] Dwaraka N Oruganti and Siva S Yellampalli. "Design of power efficient SPI interface." In: *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2014, pp. 2602–2606. DOI: 10.1109/ICACCI.2014.6968350.

[27] Rainer Zaiser Andreas Patzer. *XCP – Das Standardprotokoll für die Steuergeräte-Entwicklung*. Vector Informatik GmbH, 2016.

[28] Association for Standardisation of Automation and Measuring Systems. *About ASAM*. 2017. URL: https://www.asam.net/standards/detail/mcd-1-xcp/ (visited on 10/21/2021).

[29] Association for Standardisation of Automation and Measuring Systems. *About ASAM*. 2021. URL: https://www.asam.net/about-asam/our-vision/ (visited on 09/06/2021).

[30] Kitware Inc. *Overview | CMake: About CMake*. 2021. URL: https://cmake.org/overview/ (visited on 09/15/2021).

[31] The MathWorks, Inc. "Simulink Coder User's Guide." In: (2021).

[32] The MathWorks, Inc. *Target Language Compiler Basics - MATLAB & Simulink*. 2021. URL: https://de.mathworks.com/help/rtw/tlc/what-is-the-target-language-compiler.html (visited on 09/20/2021).

[33] The MathWorks, Inc. *model.rtw File and Scopes - MATLAB & Simulink*. 2021. URL: https://de.mathworks.com/help/rtw/tlc/introduction-to-the-model-rtw-file.html (visited on 09/20/2021).

[34] Reiner Marchthaler and Sebastian Dingler. *Kalman-Filter*. Wiesbaden: Springer Fachmedien Wiesbaden, 2017. ISBN: 978-3-658-16727-1. DOI: 10.1007/978-3-658-16728-8.

[35] Holger Lutz and Wolfgang Wendt. *Taschenbuch der Regelungstechnik: Mit MATLAB und Simulink*. 8., erg. Aufl. Frankfurt am Main: Deutsch, 2010. ISBN: 978-3-8171-1859-5. URL: http://deposit.d-nb.de/cgi-bin/dokserv?id=3430963&prov=M&dok_var=1&dok_ext=htm.

[36] Jan Lunze. *Regelungstechnik 2: Mehrgrößensysteme, Digitale Regelung*. 9., überarbeitete und aktualisierte Auflage. Lehrbuch. Berlin and Heidelberg: Springer Vieweg, 2016. ISBN: 978-3-662-52676-7.

[37] Heinz Unbehauen. *Regelungstechnik II: Zustandsregelungen, digitale und nichtlineare Regelsysteme.* 9., durchges. und korr. Aufl., korr. Nachdr. Studium. Wiesbaden: Vieweg + Teubner, 2009. ISBN: 9783528833480.

[38] Peter Iwanek et al. “Fachdisziplinübergreifende Systemmodellierung mechatronischer Systeme mit SysML und CONSENS.” In: *Tag des Systems Engineerings*. Ed. by Maik Maurer and Sven-Olaf Schulze. 2013th ed. Tag des Systems Engineering. ISBN (Buch): 978-3-446-43915-3 Titel anhand dieser ISBN in Citavi-Projekt übernehmen. Carl Hanser Verlag, Nov. 2013. Chap. Modellbasierte Systementwicklung 2, pp. 337–346.

[39] Terasic Inc. *Balance Car Daughterboard: Schematic design: Rev D0*.

[40] Andreas Prösse. “Entwicklung und Evaluierung eines STM32F4Discovery Targets für MATLAB/Simulink.” Masterstudienarbeit. FH Dortmund, 2017, 2017.

[41] Amazon Web Services Inc. *FreeRTOS: Real-time operating system for microcontrollers*. https://www.freertos.org/, 2021. (Visited on 09/23/2021).

[42] The MathWorks, Inc. *Customize XCP Slave Software*. 2021. URL: https://de.mathworks.com/help/rtw/ug/customize-xcp-slave-software.html (visited on 10/15/2021).

[43] STMicroelectronics. *STM32Cube initialization code generator*. https://www.st.com/en/development-tools/stm32cubemx.html. (Visited on 10/15/2021).

[44] STMicroelectronics. *STM32CubeMP1 MPU Firmware Package, Release STM32CubeMP1 v1.4.0 Ecosystem*. https://github.com/STMicroelectronics/STM32CubeMP1/tree/1.4.0. (Visited on 10/19/2021).

[45] The MathWorks, Inc. *Embedded Coder*. 2021. URL: https://de.mathworks.com/products/embedded-coder.html (visited on 09/18/2021).

[46] The MathWorks, Inc. *Embedded Coder Support Package for STMicroelectronics Discovery Boards*. 2021. URL: https://de.mathworks.com/matlabcentral/fileexchange/43093-embedded-coder-support-package-for-stmicroelectronics-discovery-boards (visited on 10/14/2021).

[47] The MathWorks, Inc. *NXP Support Package S32K1xx*. 2021. URL: https://de.mathworks.com/matlabcentral/fileexchange/64740-nxp-support-package-s32k1xx (visited on 10/07/2021).

[48] The MathWorks, Inc. *Embedded Coder Support Package for Texas Instruments C2000 Processors*. 2021. URL: https://de.mathworks.com/matlabcentral/fileexchange/43096-embedded-coder-support-package-for-texas-instruments-c2000-processors (visited on 10/14/2021).

[49] The MathWorks, Inc. *MATLAB Support Package for Raspberry Pi Hardware*. 2021. URL: https://de.mathworks.com/help/supportpkg/raspberrypiio/ (visited on 10/11/2021).

[50] The MathWorks, Inc. *example_file_process.tlc*. matlabroot/toolbox/rtw/targets/ecoder/example_file_process.tlc, 1994. (Visited on 10/17/2021).

[51] The MathWorks, Inc. *File location of the real-time codertarget within the MATLAB root directory*. matlabroot/toolbox/target/codertarget/rtw/, 2021. (Visited on 10/10/2021).

[52] The MathWorks, Inc. *codertarget_mainwithoutOS.tlc*. matlabroot/toolbox/target/codertarget/rtw/codertarget_mainwithoutOS.tlc, 2013. (Visited on 10/16/2021).

[53] The MathWorks, Inc. *Target Language Compiler Directives*. 2021. URL: https://de.mathworks.com/help/rtw/tlc/target-language-compiler-directives.html (visited on 10/07/2021).

[54] The MathWorks, Inc. *Code Configuration Functions*. 2021. URL: https://de.mathworks.com/help/rtw/tlc/code-configuration-functions.html (visited on 10/08/2021).

[55] STMicroelectronics: MCD Application Team. *stm32mp1xx_ll_tim.c*. https://github.com/STMicroelectronics/STM32CubeMP1/blob/master/Drivers/STM32MP1xx_HAL_Driver/Src/stm32mp1xx_ll_tim.c, 2019. (Visited on 09/04/2021).

[56] STMicroelectronics: MCD Application Team. *stm32mp1xx_ll_tim.h*. https://github.com/STMicroelectronics/STM32CubeMP1/blob/master/Drivers/STM32MP1xx_HAL_Driver/Inc/stm32mp1xx_ll_tim.h, 2019. (Visited on 10/10/2021).

[57] STMicroelectronics: MCD Application Team. *core_cm4.h*. https://github.com/STMicroelectronics/STM32CubeMP1/blob/master/Drivers/CMSIS/Core/Include/core_cm4.h, 2019. (Visited on 09/03/2021).

[58] The MathWorks, Inc. *Simulink External Mode Interface*. 2021. URL: https://de.mathworks.com/help/slrealtime/gs/simulink-external-mode-interface.html (visited on 10/13/2021).

[59] The MathWorks, Inc. *External Mode Simulations for Parameter Tuning and Signal Monitoring*. 2021. URL: https://de.mathworks.com/help/rtw/ug/external-mode-simulations-for-parameter-tuning-and-signal-monitoring.html (visited on 10/15/2021).

[60] The MathWorks, Inc. *Create a Transport Layer for TCP/IP or Serial External Mode Communication*. 2021. URL: https://de.mathworks.com/help/rtw/ug/creating-a-tcp-ip-transport-layer-for-external-communication.html (visited on 10/14/2021).

[61] The MathWorks, Inc. *Host-Target Communication for Simulink PIL Simulation*. 2021. URL: https://de.mathworks.com/help/ecoder/ug/target-connectivity-pil-api-components.html (visited on 10/15/2021).

[62] The MathWorks, Inc. *External Mode Simulation with TCP/IP or Serial Communication*. 2021. URL: https://de.mathworks.com/help/rtw/ug/external-mode-simulation-with-tcpip-or-serial-communication.html (visited on 10/15/2021).

[63] The MathWorks, Inc. *XCP Platform Abstraction Layer interface*. matlabroot/toolbox/coder/xcp/src/target/slave/platform/include/xcp_platform.h, 2016. (Visited on 10/25/2021).

[64] The MathWorks Inc. *Host-Target Communication for Simulink PIL simulation*. https://www.mathworks.com/help/ecoder/ug/target-connectivity-pil-api-components.html. (Visited on 02/02/2021).

[65] STMicroelectronics. *Coprocessor management overview Picture*. https://wiki.st.com/stm32mpu/nsfr_img_auth.php/3/3e/Copro-sw-ipc-overview.png. (Visited on 12/25/2019).

[66] STMicroelectronics. *Coprocessor management overview*. https://wiki.st.com/stm32mpu/wiki/Coprocessor_management_overview. (Visited on 12/25/2019).

[67] STMicroelectronics. *Exchanging buffers with the coprocessor*. https://wiki.st.com/stm32mpu/wiki/Exchanging_buffers_with_the_coprocessor, 2021. (Visited on 10/20/2021).

[68] STMicroelectronics. *OpenAMP_TTY_echo OpenAMP TTY echo example*. https://github.com/STMicroelectronics/STM32CubeMP1/tree/master/Projects/STM32MP157C-DK2/Applications/OpenAMP/OpenAMP_TTY_echo, 2019. (Visited on 10/13/2021).

[69] STMicroelectronics: MCD Application Team. *virt_uart.c - VIRTUAL UART HAL module driver*. https://github.com/STMicroelectronics/STM32CubeMP1/blob/master/Middlewares/Third_Party/OpenAMP/virtual_driver/virt_uart.c, 2019. (Visited on 08/01/2021).

[70] STMicroelectronics. *Linux remoteproc framework overview*. https://wiki.st.com/stm32mpu/wiki/Linux_remoteproc_framework_overview, 2021. (Visited on 10/20/2021).

[71] STMicroelectronics. *Linux Mailbox framework overview*. https://wiki.st.com/stm32mpu/wiki/Linux_Mailbox_framework_overview, 2021. (Visited on 10/20/2021).

[72] Michael Kerrisk. *open(2) — Linux manual page*. https://man7.org/linux/man-pages/man2/open.2.html, 2021. (Visited on 10/21/2021).

[73] Michael Kerrisk. *ioctl(2) — Linux manual page*. https://man7.org/linux/man-pages/man2/ioctl.2.html, 2021. (Visited on 10/21/2021).

[74] Vincent Abriou for STMicroelectronics. *copro.c - Implements Copro's abstraction layer*. https://github.com/STMicroelectronics/meta-st-openstlinux/blob/4e36cf0c7c7bdb24700a990f73afcda84cdcf0cf/recipes-samples/ai-nn-application/ai-hand-char-reco-launcher/copro.c, 2019. (Visited on 10/07/2021).

[75] The MathWorks, Inc. *rtiostream_tcpip.c*. matlabroot/toolbox/coder/rtiostream/src/rtiostreamtcpip/rtiostream_tcpip.c, 1994. (Visited on 10/17/2021).

[76] Michael Kerrisk. *pthreads(7) — Linux manual page*. https://man7.org/linux/man-pages/man7/pthreads.7.html, 2021. (Visited on 10/21/2021).

[77] The MathWorks, Inc. *MEX-file arguments*. https://de.mathworks.com/help/rtw/ref/mex-file-arguments.html, 2021. (Visited on 09/26/2021).

[78] STMicroelectronics. *Integrated Development Environment for STM32*. https://www.st.com/en/development-tools/stm32cubeide.html, 2021. (Visited on 10/15/2021).

[79] STMicroelectronics. *STM32CubeIDE user guide: User manual 2609*. 2021.

[80] Kitware Inc. *Really Cool CMake Features*. 2021. URL: https://gitlab.kitware.com/cmake/community/-/wikis/doc/cmake/Really-Cool-CMake-Features (visited on 09/15/2021).

[81] STMicroelectronics. "STM32 Cortex®-M4 MCUs and MPUs programming manual - Programming manual 0214." In: (2020).

[82] The MathWorks, Inc. *Create Hardware Interrupt Block for an ARM Cortex-M Based Processor using an Interrupt Description File - MATLAB & Simulink - MathWorks Deutschland*. 15.09.2021. URL: https://de.mathworks.com/help/supportpkg/stmicroelectronicsstm32f4discovery/ug/create-

hardware-interrupt-block-for-an-arm-cortex-m-based-processor-using-an-interrupt-description-file.html (visited on 09/15/2021).

[83] The MathWorks, Inc. *Embedded Coder Support Package for ARM Cortex-M Processors*. 15.09.2021. URL: https://de.mathworks.com/matlabcentral/fileexchange/43095-embedded-coder-support-package-for-arm-cortex-m-processors (visited on 09/16/2021).

[84] Arm Ltd. "__weak." In: (2021). URL: https://www.keil.com/support/man/docs/armcc/armcc%5C_chr1359124970859.htm (visited on 09/17/2021).

[85] The MathWorks, Inc. *Types of Custom Blocks*. 2021. URL: https://de.mathworks.com/help/simulink/ug/create-your-own-simulink-block.html (visited on 10/13/2021).

[86] The MathWorks, Inc. *Comparison of Custom Block Functionality*. 2021. URL: https://de.mathworks.com/help/simulink/ug/comparison-of-custom-block-functionality.html (visited on 10/15/2021).

[87] The MathWorks, Inc. *Maintain Level-1 MATLAB S-Functions*. 2021. URL: https://de.mathworks.com/help/simulink/sfg/maintaining-level-1-matlab-s-functions.html (visited on 10/13/2021).

[88] The MathWorks, Inc. *Block Target File Methods*. 2021. URL: https://de.mathworks.com/help/rtw/tlc/block-target-file-methods.html (visited on 10/13/2021).

[89] STMicroelectronics. "General-purpose timer cookbook for STM32 microcontrollers - Application note 4776." In: (2019).

[90] STMicroelectronics. *ADC_SingleConversion_TriggerTimer_DMA ADC example*. https://github.com/STMicroelectronics/STM32CubeMP1/tree/master/Projects/STM32MP157C-DK2/Examples/ADC/ADC_SingleConversion_TriggerTimer_DMA, 2019. (Visited on 10/17/2021).

[91] STMicroelectronics. *STM32CubeMX introduction*. 2021. URL: https://wiki.st.com/stm32mcu/wiki/STM32CubeMX_introduction (visited on 10/08/2021).

[92] STMicroelectronics. *M10_Discovery_STM32MP157C-DK2_STM32MP157CAC_Board_AllConfig.ioc*. STM32CubeMXroot/db/plugins/boardmanager/boards/M10_Discovery_STM32MP157C-DK2_STM32MP157CAC_Board_AllConfig.ioc, 2019. (Visited on 10/14/2021).

[93] InvenSense Inc. "MPU-6500-Datasheet." In: (2014).

[94] Horst Czichos. *Mechatronik*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015. ISBN: 978-3-658-09949-7. DOI: 10.1007/978-3-658-09950-3.

[95] TOSHIBA Corporation. "TB6612FNG: Driver IC for Dual DC motor." In: (2014).

[96] Terasic Inc. *Self-Balancing Robot: Hardware Manual*.

[97] ELECFREAKS. "Ultrasonic Ranging Module HC - SR04." In: (2018).

[98] STMicroelectronics. *UM2534: User manual: Discovery kits with STM32MP157 MPUs*. 2019. URL: https://www.st.com/resource/en/user_manual/dm00591354-discovery-kits-with-stm32mp157-mpus-stmicroelectronics.pdf (visited on 10/12/2021).

[99] DeviceTree c/o Linaro. *Devicetree Specification Release v0.2*. https://github.com/devicetree-org/devicetree-specification/releases/download/v0.2/devicetree-specification-v0.2.pdf. (Visited on 10/02/2021).

[100] STMicroelectronics. *How to compile the device tree with the Developer Package*. https://wiki.st.com/stm32mpu/wiki/How_to_compile_the_device_tree_with_the_Developer_Package, 2021. (Visited on 10/20/2021).

[101] InvenSense Inc. "MPU-6500 Register Map and Descriptions: Revision 2.1." In: (2013).

[102] The MathWorks, Inc. *Kalman Filter*. URL: https://de.mathworks.com/help/control/ref/kalmanfilter.html (visited on 09/15/2021).

[103] Steffen Paul and Reinhold Paul. *Grundlagen der Elektrotechnik und Elektronik 1*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. ISBN: 978-3-642-53947-3. DOI: 10.1007/978-3-642-53948-0.

[104] D. Andessner and R. Seebacher. "Exzentrisch gelagerte Asynchronmaschine als inverses Pendel." In: *e & i Elektrotechnik und Informationstechnik* 121.3 (2004), pp. 95–100. ISSN: 1613-7620. DOI: 10.1007/BF03054987.

[105] Wilhelm August. "Sicherer Transport beweglicher Lasten mittels modellbasierter Echtzeitregelung für Industrieroboter." In: (2012).

[106] Jan Polzer. "Berechnung der Nulldynamik eines inversen Pendels mit unterschiedlichen mathematischen Theorien." In: (2000).

[107] Ekbert Hering, Rolf Martin, and Martin Stohrer. *Physik für Ingenieure*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016. ISBN: 978-3-662-49354-0. DOI: 10.1007/978-3-662-49355-7.

[108] Lothar Papula. *Mathematische Formelsammlung*. Wiesbaden: Springer Fachmedien Wiesbaden, 2014. ISBN: 978-3-8348-1913-0. DOI: 10.1007/978-3-8348-2311-3.

[109] The MathWorks, Inc. *System Identification Toolbox*. 2021. URL: https://de.mathworks.com/products/sysid.html (visited on 10/14/2021).

[110] The MathWorks, Inc. *Control System Toolbox: Entwurf und Analyse von Regelungssystemen*. URL: https://de.mathworks.com/products/control.html (visited on 09/15/2021).

[111] Zou Ziming, Dai Chonghao, and Yang Ling. "Design of Control System for Double Wheel Balancing Vehicle Based on STM32." In: *Proceedings of the 2020 2nd International Conference on Big Data and Artificial Intelligence*. Ed. by M. James C. Crabbe et al. New York, NY, USA: ACM, 4282020, pp. 295–299. ISBN: 9781450376457. DOI: 10.1145/3436286.3436409.

[112] LVGL LLC. *Light and Versatile Graphics Library*. https://github.com/lvgl/lvgl. (Visited on 10/05/2021).

[113] LVGL LLC. *Light and Versatile Graphics Library drivers*. `https://github.com/lvgl/lv_drivers`. (Visited on 10/05/2021).

[114] KOAN s.a.s. *Yocto Project Meta Layer Linux STM32MP1 by Koan*. `https://koansoftware.com/yocto-project-meta-layer-for-stm32mp1-by-koan/`. (Visited on 10/12/2021).

A Appendix

A.1 Hardware Registers

The following applies to the readability and writability of the register bits:

Symbols	Meaning
rw	The bit can be read and written
rc_w0	The bit is set by hardware and reset by software
rs	The bit is set by software and reset by hardware
w	The bit can only be written

Table A.1: Symbols indicating the readability and writability of register bits

TIMx Prescaler

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC															
rw	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*

Figure A.1: Prescaler Register .cf [24, p. 2127]

PSC (Prescaler): [24]

A value is programmed into the prescaler by for dividing the incoming clock frequency. The resulting clock frequency $f_{CK_{CNT}}$ is determined as follows:

$$f_{CK_{CNT}} = \frac{f_{CK_{PSC}}}{PSC + 1} \tag{A.1}$$

TIMx Auto-Reload Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR															
rw	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*

Figure A.2: Auto-Reload Register .cf [24, p. 2127]

ARR (Auto-Reload Register): [24]

The Auto-Reload value is programmed into the ARR. If the value in the ARR is zero, the counter is frozen.

The frequency of the repeating timer period through up or downcountiong is determined by:

$$f_{Period} = \frac{f_{CK_{CNT}}}{ARR + 1} \tag{A.2}$$

TIMx Capture/Compare Mode Register 1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
							OC2M								*OC1M*
							rw								*rw*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2 CE	*OC2M*			*OC2 PE*	*OC2 FE*	*CC2S*		*OC1 CE*	*OC1M*			*OC1 PE*	*OC1 FE*	*CC1S*	
rw		*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*

Figure A.3: Capture/Compare Mode Register 1 .cf [24, p. 2118]

In the register description, x is the channel number 1 or 2.

OCxCE (Output Compare x Clear Enable) [24]

0: no effects

1: clears CxREF if a High level is detected on ETRF (output of the resynchronization circuit)

OCxM (Output Compare x Mode): [24]

110: set output to PWM mode 1 (upcounting)

See [24, p. 2120] for a deeper description of the PWM modes

OCxPE (Output Compare x Preload Enable): [24]

0: disables Preload register

1: enables Preload register

OCxFE (Output Compare x Fast Enable): [24]

This bit accelerates the processing of events to the Capture/Compare output or the trigger input.

0: Acceleration off, the minimum triggerd input delay is 5 clock cycles

1: Acceleration on, the triggerd input delay is reduced to 3 clock cycles

CCxS (Capture/Compare x Selection): [24]

Bit-field configures timer channel as input or output

00: configures Capture/Compare channel x as output

01: configures Capture/Compare channel x as external input 1

10: configures Capture/Compare channel x as external input 2

11: configures Capture/Compare channel x as internal input

TIMx Capture/Compare Register 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1															
rw	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*

Figure A.4: Capture/Compare Register 1 .cf [24, p. 2128]

CCR1 (Capture/Compare Register 1): [24]

The Capture/Compare register contains the value that is to be compared with the counter.

TIMx Capture/Compare Enable Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
										CC6P	*CC6E*			*CC5P*	*CC5E*
										rw	*rw*			*rw*	*rw*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP		*CC4P*	*CC4E*	*CC3NP*	*CC3NE*	*CC3P*	*CC3E*	*CC2NP*	*CC2NE*	*CC2P*	*CC2E*	*CC1NP*	*CC1NE*	*CC1P*	*CC1E*
rw		*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*

Figure A.5: Capture/Compare Enable Register .cf [24, p. 2124]

In the register description, x is a channel number between 1 and 6.

CCxP (Capture/Compare x output Polarity): [24]

If Capture/Compare x channel is set as output configuration:

0: Configures the output to active high

1: Configures the output to active low

If Capture/Compare x channel is set as input configuration:

See [24, p. 2125] for a deeper description

CCxE (Capture/Compare x output Enable): [24]

If Capture/Compare x channel is set as output configuration:

0: Switches the output off

1: Switches the output on

If Capture/Compare x channel is set as input configuration:

0: Disables capture

1: Enables capture

CCxNP (Capture/Compare x complementary output Polarity): [24]

If Capture/Compare x channel is set as output configuration:

0: Configures the complementary output polarity to active high

1: Configures the complementary output polarity to active low

If Capture/Compare x channel is set as input configuration:

See [24, p. 2125] for a deeper description

CCxNE (Capture/Compare x complementary output Enable): [24]

0: Disables complementary output

1: Enables complementary output

TIMx Control Register 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				UIFRE MAP		*CKD*		*ARPE*	*CMS*		*DIR*	*OPM*	*URS*	*UDIS*	*CEN*
				rw		*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*

Figure A.6: TIMx Control Register 1 .cf [24, p. 2106]

UIFREMAP (UIF status bit Remapping): [24]

0: Update interrupt flag is not copied into the Timer Counter register bit UIF (Update interrupt flag copy)

1: Update interrupt flag is copied into the Timer Counter register bit UIF

CKD (Clock division): [24] By the bit field CKD a divison ratio can be determined, which is used between dead time and sampling clock by the dead time generators and the digital filters. To get more information about the divison ratio see [24, p. 2106].

ARPE (Auto-reload preload enable): [24]

0: ARR register is configured as not buffered

1: ARR register is configured as buffered

CMS (Center-aligned mode selection): [24]

For the selection of center-aligned modes, see [24, p. 2106].

DIR (Direction): [24]

DIR bit only used in Center-aligned configuration.

0: Upcounting usage of counter

1: Downcounting usage of counter

OPM (One pulse mode): [24]

0: Continuous counting at update event

1: Stop counting at update event

URS (Update request source): [24]

The USR bit selects the update event source. Events:

0: Counter overflow/underflow, setting the update generation bit and generation of updates by the slave mode controller.

1: Counter overflow/underflow and DMA request.

UDIS (Update disable): [24]

The UDIS bit is cleaned by the software to enable update events. Update events can be generated by counter overflow/underflow, setting the UG bit and update generation by the slave mode controller.

0: Update event enabled

1: Update event disabled

CEN (Counter enable): [24]

0: Disables counter

1: Enables counter

TIMx DMA/Interrupt Enable Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TDE	*COMDE*	*CC4DE*	*CC3DE*	*CC2DE*	*CC1DE*	*UDE*	*BIE*	*TIE*	*COMIE*	*CC4IE*	*CC3ID*	*CC2IE*	*CC1IE*	*UIE*
	rw	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*

Figure A.7: TIMx DMA/Interrupt Enable Register .cf [24, p. 2112]

In the register description, x is a channel number between 1 and 4.

TDE (Trigger DMA request Enable) [24]

0: Disables trigger DMA request

1: Enables trigger DMA request

COMDE (COM DMA request Enable) [24]

0: Disables COM DMA request

1: Enables COM DMA request

CCxDE (Capture/Compare x DMA request Enable) [24]

0: Disables Capture/Compare x DMA request

1: Enables Capture/Compare x DMA request

UDE (Update DMA request Enable) [24]

0: Disables update DMA request

1: Enables update DMA request

BIE (Break Interrupt Enable) [24]

0: Disables break interrupt

1: Enables break interrupt

TIE (Trigger Interrupt Enable) [24]

0: Disables trigger interrupt

1: Enables trigger interrupt

COMIE (COM Interrupt Enable) [24]

0: Disables COM interrupt

1: Enables COM interrupt

CCxIE (Capture/Compare x Interrupt Enable) [24]

0: Disables Capture/Compare x interrupt

1: Enables Capture/Compare x interrupt

UIE (Update interrupt enable) [24]

0: Disables Update interrupt

1: Enables Update interrupt

TIMx Status Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
														CC6IF	*CC5IF*
														rc_w0	*rc_w0*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		SBIF	*CC4OF*	*CC3OF*	*CC2OF*	*CC1OF*	*B2IF*	*BIF*	*TIF*	*COMIF*	*CC4IF*	*CC3IF*	*CC2IF*	*CC1IF*	*UIF*
		rc_w0	*rc_w0*	*rc_w0*	*rc_w0*	*rc_w0*	*rc_w0*	*rc_w0*	*rc_w0*	*rc_w0*	*rc_w0*	*rc_w0*	*rc_w0*	*rc_w0*	*rc_w0*

Figure A.8: Capture/Compare Mode Register 1 .cf [24, p. 2114]

CCxIF (Compare x interrupt flag) [24]

If Capture/Compare x channel is set as output configuration:

0: Counter TIMx_CNT and register TIMx_CCR1 do not have the same value.

1: The counter TIMx_CNT has the same value as the register TIMx_CCR1.

If Capture/Compare x channel is set as input configuration:

0: Input capture not occurred

1: The counter value at the moment of the capture input has been written to register TIMx_CCR1

SBIF (System Break interrupt flag) [24]

It is required to reset the bit before the PWM can be restarted.

0: No system break interrupt pending

1: Break interrupt occurrences if the system break line detected a high level. If the BIE bit of register TIMx_DIER is set, an interrupt is generated.

CCxOF (Capture/Compare x overcapture flag) [24]

0: Overcapture not detected.

1: While the CCxIF bit was set, a counter value has been stored in the TIMx_CCR1 register.

B2IF (Break 2 interrupt flag) [24]

0: No break interrupt pending

1: Break interrupt occurrences if teh break line 2 detected a high level. If the BIE bit of register TIMx_DIER is set, an interrupt is generated.

BIF (Break interrupt flag) [24]

0: No break interrupt pending

1: Break interrupt occurrences if the break line 1 detected a high level. If the BIE bit of register TIMx_DIER is set, an interrupt is generated.

TIF (Trigger interrupt flag) [24]

0: No trigger interrupt pending

1: Trigger interrupt has occurred

COMIF (COM interrupt flag) [24]

0: No COM interrupt pending

1: COM interrupt has occurred

UIF (Update interrupt flag) [24]

0: No update interrupt pending

1: Update interrupt has occurred

DMA stream x configuration register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								MBURST		*PBURST*			*CT*	*DBM*	*PL*
										rw	*rw*		*rw*	*rw*	*rw*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PINCOS		*MSIZE*	*PSIZE*		*MINC*	*PINC*	*CIRC*	*DIR*		*PFCTRL*	*TCIE*	*HTIE*	*TEIE*	*DMEIE*	*EN*
rw	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*

Figure A.9: DMA stream x configuration register .cf [24, p. 1211]

PL (Priority Level) [24]

The higher the bit value of the bit field, the higher is the priority

MSIZE (Memory data SIZE) [24]

00: 8 bit

01: 16 bit

10: 32 bit

11: reserve

PSIZE (Peripheral data SIZE) [24]

00: 8 bit

01: 16 bit

10: 32 bit

11: reserve

MINC (Memory INCrement mode) [24]

0: fixed memory pointer adress

1: memory pointer adress is incremented after each data transfer

PINC (Peripheral INCrement mode) [24]

0: fixed memory pointer adress

1: memory pointer adress is incremented after each data transfer

CIRC (CIRCular mode) [24]

0: none circular mode

1: circular mode

DIR (data transfer DIRection) [24]

00: peripheral-to-memory

01: memory-to-peripheral

10: memory-to-memory

11: reserved

PFCTRL (Peripheral Flow ConTRoLler) [24]

0: DMA controlles the flow

1: Peripheral controlls the flow

TCIE: (Transfer Complete Interrupt Enable) [24]

0: Transfer complete interrupt disabled

1: Transfer complete interrupt enabled

TEIE: (Transfer Error Interrupt Enable) [24]

0: Transfer error interrupt disabled

1: Transfer error interrupt enabled

EN (stream ENable) [24]

0: stream disabled

1: stream enabled

DMA stream x memory 0 address register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
M0A															
rw	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M0A															
rw	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*

Figure A.10: DMA stream x memory 0 address register .cf [24, p. 1215]

M0A (Memory 0 Address) [24]

Base address of the memory area 0 from or to data is read or written.

DMA stream x peripheral address register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PAR															
rw	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PAR															
rw	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*

Figure A.11: DMA stream x peripheral address register .cf [24, p. 1215]

PAR (Peripheral AddRess) [24]

Base address of the Peripheral data register from or to data is read or written.

DMA stream x number of data register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NDT															
rw	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*

Figure A.12: DMA stream x number of data register .cf [24, p. 1214]

NDT (Number of Data items to Transfer) [24]

numbers can be between 0 up to 65535

DMAMUX request line multiplexer channel x configuration register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
					SYNC_ID			NBREQ					SPOL		SE
					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						EGE	SOIE		DMAREQ_ID						
						rw	rw		rw	rw	rw	rw	rw	rw	rw

Figure A.13: DMAMUX request line multiplexer channel x configuration register .cf [24, p. 1236]

DMAREQ_ID (DMA REQuest IDentification) [24]

To select the input DMA request

DMA low interrupt status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
				TCIF3	HTIF3	TEIF3	DMEIF3		FEIF3	TCIF2	HTIF2	TEIF2	DMEIF2		FEIF2
				r	r	r	r		r	r	r	r	r		r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				TCIF1	HTIF1	TEIF1	DMEIF1		FEIF1	TCIF0	HTIF0	TEIF0	DMEIF0		FEIF0
				r	r	r	r		r	r	r	r	r		r

Figure A.14: DMA low interrupt status register .cf [24, p. 1210]

TCIF (Stream x Transfer Complete Interrupt Flag) [24]

0: none conplet transfer event detected

1: conplet transfer event detected

TEIF (Stream x Transfer Error Interrupt Flag) [24]

0: none error event detected

1: error event detected

DMA low interrupt flag clear register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
				CTCIF3	*CHTIF3*	*CTEIF3*	*CDMEIF3*		*CFEIF3*	*CTCIF2*	*CHTIF2*	*CTEIF2*	*CDMEIF2*		*CFEIF2*
				w	*w*	*w*	*w*		*w*	*w*	*w*	*w*	*w*		*w*
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				CTCIF1	*CHTIF1*	*CTEIF1*	*CDMEIF1*		*CFEIF1*	*CTCIF0*	*CHTIF0*	*CTEIF0*	*CDMEIF0*		*CFEIF0*
				w	*w*	*w*	*w*		*w*	*w*	*w*	*w*	*w*		*w*

Figure A.15: DMA low interrupt flag clear register .cf [24, p. 1210]

CTCIF (Stream x Clear Transfer Complete Interrupt Flag) [24]

1: cleans the transfer complete interrupt flag

CTEIF (Stream x clear Transfer Error Interrupt Flag) [24]

1: cleans the error interrupt flag

SPI configuration register 1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	MBR								*CRC*		*CRC SIZE*				
	rw	*rw*	*rw*						*rw*		*rw*	*rw*	*rw*	*rw*	*rw*
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TX DMA EN	*RX DMA EN*		*UDRDET*		*UDRCFG*		*FTHLV*				*DSIZE*				
rw	*rw*		*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*

Figure A.16: SPI configuration register 1 .cf [24, p. 2752]

TXDMAEN (TX DMA stream ENable) [24]

0: disables Tx DMA

1: enables Tx DMA

RXDMAEN (RX DMA stream ENable) [24]

0: disables Rx DMA

1: enables Rx DMA

SPI configuration register 2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AF CNTR	*SSOM*	*SSOE*	*SSIOP*		*SSM*	*CPOL*	*CPHA*	*LSB FRST*	*MAS TER*	*SP*			*COMM*		
rw	*rw*	*rw*	*rw*		*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IOSWP								*MIDI*				*MSSI*			
rw								*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*	*rw*

Figure A.17: SPI configuration register 2 .cf [24, p. 2755]

AFCNTR: (Alternate Function GPIOs CoNTRol) [24]

0: peripheral has no control to the GPIOs

1: peripheral has control to the GPIOs

SPI/I2S control register 1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
															IO LOCK
															rs

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TCRC INI	*RCRC INI*	*CRC33 _17*	*SSI*	*HDDIR*	*CSUSP*	*C START*	*MAS RX*								*SPE*
rw	*rw*	*rw*	*rw*	*rw*	*w*	*rs*	*rw*								*rw*

Figure A.18: SPI/I2S control register 1 .cf [24, p. 2750]

CSTART (master transfer START) [24]

0: master transfer is in idle state

1: master transfer is temporary suspended or running

SPE (Serial Peripheral Enable) [24]

0: disables Serial Peripheral

1: enables Serial Peripheral

GPIO port input data register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	*IDR14*	*IDR13*	*IDR12*	*IDR11*	*IDR10*	*IDR9*	*IDR8*	*IDR7*	*IDR6*	*IDR5*	*IDR4*	*IDR3*	*IDR2*	*IDR1*	*IDR0*
r	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*

Figure A.19: GPIO port input data register .cf [24, p. 1078]

IDR (Port x input data I/O pin) [24]

It contain the input value of the corresponding pin

A.2 STM32CubeMX Configurations note

The STM32CubeMX configuration for the IPCC is shown in figure A.20.

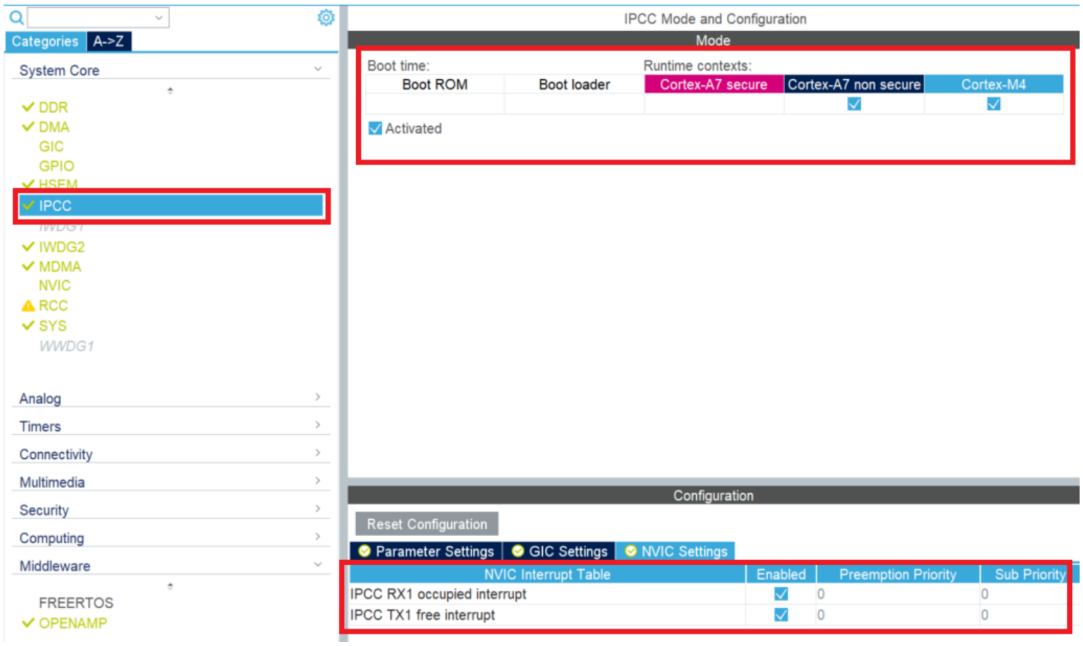


Figure A.20: STM32CubeMX configuration of the IPCC

The STM32CubeMX configuration for the OpenAMP Framework is shown in figure A.21.

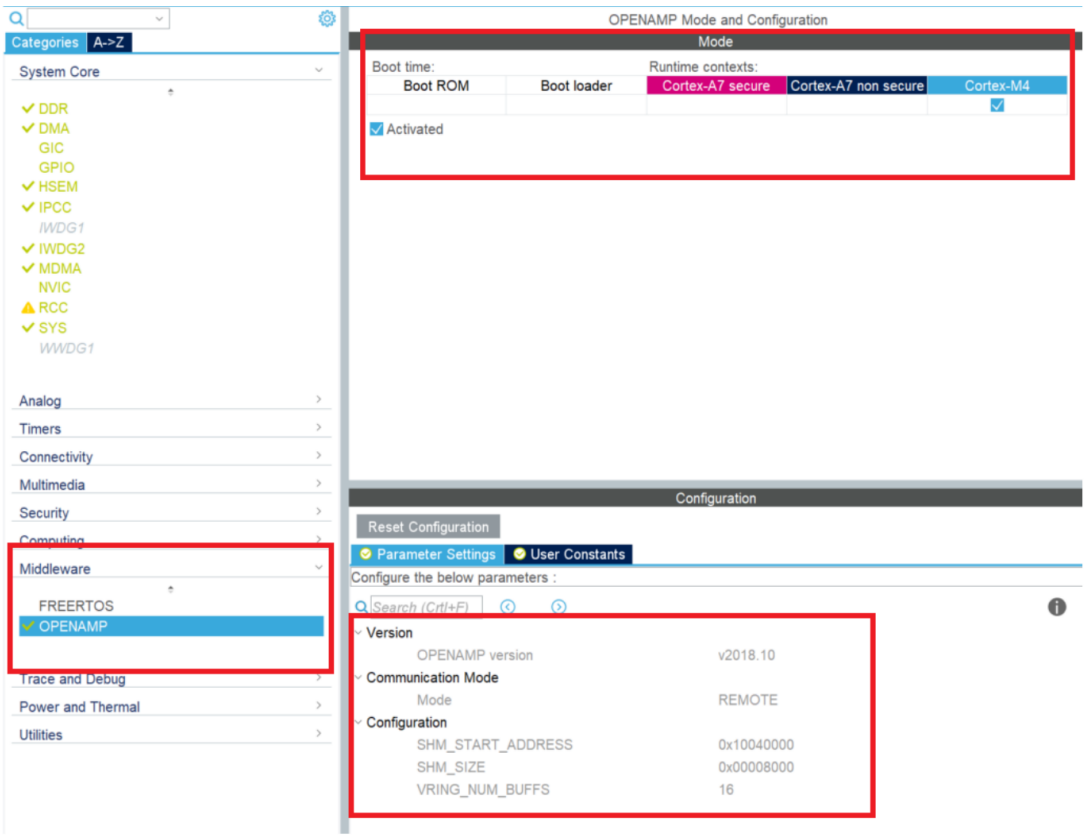


Figure A.21: STM32CubeMX configuration of the OpenAMP Framework

A.3 Attached Data

1 Software requirements specification

2 MATLAB Simulink target, model, and block library `Smart_RCP`