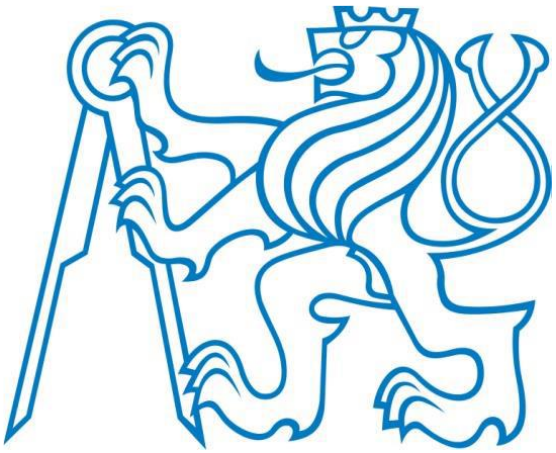
CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF ELECTRICAL ENGINEERING

DEPARTMENT OF MEASUREMENT



MASTER THESIS

CAN FD Gateway

By: Srinath Rangarajan

Supervisor: doc. Ing. Jiří Novák, Ph.D.,

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Rangarajan Srinath** Personal ID number: **481380**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Measurement**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

CAN FD Gateway

Master's thesis title in Czech:

CAN FD Gateway

Guidelines:

Design and develop CAN Gateway module with the following features:
1. Hardware based on STM32 family of microcontrollers with 3 CAN FD interfaces.
2. 12 VDC power supply, overvoltage and polarity inversion protected.
3. CAN FD or RS232 control interfaces alternatively.
4. Support for CAN frame retransmission, blocking, and new frame generation.
5. RS 232 port based control via human readable commands including help.

Bibliography / sources:

[1] ISO11898 standard - Controller Area Network
[2] Etschberger K.: Controller Area Network, IXXAT Automation 2001, ISBN: 978-3000073762
[3] Schwank, F.: Programové vybavení pro CAN Gateway. Diploma thesis ČVUT FEL 2018
[4] Pekárek, D.: CAN Gateway. Diploma thesis ČVUT FEL 2018

Name and workplace of master's thesis supervisor:

doc. Ing. Jiří Novák, Ph.D., K 13138 - katedra měření

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **04.02.2020** Deadline for master's thesis submission: **14.08.2020**

Assignment valid until:
by the end of winter semester 2021/2022

doc. Ing. Jiří Novák, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt	Student's signature

DECLARATION

I hereby declare that this master's thesis is the product of my own independent work and that I have clearly stated all information sources used in the thesis according to Methodological Instruction No. 1/2009 – "On maintaining ethical principles when working on a university final project, CTU in Prague".

Date: **Signature**

ACKNOWLEDGEMENT

In academic life, a student will get exposed to a numerous amount of intellectual influences, making a path to countless ideas and perspectives. In my academic life, one of those influences came from doc. Ing. Jan Fischer, who taught videometry and contactless measurements and doc. Ing. Jiří Novák, Ph.D, who taught the subject Computer Interfaces in the third semester of my masters at Czech Technical University, the subjects aroused my curiosity to explore the concepts and technologies related to sensors and microcontrollers respectively.

I express my utmost gratitude to doc. Ing. Jiří Novák, Ph.D for being my thesis supervisor and for providing essential guidance throughout the course of my thesis.

I am sincerely thankful to all the staffs of the Faculty of Electrical Engineering for providing a base to formulate my thesis.

Lastly, I would like to express my gratitude to my precious family for their impeccable support throughout my life.

Abstract

The diploma thesis deals with design of hardware and implementation of software for CAN FD Gateway. The device is designed for the purpose of testing in automotive industry, where the device acts as a gateway between two CAN FD networks. The device will block, forward or modify the messages between the networks. The rules are provided to the application via the third Control CAN FD network. This work deals with realization of hardware according the application requirements and development of application on the hardware. Several tests are performed and verified to ensure that the gateway satisfies all the functions needed.

Abstrakt

Diplomová práce se zabývá návrhem a implementací softwaru pro CAN FD Gateway. Přístroj je určen pro účely testování v automobilovém průmyslu, kde zařízení funguje jako brána mezi dvěma sítěmi CAN FD. Gatewayumožňuje blokovat, předávat nebo modifikovat zprávy předávané mezi mezi sítěmi. Pravidla jsou aplikaci poskytována prostřednictvím třetí (řídicí) sítě CAN FD. Tato práce se zabývá realizací hardware podle požadavků aplikace a vývojem aplikace pro tento hardware. Dále popisuje provedení vybraných testů a jejich výsledky.

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

Automotive networks strive to satisfy safety and bandwidth needs. A quiet revolution is sweeping through automotive in-vehicle, vehicle-to-vehicle, and vehicle to-infrastructure communications and networking. Companies as well as standards organizations continue to successfully tackle major design challenges, such as the adaption of hardware and software approaches to meet demanding bandwidth, fault-tolerance, determinism, and reliability requirements.

In fact, there's marked improvement among several communications and control protocols, both hardware and software. Designers are now attempting to slim a vehicle's typical 35 to 40 MCUs (up to 100 in some cases) down to about 20 to 25 units that pack the same functionality. As for body, powertrain, chassis, and other controls, no one protocol, and architecture can be considered as a "one size fits all" solution.

Vehicles must pass many tests to meet all demanding and new requirements. This diploma work is focused on designing a CAN FD gateway module for testing in-vehicle networks. The module acts as a bridge between two networks and it is designed considering the current industrial needs, especially low cost and low power consumption.

2. SCOPE & OBJECTIVES

As said by Dr. NIK Dimitrakopolous from Rohde-Schwarz “Autonomous driving and on-board computing technology creates a demand for higher data rate traffic within vehicle communication systems. To verify the functionality and quality of these data streams automotive engineers need to be able to search for specific signals, trigger, decode, measure crosstalk and performance compliance testing” [5]. However, not every component can easily induce an error condition, so the topic of this work is to design and verify a compact and low power module in CAN FD communication protocol which is able to block, modify, or pass messages between two networks and also to create a possibility to simulate the disconnection of some units from the network, or to intentionally disrupt their communication, to send meaningless data, or even valid but incorrect data.

A gateway is a device that separates two networks operating with not necessarily different communication protocols [1]. Within the application, there are 2 FD CAN networks, where within one network the units exchange CAN FD frames. The task of the gateway is then to interconnect 2 such networks - to forward messages according to the requirements of the frame, and to block, release, or modify according to rules provided by the third CAN FD interface, which receives commands from the control system. There are some rules to be active only for a certain period (determined by time or number of frames). Gateway also supports application layer CRC and reports number of transmitted messages. The hardware should be designed and developed using STM32 family of microcontrollers with three CAN FD interfaces along with overvoltage and polarity inversion protection. In order to control the gateway module using UART (alternatively to the third CAN FD interface) it is also necessary to add a slot for UART transceivers.

3. CAN COMMUNICATION PROTOCOL

CAN is a protocol developed by Bosch in 1983 and officially released in 1986. It was released in a later version 2.0 in 1991 (version 2.0A with a standard 11 bit identifier and 2.0B with an extended 29 bit identifier). The main area of application of the CAN protocol is especially in automotive industry, where it is used for communication between electronic components of the vehicles (brakes, engine, steering wheel, etc.)[3].

3.1 Principle

The protocol itself works based on collision resolution (CSMA / CR Carrier Sense Multiple Access with Collision Resolution) method [4], where the transmission uses the so-called wire product. In the idle state, there is a logic 1, or recessive state, on the CAN bus. As soon as a device on the bus wants to start transmitting, it pulls the bus to a logic 0, or to a dominant state. Arbitration of CAN messages resolves the collision, if many devices start communicating at same time Following figure explains aribitration .

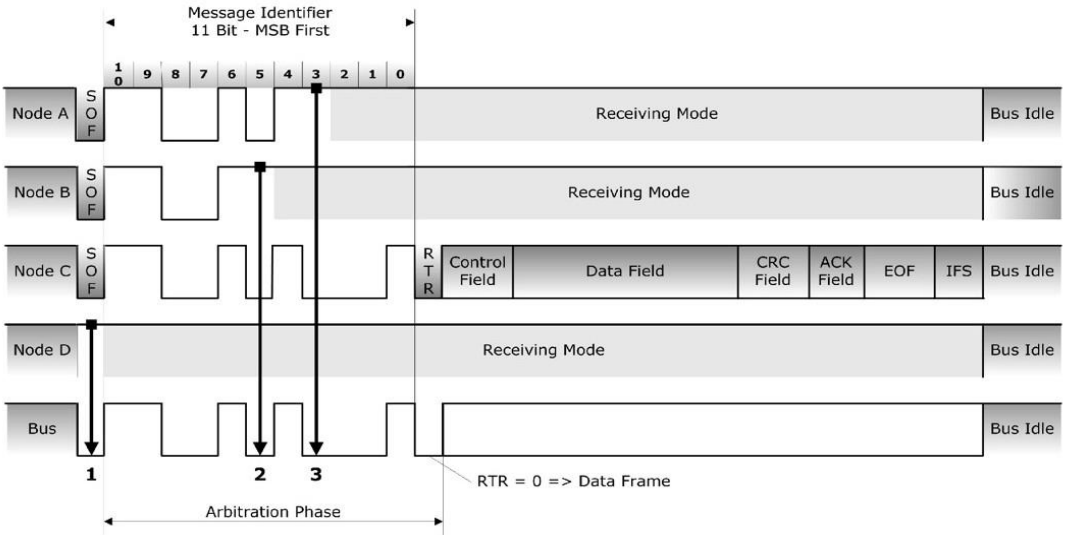


Figure 1- CAN Arbitration

The arbitration of which CAN devices will be transmitting at a given time and the overall addressing of the units are determined using the message identifier (11 or 29 bits). Logic 0 as the dominant bit has a higher priority, so the lower the identifier, the higher the priority. Each device on the bus also reads data from the bus backwards during the transmission, and if, for example, a unit has currently sent a recessive bit of its identifier but reads a dominant bit on the bus, arbitration ends for that unit because it is likely on the bus that the other

device wants to transmit with a higher priority message identifier. To ensure that no collision occurs after arbitration, the identifier must be unique in the CAN network [20][2].

For a node to start communication, it must detect an idle state on the bus. When communication is detected, it may not transmit and, on the contrary, it acknowledges receipt of the message from the bus already within the given message (ACK bit and only after it comes the End of the frame). If the unit detects an error in communication, it also announces it on the bus (by sending 6 consecutive recessive or dominant bits - dominant only if the unit is not in the so-called error passive state). This will also destroy communication for other nodes so that they no longer receive the message, because they do not have to detect the error as well - ie the system works here, when one device detect the error, no one will have the message. Then the message must be sent again. This behavior also has the negative consequence that a faulty unit would constantly destroy communication on the bus. Therefore, error states are also implemented - these are simply counters, how many times the unit has detected an error on the bus and according to the set constants switches between error states (error active state - default error-free, error passive and bus off). In bus off - the unit is practically separated from the bus, it can neither send nor receive, and controller reset is required). From the point of view of the ISO OSI model, the CAN protocol can be classified as follows [2][20]:

- Application layer
    i) Content of frames
    ii) When and under what conditions the frames are sent.
- Data link layer
    i) CAN protocol,
    ii) Media access control - collision
    iii) Addressing - arbitration
    iv) Security
    v) Response to error conditions
- Physical Layer
    i) Bit representation and signal level
    ii) Transmission medium
    iii) Line parameters, connectors, speed

CAN works with four types of frames - data frame (for data transmission with a length of 0 - 8 bytes), data request frame (remote request frame unit requests data with a given identifier), error frame (6 dominant or recessive bits are sent in in case of error detection), overload frame (not used today - same format as error, but the unit thus asks to postpone another frame) [3].

For data security and error detection, CAN uses a CRC checksum (if the received and calculated CRC code differs, an error is detected). NRZ (not return to zero) encoding as well as a bit stuffing is used. transmitting If 5 or more bits of the same level are transmitted, the opposite bit is inserted - because CAN is not synchronous (ie the clock signal to which the bus would be synchronized is not distributed), it is necessary to change the state of the bus so that the synchronization is available and data is read correctly. Another effect of bit stuffing is that 6 bits of the same value in a row are already detected as an error [20].

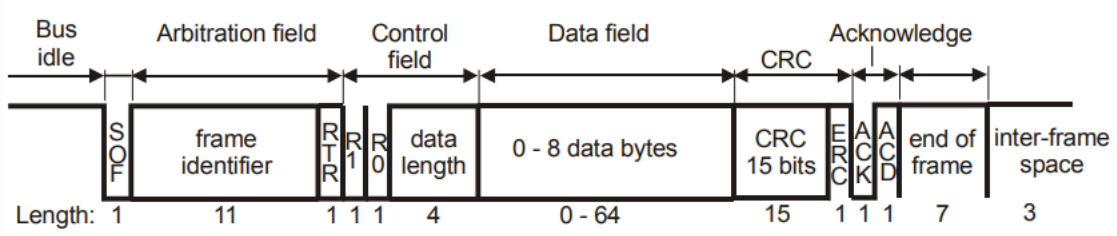


Figure 2- Classic CAN data frame

As mentioned above, CAN is not synchronous, but due to the different distances of the units on the bus, it is necessary to compensate for the delay of the signals given by their propagation over the physical channel. To do this, CAN uses so-called Time quanta - short time periods to compensate for signal propagation. In essence, there is a pre-divider of the frequency on which the unit runs, and within the settings, the number of these quanta is determined to correctly define the Sample point - the point where the current state of the bus is taken as a valid bit value within the bit interval. This process also synchronizes the units with each other. However, if the devices are too far apart when using a given nominal baud rate, then the maximum number of time quanta may not be enough to compensate, and it is necessary to reduce the overall communication speed [22].

It is also necessary to add terminating resistors - terminators for the correct function of the CAN bus. They prevent signal reflections on the bus ends by simply converting electric

energy into heat and thus attenuating the reflection. Usually a 120 Ohm resistor inserted on both sides of the bus between the CAN_H and CAN_L wires is used. The channel then uses differential (logic level is given by voltage difference) communication between these conductors [20].

It also follows from the description above that at least two units are required for the CAN bus to function properly. One that broadcasts and another that acknowledges the frames.

3.2 CAN FD version

Because today's vehicles demand more and more data throughput, the original 1 Mbit/s CAN specification may no longer be enough. Therefore, in 2012, Bosch came up with another extension that covers the area between standard CAN and more expensive technologies such as FlexRay, Ethernet, etc. At the same time, switching from CAN to FlexRay or other technology could be costly for hardware and implementation. Therefore, the FD version has higher throughput, but still provides backward compatibility with classic CAN units [19]. Compatibility only works if standard CAN frames are transmitted, the older unit does not recognize the FD frames and reports errors. As for arbitration and message acknowledgment, CAN FD does not change this function in any way. What has changed is the bit rate switch, the data frames in CAN FD version transfer from the original 0 - 8 bytes the new 0 - 8, 12, 16, 20, 24, 32, 48, 64 bytes encoded in the length field (DLC field). Therefore, it is now possible to transfer much more data per time period corresponding to the transmission of up to 8 bytes of the original CAN protocol. From a software perspective, nothing needs to be changed unless higher speeds are required. From the hardware point of view, it is necessary to change the physical transceiver (driver) - if a speed higher than 1 Mbit/s is used [5][24].

Figure 3- Classic CAN vs CAN FD –with BRS

With the classical 11 bit and extended 29-bit identifiers, the beginning of the frame is the same for both CAN versions. The difference between RTR and RRS (Remote request) is only in the rename and is usually dominant for data frames. The real change is in bit r0 / FDF, in the original CAN this bit was reserved and in log. 0, in CAN FD it means Flexible Data Format - it determines whether it is a standard or CAN FD frame. This bit is followed by DLC for standard and res bit for FD, so the classical CAN units at this point misinterpret the frame and report an error. Conversely, CAN FD units can work with both formats. The last parts of both the CAN frames are identical[27].

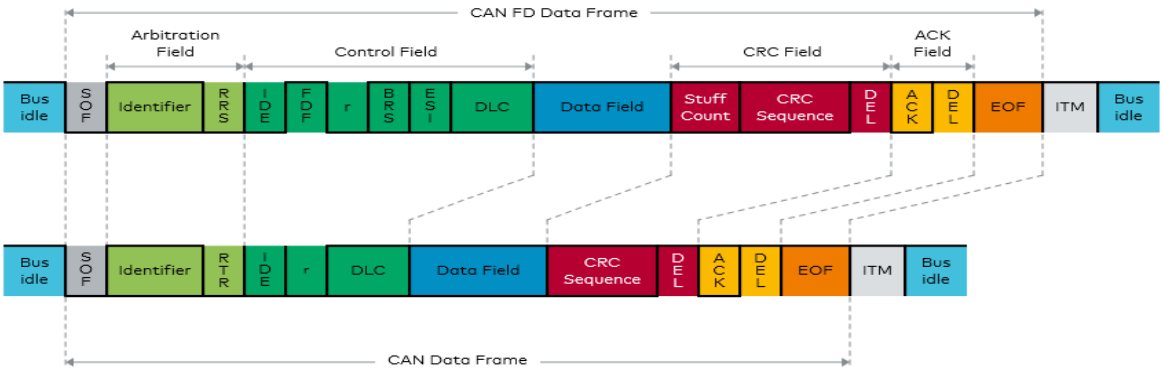


Figure 4- CAN Classic Vs CAN FD

3.3 Merits & Demerits.

Like any other network protocol used in automotive industry CAN and CAN FD has its own constraints and advantages. It is used to reduce wiring in various automotive applications. Due to less complex interface, it is widely used across various industries. It supports auto retransmission of lost messages; it works in various electrical environments without any issues.

Though maximum number of nodes is not specified for the network, it supports up to 64 nodes due to electrical loading. Node removal requires use of termination resistors of 120 Ohm value at appropriate places on the CAN bus. Network should be wired in topology which limits stubs as much as possible.

3.4 Application CRC

CRC, or Cyclic Redundancy Check is a function most often used in communication protocols to ensure error-free data transmission - resp. to detect a transmission error (due to noise and other negative effects on the transmission channel). The goal is to maximize the Hamming distance (the number of bits by which one codeword is changed to another codeword - the greater the distance, the greater the probability of error detection) and to minimize the size of redundant (redundant) data.

In general, the data is represented as a binary polynomial M (x) of degree n - 1 where the values of the coefficients of the polynomial correspond to the individual data bits. This data polynomial is then divided by the generating polynomial G (x) of degree k. The remainder after dividing the polynomial M (x) by a polynomial G (x) = R (x) of length k is added as redundant information to the data and the whole codeword of length $n + k$ is then sent. As soon as the other party receives this data, it also calculates the CRC, and if the residues after division differ, an error has occurred. If they are equal, an error could still have occurred, but it was not detected and depends on the CRC used and the probability of the error, whether it is enough for the given application requirements.

Because the checksum at the CAN protocol level is sometimes not enough in terms of application requirements (the probability of error is still too high), the application CRC is also introduced, so the CAN peripheral automatically calculates and verifies the CRC and reports the error when receiving the frame.

4. COMMANDS

As mentioned in previous chapters, the application acts as a Gateway on the CAN FD protocol. Materials of older version of CAN Gateway, CAN FD gateway with UART interface and CAN FD gateway module based on different microcontroller were provided for this work. The aim of this work is to design and develop a module capable of working with three CAN FD interfaces up to 2 Mbits/s bitrate. A gateway module between networks will have many tasks to perform according to requirements. Firstly, it is necessary to understand the requirements for the gateway module, which are blocking, forwarding and modification of messages passed between the networks. The above processes may be depending on some variables which are called parameters. We use commands to represent these processes and each command will have its individual parameters. All the commands are sent to the module via user or tester during the testing of network. By default, the application passes all frames in both directions. If it receives a command from the third CAN (FD) interface, it will be preserved as follows - the commands defining the new rule will only be saved for the time being, only after receiving the Trig command the new rules are applied. There is also a Reset command that clears saved rules and sets only the default rule for working with frames (block or forward all). All commands are described in more detail on the following pages.

4.1 Definitions

The CMD field is one byte of data, each configuration command contains a CMD field and then up to 7 parameters (depending on the command type). CMD byte is further divided into the command itself (its type is encoded in the upper 4 bits) and the decreasing frame sequence number - if it is necessary to divide the configuration message into multiple frames (data exceeding 8 or 64 bytes - for FD). In such a case the first packet has the sequence number N-1, where N denotes the number of frames. The value therefore gradually decreases to 0. The CMD field is therefore included in each frame to ensure the correct delivery of all data. The following table shows the commands and their codes[21]:

Command Code	**Name**	**Function**
0x00	Reset	Reset the module to default state.

0x01	Block/Pass	Request to block/forward message.
0x02	Modify	Request to modify the forward message.
0x03	Trig	Request to activate he preset function.
0x04	Stat	Request to send the statistics.
0x05	Send	Sends a message to the specified CAN once
0x06	Read	Activates message mirroring on CAN control

Table 1Command types

At the same time, because the CMD field also contains the type of command, it simplifies the implementation of data breakdown. Parameters have a variable length and number, depending on the type of command. The minimum size of the parameter is 1 byte. The following format is used to confirm the command - again the CMD field (upper 4 bits of the command type to which it corresponds, lower 4 bits always 0 - there is no need to count frames here). The confirmation also has 2 parameters, where the status register of the CAN peripherals is inserted, or statistics [21].

4.1.1 RESET

The single parameter Reset command sets the default blocking or forwarding rule. It deletes specific rules (currently being developed). The following table describes the Reset command and its parameters [21]:

Parameter	Size	Value	Function
State	1B	0x00	After the Trig command, all frames are forwarded in both directions.
		0x01	After the Trig command, all frames are blocked in both directions.

Table 2- RESET command parameters

4.1.2 Block/Pass

The Block / Pass command specifies a rule for exception to the default behavior (specified after power-up or with the Reset command) for certain messages with a given identifier.

Rules of this type are saved and are active after the Trig command. The Reset command clears the pending rules. The following table clearly describes the functions of the Block / Pass command [21]:

Parameter	Size	Value	Function
ID	4B	11- or 29-bit ID	Identifier whose frames will be blocked or released using this rule - the opposite of the default rule.
Type	1B	0x00	Permanent rule without time limit
		0x01	A rule limited by the number of frames in the Length parameter.
		0x02	A rule limited by the time given in the Length parameter
Length	2B	Number of times	When Type = 0x01, the Length parameter specifies the number of frames to which the rule applies and then terminates.
		Usage time(ms)	When Type = 0x02, the Length parameter specifies the time in milliseconds for which the rule is active and then ends.

Table 3- Block/Pass command parameters.

4.1.3 MODIFY

The Modify command is probably the most complex, but also the "most powerful" rule in the application. It has several functions, such as modification of data based on the mask and rule data, it is possible to modify the application CRC, either to correctly calculate the new CRC, or intentionally wrong. Like the Block / Pass command, modify has parameters for a time- or number-limited duration. Unlike Block / Pass, however, modify is not dependent on the default rule given by the activation or Reset command.

However, there are some pitfalls with this command, if for some reason it would be necessary to transfer the rule only with packets smaller than 8 bytes, then it would not be possible to use the frame counter as designed (at 8 bytes, where the first byte is always an

array CMD, so effectively 7 bytes is necessary to transfer 138 bytes of data Modify, which is 20 frames, but the counter is only 4 bits, ie a maximum of 16 frames).

Therefore, the counter would have to be 5 bits for the frame order and 3 bits for the command type. If it is ensured that at least 12 bytes of data are transferred (11 bytes effectively, ie 13 Modify rule frames) in each frame, the CMD field does not need to be changed. The below table again clearly describes the parameters of the Modify command [21].

Parameter	**Size**	**Value**	**Function**
ID	4B	11- or 29-bit ID	An identifier whose frames will be modified using this rule
Mask	64B	Mask	Data mask, to replace the rule data.
Data	64B	Data	Rule data to replace frame data (by mask).
DLC/CRC	1B	Data length/CRC	The upper 4 bits determine the modification of the data length, here in the original application the range 0-8 applied, the value 0xF left the original length. New for CAN FD, this field must be extended to use standard length coding (see 2.3.2). In the new application, 0x10 means size without change and, among other things, it is necessary to change the encoding to the upper 5 bits are the length and the lower 3 bits the CRC. If CRC = 0x00, only data and mask modifications are performed. If CRC = 0x01, the CRC is correctly calculated from the modified data, and finally, if CRC = 0x02, then the calculation is intentionally incorrect (correctly calculated and incremented by 1).
Seed	16B	CRC Seed	Initialization vector for application CRC calculation.
Type	1B	0x00	Permanent rule without time limit

		0x01	A rule limited by the number of frames in the Length parameter.
		0x02	A rule limited by the time given in the Length parameter.
Length	2B	Number of Usages	When Type = 0x01 Length parameter determines the number of frames to which the rule applies, and then ends
		Time of Usage(ms)	When Type = 0x02, the Length parameter specifies the time in milliseconds that the rule is active and then ends.

Table 4 - MODIFY command and its parameters

4.1.4 TRIG

The Trig command has no parameters, so far it only activates saved settings obtained after the previous Trig command or after switching on the device.

4.1.5 STAT:

This command has a single parameter that decides whether it is to activate or deactivate the sending of statistics on the number of messages on both CAN interfaces. After receiving this command, the application confirms by default using ACK / NACK, but the parameters do not contain the CAN (FD) status registers of the controllers, but statistics for both interfaces. The average number of frames that pass-through a given interface per second is calculated[21].

Parameter	Value	Description
Type (1B)	0x00	Statistics transmission is off
	0x01	Statistics transmission is ON

Table 5 -STAT command Parameter

4.1.6 ACK/NACK:

After each command, the module sends an acknowledgment at the application level, together with the error ID that may have occurred and with the status registers of both CAN (FD) interfaces. The following table summarizes the format of the confirmation message:

Parameter	Size	Value	Function
ErrID	1B	Error ID	Indicates the error ID that has occurred since the last ACK / NACK acknowledgment. If no problem occurs, the ID is 0
CAN1 FD ERR	2B	Error register	Status error register CAN1 FD interface
CAN2 FD ERR	2B	Error register	Status error register CAN2 FD interface

Table 6 -ACK/NACK commands

4.1.7 Trig Broadcast

The Trig command also supports broadcasting, where module address 0 is reserved for broadcasts. On Trig, either the unicast or broadcast, the module responds in the standard manner described above [21].

4.1.8 SEND

This command can be used to immediately send a message from the control CAN to the required CAN FD interface. If the entire command does not fit in a single frame, it is divided into two, just as with the Modify command. A detailed description is provided by the table:

Parameter	Size	Value	Description
ID	4B	Frame Identifier	Identifier of the frame to be sent. The frame in the extended identifier format has the highest bit in one, the standard in zero.

DLC	1B	Length/Target	The length of the frame data is encoded in the lower 4 bits. The next bit in ascending order (5thbit.) determines the frame format: 1 - CAN FD, 0 -Standard CAN. (6thbit) specifies BRS, only make sense for FD frame. (7thbit) specifies target CAN interface: 1 CAN2, 0- CAN 1 (8thBit) not used
Data	64B	Data	Generated frame data

Table 7 – SEND command parameters

4.1.9 READ

This command can be used to listen to selected messages, ie to mirror them on the control CAN. A detailed description is provided by the following table:

Parameter	Size	Value	Description
ID	4B	Frame Identifier	Identifier of the frame that we want to mirror on the control CAN. The frame in the extended identifier format has the highest bit in one, the standard in zero.
Type	1B	0x00	A parameter without time limit
		0x01	A rule limited by the number of frames in the Length parameter.
		0x02	A rule limited by the time in the Length parameter.
Length	2B	Number of usages	If Type == 0x01, the parameter defines how many times the frame should be forwarded.
		Time of usage(ms)	If Type == 0x02, the parameter defines the time in milliseconds for which the frame is to be forwarded.

Table 8 – READ command parameters

4.2 Gateway Identifier:

The Gateway module identifier is an extended 29 bit. The 20 highest bits are 0x40000, followed by 5 bits - the module address determined by the hardware DIP switch. The next 3 bits are again zero, the last bit has a value of 0 if it is a command and a value of 1 if it is an ACK / NACK acknowledgment. The following table summarizes the identifier format [21]:

20bits	5bits	3bits	1bit
010…..00	DIP address	000	ACK/NACK

Table 9- Identifier Structure

5.Software Design

Even though the software layer has been designed previously and verified, it is necessary to understand how the software is designed so that it will be easy to select the microcontroller with exact features which will ensure smooth functioning of the application layer.The application has several functions, after switching on the application it is necessary to initialize and activate all interfaces (CAN FD peripherals, timers, Internal watchdog). Then follows an infinite loop, which gradually asks all CAN FD interfaces for new data, if there are any, it is necessary to process them accordingly. First, the application queries the control CAN FD interface, where the system commands and rules come from. Data processing of the control CAN FD interface takes place in the function Parsing_Rule. Furthermore, the application queries both CAN FD peripherals, between which messages are forwarded. For both interfaces, the GW_transfer_msg function is used to process new data. This ends the main loop and at the end of every loop the watchdog is refreshed [21].

5.1 Outline

The main part of the application is hidden inside the functions Parsing_Rule and GW_transfer_msg. For the whole application to work as it should, there are several more functions between Transmit / Receive and forwarding / parsing functions. Here the idea is very easy, it is the actual execution of the command, or its decoding. In the case of Block / Pass, the data of a given message is simply copied to the data to be sent on the second interface than the one from which the message came (function GW_Passing_msg). For the Modify command, there is the GW_Modify_msg function, which in addition to copying also modifies the data based on the mask and rule data, or shortens or lengthens the length (DLC parameter) and finally calculates the CRC if required - either correctly or intentionally incorrectly (according to the command CRC parameter) with the help of the initialization vector (CRC Seed). The last important function is GW_Progress_Mod_rule, which uses the global indexes of the developed rule (index for the position in the sheet and index for the position within the structure of the rule) and processes the parameters of the Modify command, which must be divided into several frames (max. data length is 64 bytes, ie at least 3 frames for Modify). The function gradually completes the structure of the new rule, once it is completed, the indexes are reset. Only one Modify rule can be developed at a time [21].

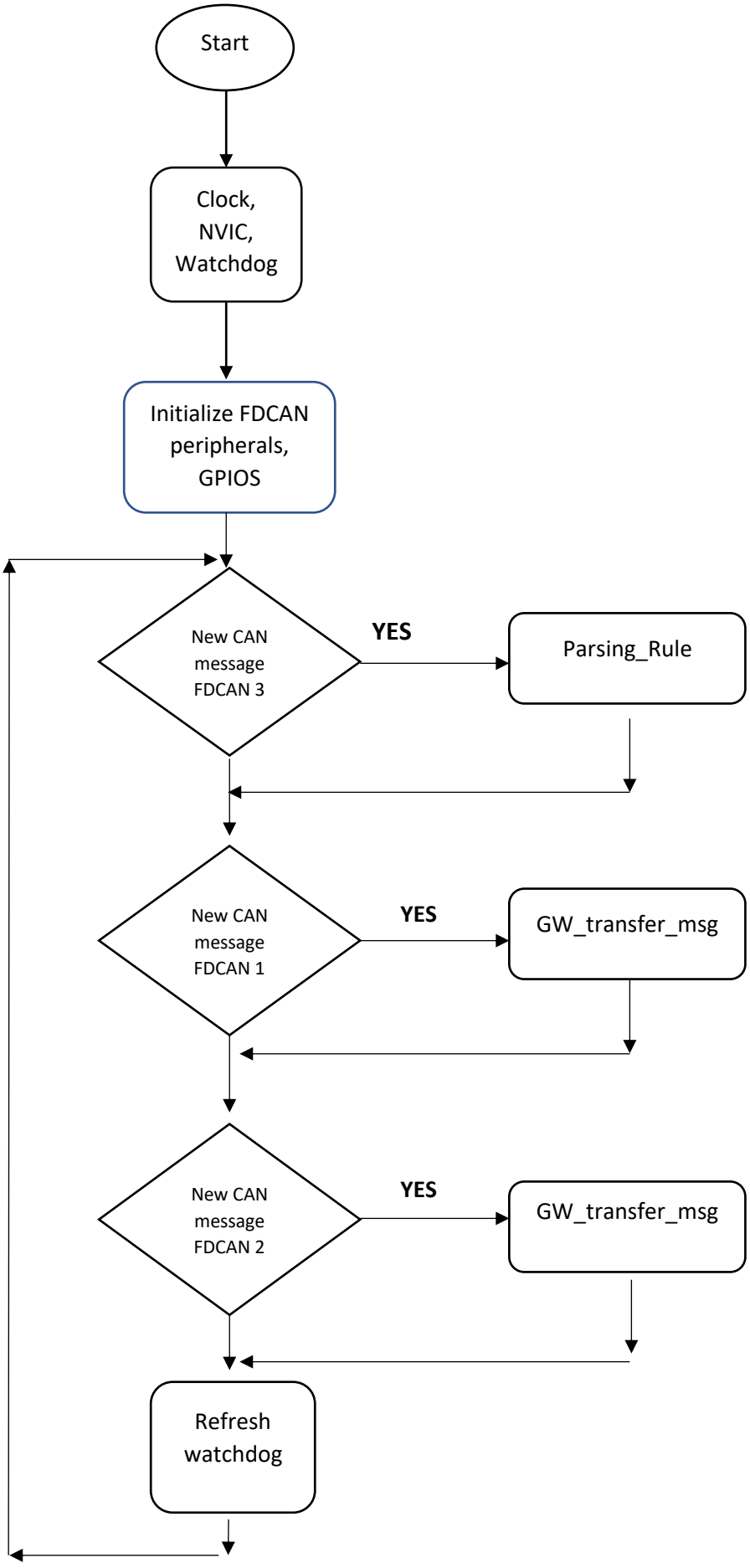


Figure 5- Gateway Application Overview

5.1.1 GW_transfer_msg:

The GW_transfer_msg function (see Figure 9) takes care of the actual message forwarding between CAN FD 1 and 2 interfaces. It is decided on the basis of currently valid rules given within the static sheet up to 10 items (theoretically you can specify any number of rules, because it is given a constant, how many rules can be applied at one time - of course with a large number of rules it can affect the overall performance of the application and it is also necessary to make sure that there is enough memory).

The implementation of the GW_transfer_msg function proceeds as follows. First, the necessary variables are initialized and mainly the ID of the message to be decided is taken. It is also necessary to go through the entire active rule sheet to see if there is a special rule for the given message ID. If no such one is found, the default rule is followed (blocking or passing rule given after activation or the last Reset command). If the ID matches one of the rule IDs in the worksheet, it is further decided which rule it is and how to handle the message. It is decided using an enumeration type that defines the state of the rule. The state is a combination of several parameters of the rule for easier breakdown. Rules can be of the block / pass, modify, or read type and can be valid indefinitely, for a certain time, or the number of messages - a total of 6 types of rules [21].

For Block / pass rules, the status of the default rule must also be considered, as the application must be reversed - a release Reset blocks the message with the given ID. For Modify rules, the rule is applied independently of the default Reset rule.

In terms of time types of rules, unlimited ones always apply to the given ID, but for rules limited by number, the Length parameter must be reduced, as soon as 0 occurs, the rule is deleted and the message is already processed as without a special rule. Time-limited rules are deleted by the timing function (the timer ISR takes care of deleting the rule). Again, the Length parameter is reduced here, and the ISR function clears the rule when 0 occurs. So, in decision-making, the time rule is applied just like the unlimited one [21].

5.1.2 Parsing_Rule:

The Parsing_Rule function is the second important part of the application. As the name suggests, it is used to decode a new rule that came from the control CAN FD interface. All rules can be transferred in a single frame (64 bytes), except for the Modify rule, which must be split in 3 frames. Implementation first, the variables are initialized and the command itself is selected from the data area of the message, which is always contained in the first byte of data - this also facilitates the breakdown. According to the order code, it is then decided which order it is, and it is then processed accordingly. The easiest rules in terms of processing are Reset, Trig, Stat. The reset simply sets the flag that it has been executed and also sets a new default rule (only saved, applied only after the Trig command) given by the second byte of the message data. The Trig command acts as a trigger for a new saved setting, so its processing is also very simple. If the Reset flag has been set, the Trig command activates the new default rule and the pointers to the active and new setting sheets are swapped. The new sheet (at the moment with the old settings) is deleted [21].

A Block / Pass rule already has more parameters than simple statements, and it is also one of the types of rules that are stored in a worksheet. First, you must specify the ID to which the rule will apply. The entire sheet must be searched if a rule with the same ID no longer exists, if so, an error ID is set, and an error is sent in the next confirmation. The error is also generated if no space is found in the worksheet. If the place exists and there is no conflicting ID, it is only necessary to determine the status of the rule (ie enumeration type - combination of command code and validity type). Once the status is set, a confirmation is sent, and the new Block / Pass rule is processed [21].

The most complicated implementation is the processing of the Modify rule because it is divided into several frames. Therefore, auxiliary global variables have been introduced that specify the index of the rule in progress and the index into the structure of the rule itself, to which additional data is written to the appropriate parameters. First, it is necessary to find out, on the basis of the mentioned global variables, whether a Modify rule is already being processed - that is, whether it is another frame of one rule or a new one is accepted. If this is a new rule, it is necessary, as with the Block / Pass rule, to find an empty space in the worksheet (otherwise error), then set the empty space ID to 0, initialize global variables 0). Subsequent calls to the GW_Progress_Mod_rule function move the data from the message

into the rule structure. Only at this point is it possible to verify whether the rule with this ID is no longer in the worksheet. If such a rule already exists, an error is set and the unfinished rule is discarded and deleted from the worksheet. If there is no problem, the next frame is waiting, ie the next part of the Modify rule. When the next part arrives, the GW_Progress_Mod_rule function takes care of inserting the data into the structure again. If the rule is complete, it is necessary to decide again on the status of the rule, and since the time type is already known, it is possible to determine the status at this time. Global variables are set to invalid values, which handles the Modify rule[21].

5.2 Statistics Implementation:

To monitor the load of individual CAN FD interfaces, between which messages are forwarded, a simple statistic has been implemented. This function is again taken care of by a timing function (Timer interrupt service routine), which, in addition to monitoring time rules, simply adds to the circular buffer every 100 millisecond the number of received messages on each interface separately. If statistics are subsequently requested, the average number of received messages per second is sent instead of the error register. The averaging is taken over the entire circular buffer [21].

6. HARDWARE

Hardware is designed and developed specifically for this project. The components are carefully selected in order to maintain the operating voltage of individual components at 3.3V. The board itself contains only a few basic components. The main heart of the system is a microcontroller that controls the entire application and communication, while the test system itself has a distribution of 12 volts, so it is necessary to use a voltage regulator for the microcontroller. For debugging purposes SWD is used. LEDs are used for indicating various states of the system. The CAN interface is used for communication with the world, but it also needs a so-called transceiver as the periphery of the microcontroller - or conversion of CAN signals to differential signals CAN_H and CAN_L.

We have only few requirements for our work. We must focus only on power, cost and availability of the suitable components. Following the guidelines from the project guide and some references from arm community [14], there are some steps to be followed before selecting hardware components. After following the steps such as identifying memory needs, cost effectiveness, availability of peripheral interfaces, code development environment and resources from microcontroller manufacturer few microcontrollers from ST's STM32G0 Series, STM32G4 Series, STM32H7 Series, STM32L5 Series[15],[15] NXP's LPC546xx MCU family, Cypress Traveo family MCU's were some of the shortlisted microcontrollers shortlisted during the initial stages of the work.

All these MCU's have Cortex processors of various generations and versions. We must select microcontroller which has to be suitable for our work, also having a more powerful processor which is not required for our work and it will increase the cost. The previous version of the gateway module was implemented using Cypress's Traveo family [17].

According to the guidelines given for this work is, to design a hardware using STM32 family because ST's resources, online community, documentation and the MCU's are suitable for the application requirement therefore we have decided to proceed with MCU from STMG4 series Microcontrollers. ST not only provides better resources for coding, it also provides better development environment [18].

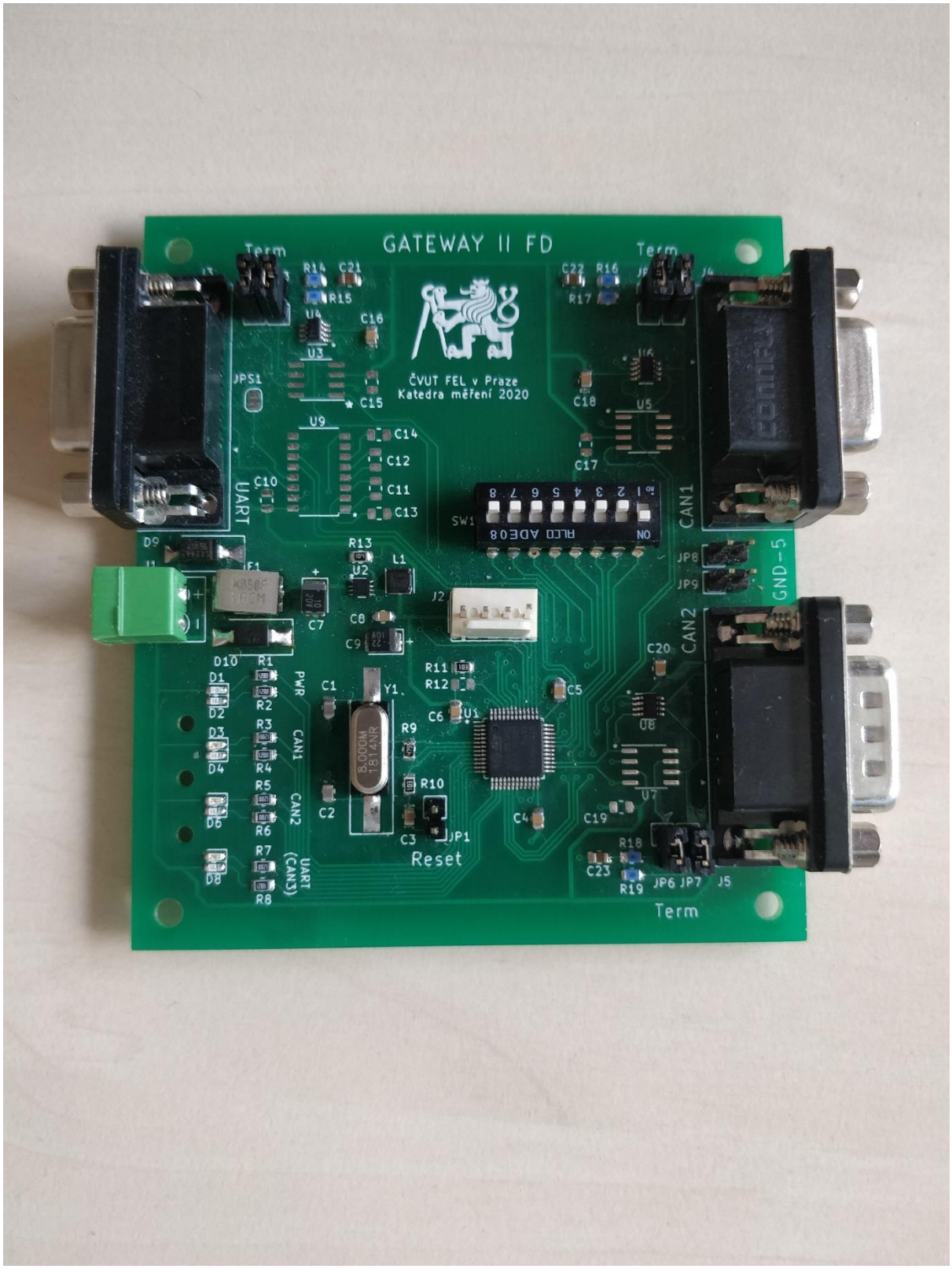


Figure 6 - Hardware

The hardware shown in the above Figure is designed using KICAD software. The board schematic and PCB layout are attached in the appendix of this document (A) (B). The hardware is designed for controlling the gateway using a third control CAN FD and (alternatively) the UART interface. In order to switch between these two interfaces easily the UART and CANFD transceivers are connected the same pin outs of the microcontroller. Here in this work the gateway module is implemented using only CAN FD interfaces. There are totally 8 LEDs used; out of those 6 LEDs in pairs are used to indicate the error states and reception of messages respectively of the three CANFD interfaces. One LED is used to indicate power and other is used to indicate any other errors state of the system.

6.1 Microcontroller

The board is equipped with a STM electronics microcontroller. The basis is ARM Cortex-M4 core. In order to understand more about the peripherals and programming the chip for user application it is necessary to refer to the programming manual provided for this chip. The exact variant is STM32G474CB [8], its key features are:

- External memory interface for static memories FSMC supporting SRAM, PSRAM, NOR and NAND memories
- 12 Kbytes of Flash memory with ECC support, two banks read-while-write.
- Up to 107 fast I/Os
- 5 x 12-bit ADCs 0.25 µs, up to 42 channels.
- 6-channel DMA controller
- x ultra-fast rail-to-rail analog comparators
- 5 x USART/UARTs (ISO 7816 interface, LIN, IrDA, modem control)
- 17 timers
- 3 x FDCAN controller supporting flexible data rate
- 6 x operational amplifiers that can be used in PGA mode, all terminals accessible
- Internal voltage reference buffer (VREFBUF)
- True random number generator (RNG)
- CRC calculation unit, 96-bit unique ID
- Development support: serial wire debug (SWD), JTAG, Embedded Trace Macrocell™

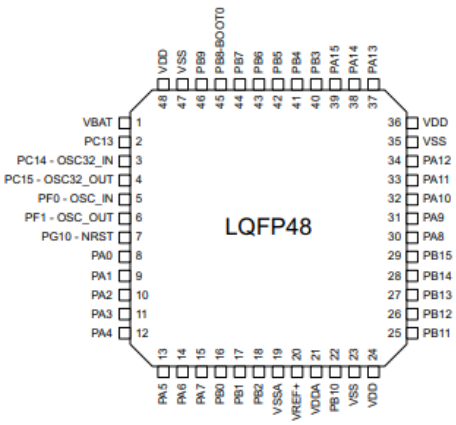


Figure 7-Microcontroller -STM32G4474CB

6.2 Voltage regulator

A voltage regulator is required for the processor to function, which converts the 12-volt input voltage to 3.3V, from which the processor is powered. To meet out voltage constraints the board is equipped with TPS6217x device family from Texas instruments, an easy to use synchronous step-down DC-DC converters optimized for applications with high power density [9]. The key features are:

- Wide input voltage range - from 3Volts up to 17 Volts.
- Up to 500-mA Output Current, Adjustable Output Voltage from 0.9 V to 6 V
- Three outputs - one with a current of 2 A, two with a current of 1 A.
- Operation at up to 100% duty cycle.
- Short-circuit protection.
- Overtemperature protection
- Applications - Standard 12-V Rail Supplies, POL Supply from Single or Multiple Li-Ion Battery, LDO Replacement, Embedded Systems

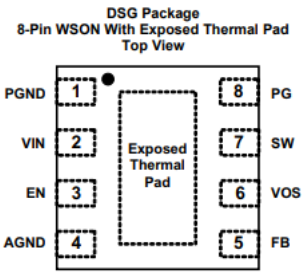


Figure 8-Voltage Regulator

6.3 CAN Transceiver

For the processor to be able to communicate with the rest of the world, and to communicate with the connected CAN networks, a transceiver is required with CAN FD support. For this sole purpose the board is equipped with TCAN337G transceivers from Texas Instruments. The key features are [10]:

- 3.3-V Single supply operation
- Data rates up to 5 Mbps
- Four operating Modes
- Wide common mode range of operation ±12 V
- Undervoltage protection on VCC
- Thermal shutdown protection
- Current limiting on bus pins

Figure 9- CAN transceiver

7. IMPLEMENTATION in HARDWARE

The implementation of the firmware architecture of the CAN FD Gateway on real hardware - the board described in the previous chapter, will be the last step of the diploma. This step is broadly classified, the selection of the programming environment and then the activation of individual peripherals for our application requirement.

7.1 Development environment - STM32CUBE

C is much more suitable for programming and assembler more debugging of the program, where the compiled code can be read in the so-called disassembly view also, C language is the most widespread in the field of microcontrollers. The whole application is written in C language. The STM32 family of processors is currently supported by many environments such as the IAR Embedded Workbench, Atollic, STM32CubeIDE etc. For this work STM32CubeIDE – part of STM32CUBE package, is used. This environment is like one-stop solution for embedded developers. It is a set of tools and embedded software bricks available free of charge to enable fast and easy development on the STM32 platform which simplifies and speeds up developers' work. In the following Figure from ST, the advantage of CUBE library is shown graphically[26].

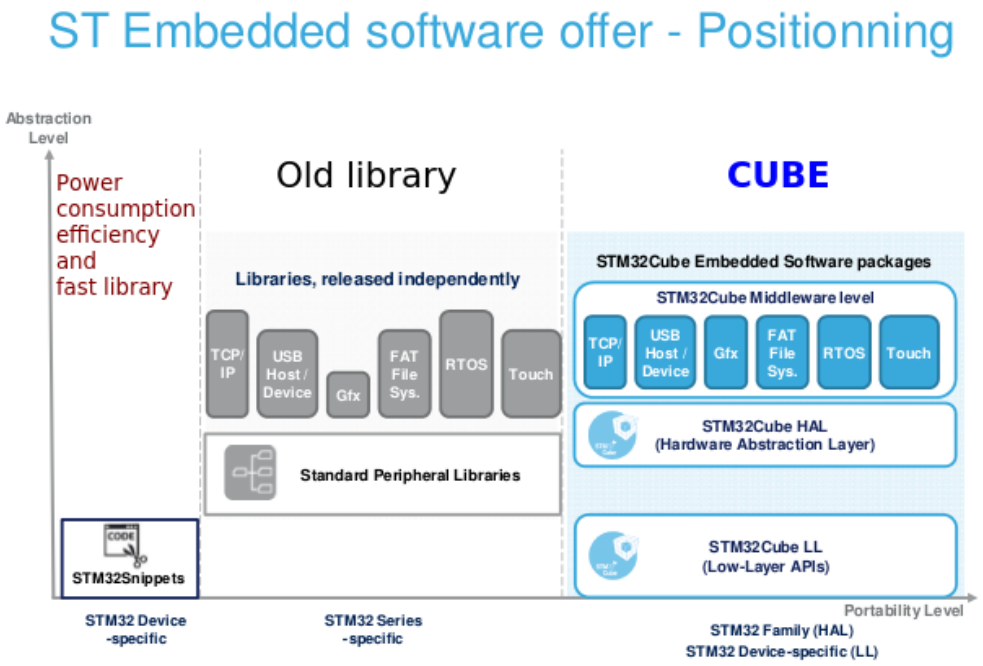


Figure 10– STM32CUBE-Layer

Generally, we need Standard peripheral libraries for the specific MCUs and MPUs. The STM has created an abstraction layer which enables the developer's code to migrate from one MCU to the other easily. The STM32Cube comes with STM32Cube MCU and MPU packages for each individual STM32 MCU and MPU series. This package includes the Hardware abstraction Layer (HAL) and CMSIS- CORE enabling portability between different STM32 devices via standardized API calls [13]. HAL for cortex-M processor registers includes

standardized register definitions for NVIC, system control Block registers, SYSTICK register, MPU Registers and several NVIC and core feature access functions [14]. Low-layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency. The below Figure will give a birds-eye view of the points discussed above.

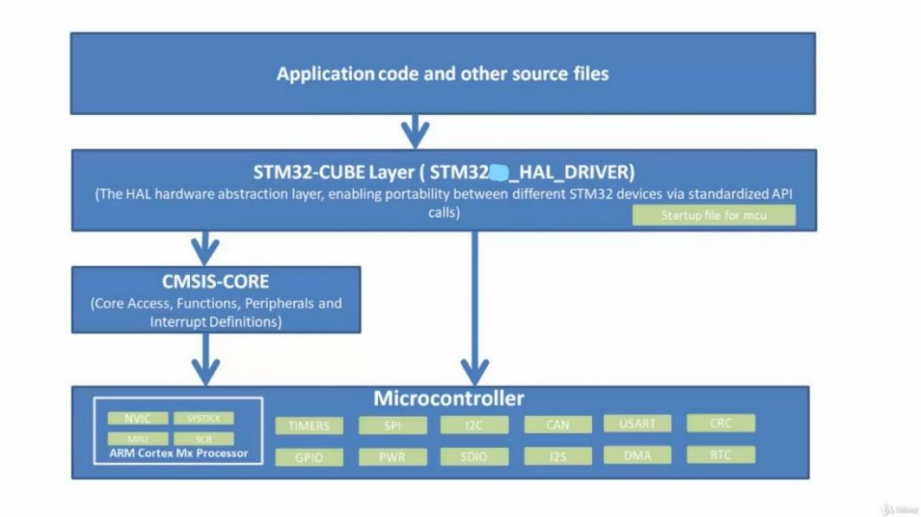


Figure 11- STM32 CUBE Framework

The gist of this diploma work can be easily conveyed using the above Figure[25]. The application code layer contains our source codes which is written to work as gateway. The application layer will call the API's from HAL driver files which enables the application to make use of the microcontroller's peripherals. Then, the HAL layer will also call the API's related to processors using CMSIS's APIs.

7.2 Clock setting

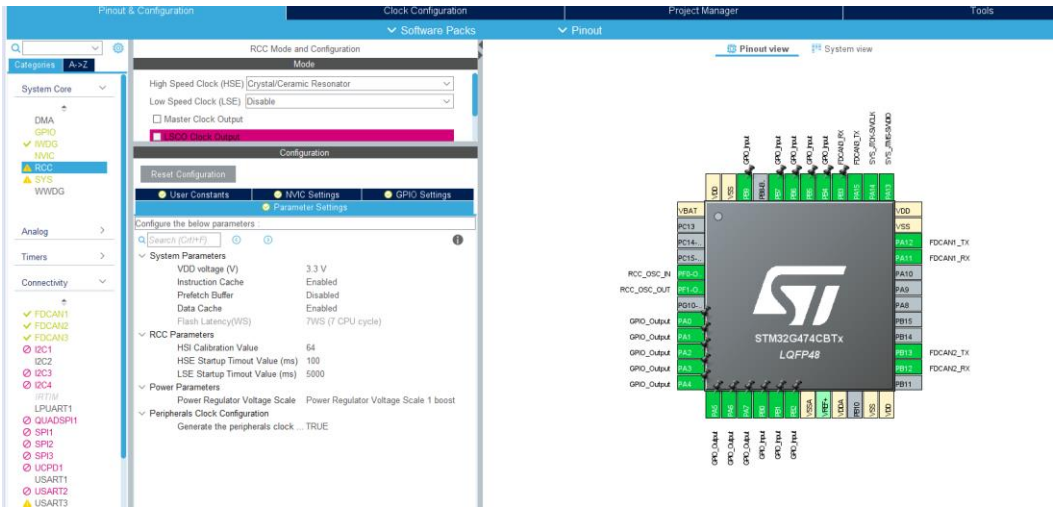


Figure 12- Clock setting

The STM32CubeMx project manager allows configuring the system clock from configuration tab. The internal oscillator in the selected microprocessor is 16 MHz which is not sufficient for our high-speed application. In order to navigate through the Cube Mx software the step by step instructions are provided in the reference manual [12]. We need to switch from internal oscillator to external high frequency RC Oscillator as source for phase lock loop engine (PLL) to further increase the frequency.

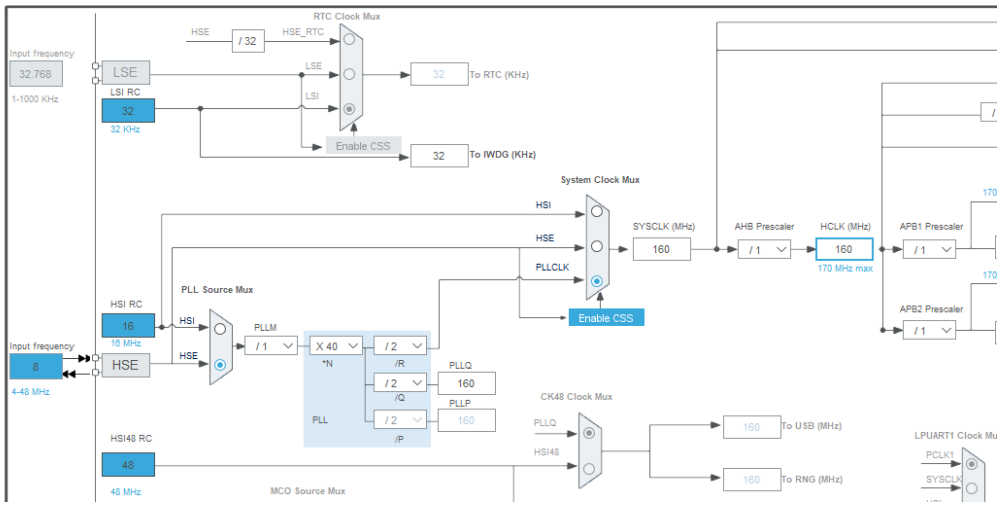


Figure 13- PLL Engine

To change the clock setting to a higher or lower frequency is possible, by modifying the parameters PLL engine. The system clock is derived from the PLL and other peripheral clocks are derived from the system clock. The parameters of PLL engine such as R/Q/M/N are 2/2/1/40 respectively for this application. The processor can handle up to 170MHz. In the RCC tab of system core settings we need to select ceramic oscillator as HSE. Also, make sure LSE (low speed External) is disabled. The Cube MX will automatically modify respective RCC registers during project creation. All the peripheral clocks are supplied with 160 MHz for minor changes of the firmware in future.

7.3 NVIC Interrupt Handler

The Nested Vector Interrupt Controller (NVIC) is one of the internal interrupt peripherals of the STM32 MPU micro processor's interrupt controller. The NVIC and Cortex-M4 processor are closely coupled which enables low latency interrupt processing and efficient processing of late arriving interrupts. All the interrupts including the Cortex-M4 core exceptions are managed by the NVIC. The features of NVIC include 150 makeable interrupt channels, 16

programmable priority levels and implementation of system control registers. For this application the priority levels are set using NVIC tab under system core setting in pin and configuration setting window of STM32CubeMx. All the information regarding NVIC programming is in Cortex – M4 programming manual [15].

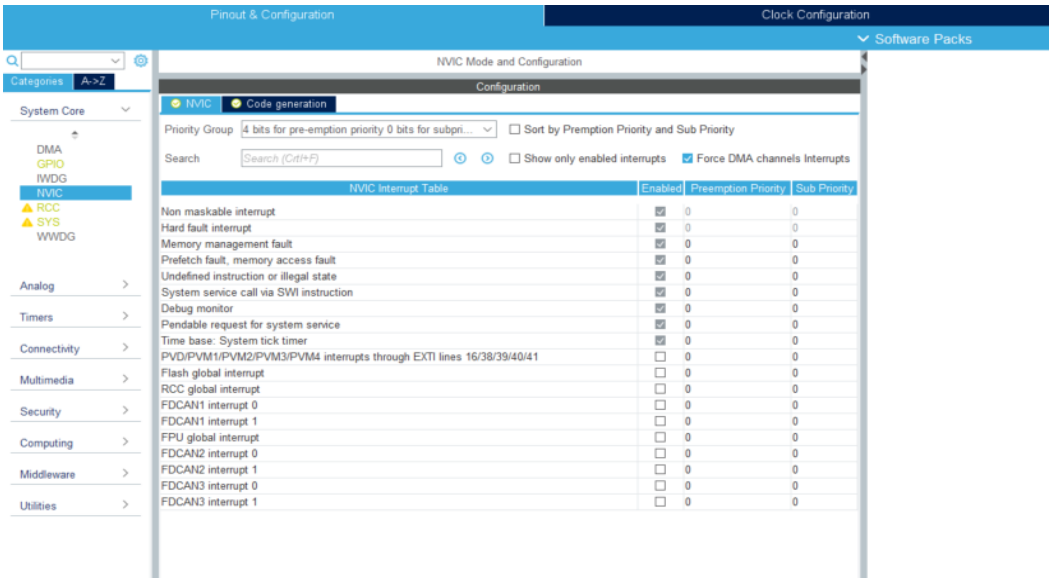


Figure 14- NVIC Configuration

There are nine interrupts enabled in the NVIC interrupt controller table for this application. They are:

- Hard fault interrupt
- Memory management fault
- Prefetch fault, memory access fault
- Undefined instruction or illegal state
- Time base: System tick timer.
- System service call via SWI instruction
- Debug monitor
- Pendable request for system service
- Non Maskable interrupt.

All the interrupt handler APIs and theirscall backs are included in the project during initial code generation by Cube Mx/ STM32CubeIDE. A source file is generated separately which

have function definitions to handle each interrupt, the respective call backs can be called from these functions.

7.4 Peripheral Settings

To get started with programming the microcontroller there two ways, either we can use STM32CubeMx or we can use STM32CubeIDE directly. In this work we use STM32CubeMx for configuration settings and the initialization codes are generated for STM32CubeIDE. Firstly, with help of project manager we can choose microcontroller. Now the STM32CubeMx software will download respective driver files of the microcontroller. We can also choose one of the example projects for the microcontroller.

7.4.1 GPIO

To set the I/O port, you only need to specify a few things. What type of GPIO will it be - analog or digital input or output, use of pull-up or pull-down resistors and data source or receiver - GPIO data register or some alternative function (peripherals). All these settings can be found in the manual [16] together with the procedure how to set the pin correctly. For a signaling LED, its location must be found.

According to the board diagram, the user LEDs are located on pins 8-15 - but these are the housing numbers, so it is necessary to find out the number of the processor pin directly. Pins 8-15 correspond to pins PA0 – PA7 according to the manual [16]. Once it is known where the LED is connected, it is necessary to set the pin as an output, without any pull resistors and use the default value as a data source. It is also necessary to set transfer and receive GPIO pins of FDCAN peripherals which are described in detail in (7.3.4) The following Figure will show screen shot of graphical user interface which helps to configure GPIOs.

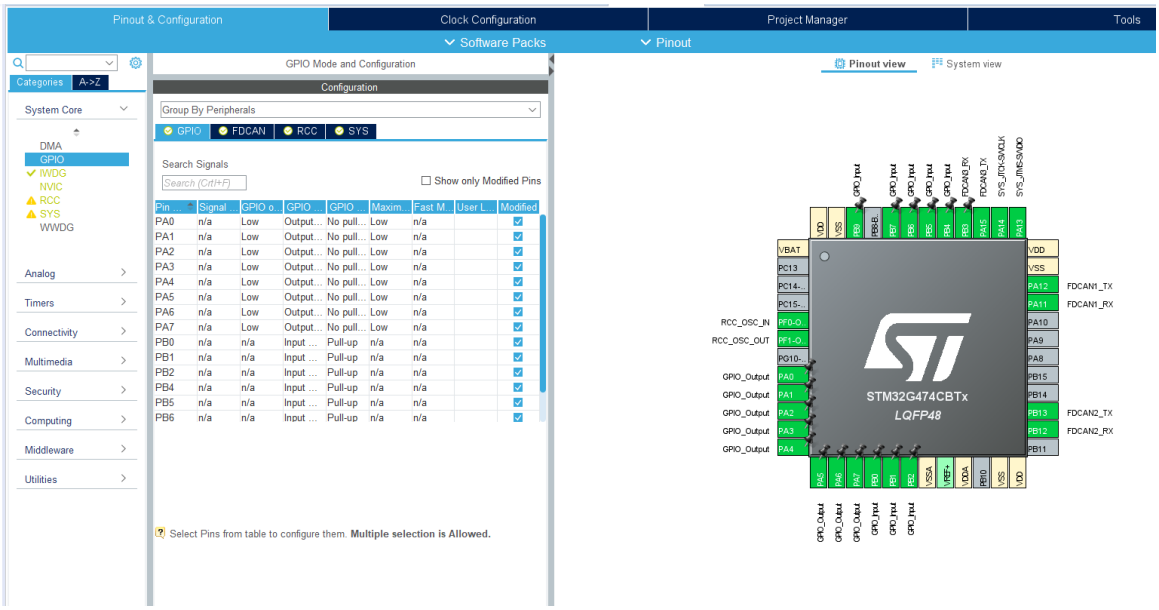


Figure 15- GPIO configuration

7.4.2 CAN Interface:

In order to work with CAN FD peripheral, it is necessary to understand the CAN-FD implementation in STM32 microcontrollers and microprocessors. The FDCAN on STM32 devices are compliant with CAN protocol version 2.0-part A, B and ISO 11898-1: 2015, -4 [22].

FDCAN offers many advantages over the traditional BxCAN (basic c extended CAN), including faster data rates and the extension of the number of data bytes that decreases the frame overhead. The bus load can be also reduced. There is an increase on the number of messages in transmission and reception that requires an improvement of the RAM memory. BxCAN developers can easily migrate to FDCAN given its BxCAN compatibility and as the FDCAN can be implemented without imposing a revision of the entire system design.

The FDCAN contains all BxCAN features in an improved manner and meets the requirements for our application. The various operating modes and RAM management of FDCAN refer to the resource from STM32 [13]. Some of the main features are:

- Improved acceptance filtering
- Two configurable receive FIFOs
- Up to 64 dedicated receive buffers
- Up to 32 dedicated transmit buffers

- Configurable transmit FIFO and transmit queue
- Configurable transmit event FIFO
- Transceiver delay compensation

The selected STM32G4 series MPC have three FDCAN interface. All the 3 interfaces share a common core, Tx handler, Rx handler and message RAM interface. According the reference manual [16] and data sheet from ST for this microcontroller pin numbers are mapped as mentioned in the below table ().

FDCAN interface	RX - pin	Port pin	TX - pin	Port pin
FDCAN1	33	PA11	34	PA11
FDCAN2	26	PB12	27	PB13
FDCAN3	40	PB3	39	PA15

Table 10– Pin Mapping

The FDCAN peripheral is configured using connectivity tab in pin & configuration window of the Cube MX/ STMCube IDE. The corresponding pin outs in the microcontroller is enabled using graphical user interface by just selecting the pins with help of the above table. To start the CAN peripheral, it is necessary to initiate the peripheral related registers and configuring the registers to required operating settings. The Cube Mx software allows us to configure the parameter settings, Mode, GPIO settings of the peripherals.

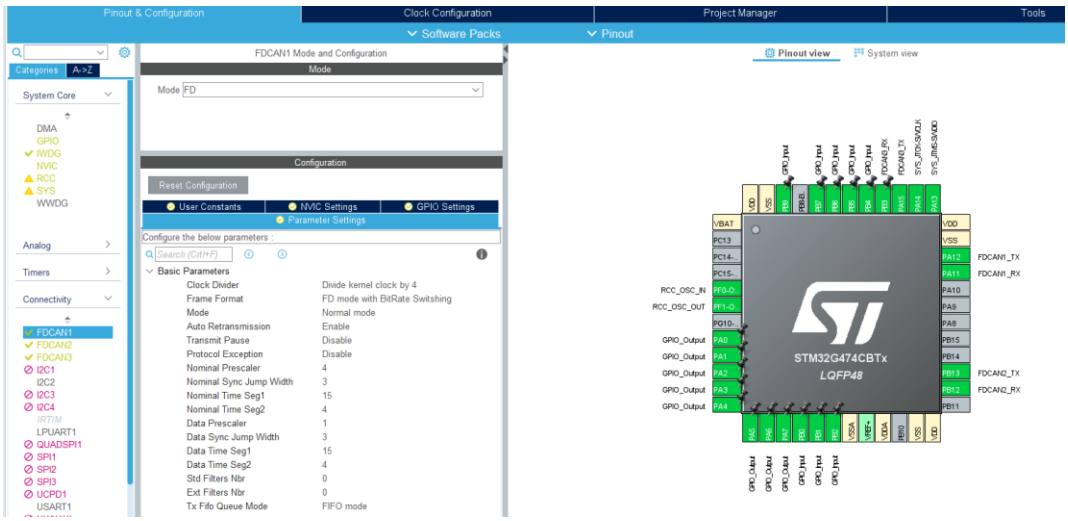


Figure 16- FDCAN Configuration

The input frequency to the FDCAN peripheral from PCLK1 is 160 MHz. The basic parameters of the FDCAN peripheral are calculated carefully according to the application requirements. As a safeguard against programming errors, the configuration of the Bit timing register is only possible while the device is in Standby mode. Configuration settings for both FDCAN1 and FDCAN2 are exactly same. Since FDCAN3 interface is used for controlling the gateway it must be initialized with different parameters. It is necessary to set the internal CAN divider, both Time segments and the synchronization jump correctly. The following table will summarize the settings.

Parameters	FDCAN1/FDCAN2	FDCAN3
Nominal prescalar	4	2
Nominal sync jumpwidth	3	3
Nominal Tseg1	15	15
Nominal Tseg2	4	4
Data prescalar	1	2
Data sync jump width	3	2
Data Tseg1	15	15
Data Tseg2	4	4

Table 11- FDCAN parameter settings

For the given speed requirements - standard speed 500 kBit/s, speed at FD Bit rate switch 2 Mbit/s. So, if the divider is set to 4 for standard speed, the output is 10 MHz, or 100 ns. If the value is divided by 500 kBit /s or 2μs, 20-time quanta are needed to reach the desired speed. Since the Sample point should be around 80%, TSeg1 is set to 15 and TSeg2 to 4. This gives (with a synchronization segment) a total of 20-time quanta. For the FD BRS speed, division 1 is used, 25 ns, so for a speed of 2 Mbit/s, or 500 ns, 20-time quanta must be used again. So, the TSeg1 and TSeg2 settings are the same as for the standard speed. The sync jump is set to 4 at both speeds.

All messages must be received for the FD CAN Gateway application, so this filter has been removed and the Accept All filter has been introduced and saved in the Rx FIFO. Against a

dedicated buffer, the FIFO has the advantage that it is able to store several messages (up to 64 FD CAN messages) at once, against this buffer only one and must be emptied before the next reception (used mainly for storing special messages that do not go as often). FIFO is also much more suitable for the application, because when processing messages in the main loop (without interruption) it takes different processes to process (for example, blocked message vs message with rule for modification and CRC calculation), so messages must be stored so that some are not lost.

7.4.3 Independent Watchdog

A watchdog timer is a specialized timer module that helps a microprocessor to recover from malfunctions. A watchdog timer is based on a counter that counts down from some initial value to zero. If a watchdog timer reaches the end of its counting period, it resets the entire processor system. In order to prevent this, a processor must perform some type of specific action that resets the watchdog. Thus, a watchdog timer can be configured such that it will reach the end of its counting period only if a processor failure has occurred, and by forcing a system reset, the watchdog timer helps the processor to escape from the failure mode and continue normal operation[16].

In STM32 microcontrollers the independent watchdog (IWDG) is a 12-bit down-counter timer clocked by its own dedicated low-speed clock (LSI) and thus stays active even if the main clock fails.

Prescalar	4
Window	4095
Reload	4095

Table 12– Watchdog Parameter settings

7.5 Debugging tools

In order to be able to communicate with the board and load the code into it, it is necessary to have a so-called debugger. Since we use STM32CubeIDE we can use the ST-Link debugger from ST-family.

Figure 17- ST-link Debugger

ST's ST-Link debugger is supported by all popular IDEs such as IAR, Keil, Eclipse, and more. Works under all used OS - Windows, Linux, Mac. Includes JTAG and SWD interfaces for connection to a microcontroller. ST-LINK is a USB device and has to be connected to a PC host. It can be either embedded on ST boards or provided as standalone dongle. One Such dongle is provided for this work [10]. Unfortunately, the ST-link debugger didn't come with SWD 4-pin wire and our hardware does not have the same pinout as the large classic 20-pin connector, so a reduction is needed. We had to look for the exact pins in 20-pin output from ST's debugger to SWD pins in the hardware.

8. Structure of software implementation

After configuration of GPIOs, Timers, system clock, interrupt timers and FDCAN peripherals in the STM32CubeMx the project is created using project manager for suitable coding platform. For this application the initial codes are generated for STM32CubeIDE platform. The project workspace and STM32 repositories are selected according to user requirements. The code generator will include all the necessary CMSIS files, startup files and also includes HAL driver files. Once project has been created it is opened using STM32Cube IDE in a workspace. Many projects can be created in a workspace it is better to get familiar with the IDE from the resources provided by STM [11].

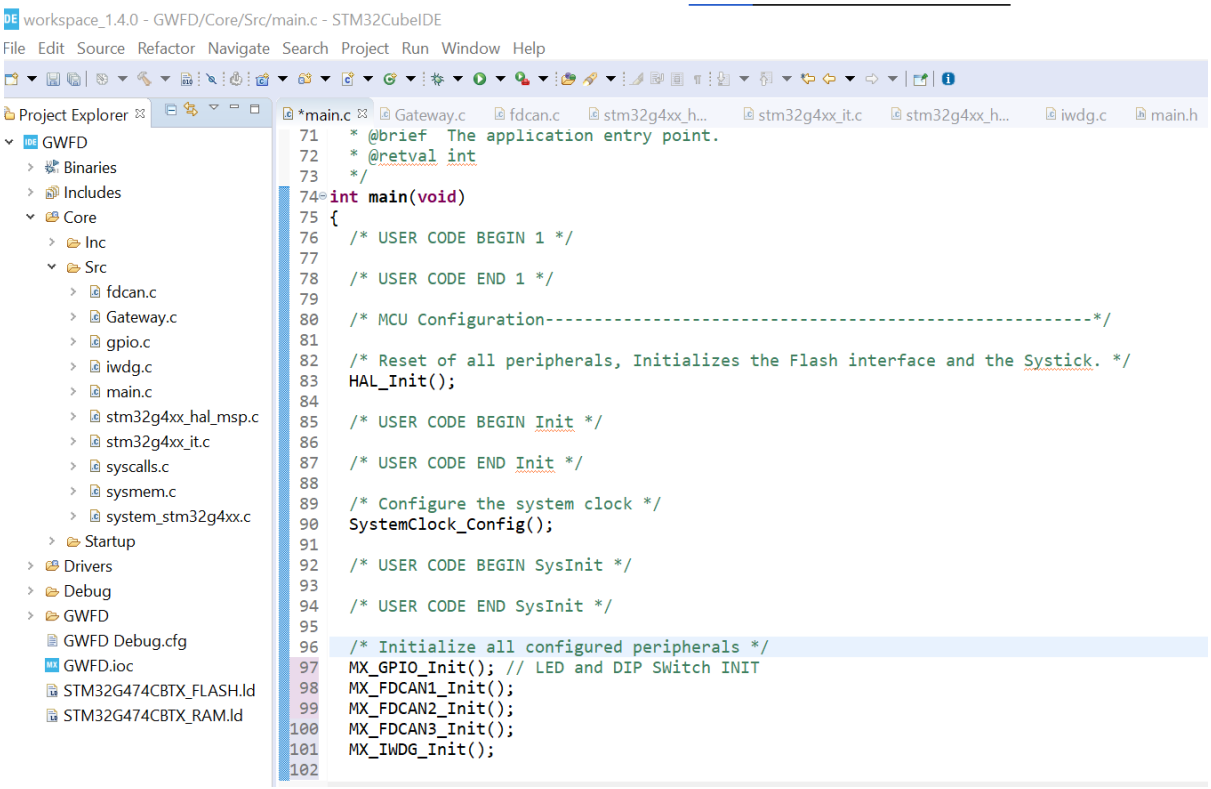


Figure 18– Project structure

So, before you start working on STM32 based board using STM32 cube framework, you should understand the program flow of the project. We cannot start working with peripheral directly by calling its associated driver APIs as soon as you reach main function of the project. Generally, there are some mandatory and optional MCU initializations must be done. They are:

- Flash controller Initializations
- Floating point unit initialization (if supported)
- Setting up stack (mandatory)
- System clock settings (optional), in this project it is taken care by STM32Cube MX
- Flash wait state settings when system clock is more(mandatory)
- Systick Timer Initializations to trigger interrupt for every 1ms (required when STM32 HAL APIs are used)

All these initializations are taken care by STM32CubeMx project creator it is necessary to understand the program flow to create application layer program. There are three important source codes generated namely, main.c, stm32g4xx_msp.c and stm32g4xx_it.c

along with peripheral specific source code file. Generally peripheral initialization is done in two steps and the following figure shows the program flow[25]:

- High level initialization (this code goes into main.c or peripheral source code file) such as clock divider and other peripheral settings (see 7.)
- Low level Initialization (this code goes to msp.c) such as enabling FDCAN IRQ in NVIC and conFigureuring GPIO pins to behave as FDCAN Rx and Tx pin (see 7.)

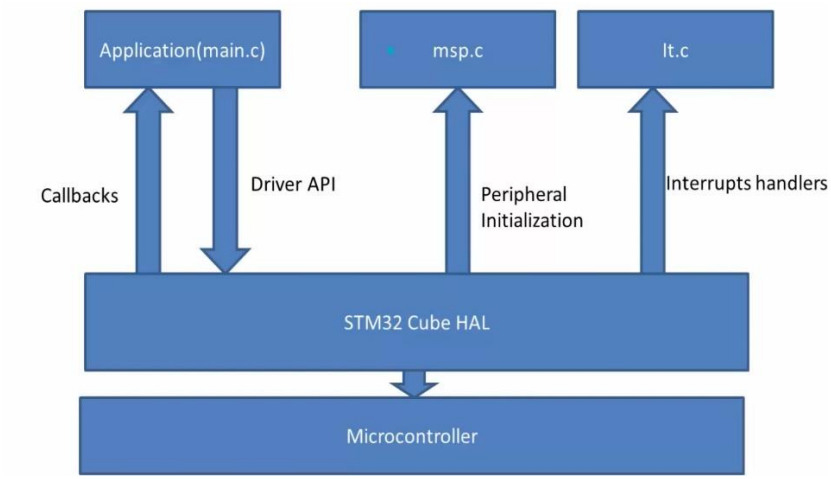


Figure 19– STM32 Program flow

The main logic of application goes here in the main.c. And in the main.c, if we want to talk to the micro-controller specific peripherals then we will use driver APIs like HAL_FDCAN_transmit, HAL_FDCAN_recieve or timer start, timer stop etc... So, those are exposed by the respective driver file of the peripheral which is present in the STM32 HAL layer. Let's say we want to communicate with the CANFD peripheral of the micro-controller. Now, after all the initialization in the main.c, it calls the FDCAN_init() which is actually a driver API provided by the FDCAN driver, Stm32g4xx_hal_fdcan.c which is present in the STM32 Cube layer. Once application calls FDCAN_init() that is the high level initialization. The cube layer calls back FDCAN_Msp_int() function, which must be implemented in msp.c in order to accomplish the low level initialization. So, that is step number two. So, collectively step number one and step number two completes the initialization of a peripheral. So, after that application can go ahead and transmit the data.

Let's say, application uses FDCAN_transmit_it() that is FDCAN transmit with interrupt in order to transmit some data over the CAN peripheral. So, HAL_FDCAN_transmit_it will be

implemented in Stm32g4xx_hal_FDCAN.c and that function or that API will take care of sending data over the FDCAN to the external world. Now, once the data is transmitted successfully then what the peripheral does is, it issues interrupts to the processor Hence your interrupt handler which is written in the stm32g4xx_It.c will run.

As soon as you enter the main function of the project which is implemented in main.c, the first function to be executed from main function has to be HAL_Init(). So, the HAL_Init() function is actually used in order to initialize the device HAL layer or cube HAL layer of STM32 cube framework. The HAL_Init() function does three things it initializes the flash interface unit, which is actually provided by the Stm32f4xx_hal_conf.h means for every device family there will be conf.h which is provided by the ST's Cube layer and that conf.h will have all the configuration required in order to properly initialize a micro-controller. Following figure shows the flow of the API[25].

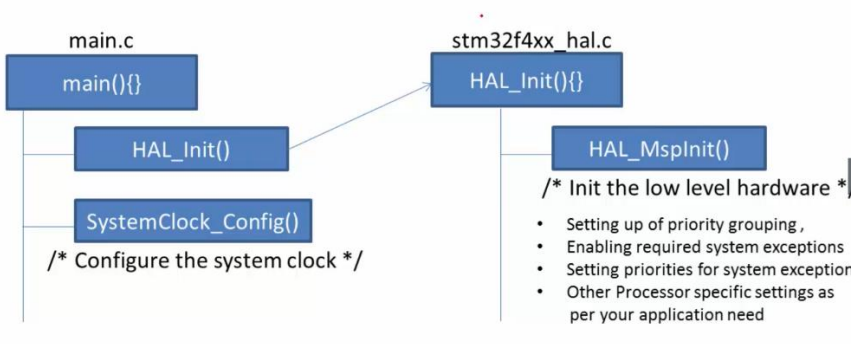


Figure 20– HAL_init() flow

The second thing the HAL_Init() does is very important that is SysTick timer initialization. So, basically it initializes the SysTick peripheral of the processor to generate interrupt for one millisecond. So, that means a background clock will be always it will be ticking in Cube Mx generated code, it is like a heartbeat. For every 1ms SysTick timer will be generating interrupts. So, this is actually a requirement for Cube HAL APIs to work properly, like the APIs to transmit data over FDCAN or APIs to transmit data over SPI, I2C so almost all data transfer and other APIs, they actually depend upon this SysTick timer. That's the reason why HAL_Init() actually configures the SysTick timer to generate interrupt for every 1ms.

The third important thing what HAL_Init() does is, it actually calls HAL_MspInit() in order to perform other Processor Specific Low Level Initialization. So, this is to deal with some of the

processor level details and this is application specific. So, that's why HAL_Init() actually calls HAL_MspInit(), MSP stands for Micro-Controller Support Package.

File or Folder	Description
Main.c	The main program loop contains APIs for initializing HAL, system clock configuration, peripherals and then call main application
Msp.c	Microcontroller support package source file where low-level initialization such as GPIO and NVIC
It.c	Contains the interrupt handler function call backs of systick timer, NVIC handler APIs
Fdcan.c	FDCAN initialization and contains HAL call backs to transmit, receive and other peripheral functions
Gpio.c	Initialization of GPIO pins for LED and DIP switch
Iwdg.c	Initialization of Individual watchdog contain function definition MX_IWDG_Init()
Gateway.c	The main part of the application, message processing and rules.

Table 13– Source files summary

9. Testing

For complete testing, it is possible to generate all combinations of input parameters, in this case all rule parameters. However, such testing is very demanding and overall, there are many combinations, it takes up memory, testing can be time consuming before all combinations are performed, and finally manual checking of each test case is very difficult for a person not to miss any mistake. Since all the functionality has been implemented and verified in the previous versions it is enough consider few test cases for our work.

9.1 Testing Methods

Even though all the functionality has been implemented and verified in previous versions it is necessary to understand the testing techniques which were done to benchmark the ability of the software [21]. There are many testing techniques followed by embedded community and the most demanding but most covering testing, using all the combinations is called MCC - Multiple Condition Coverage. Another technique is called MC/DC - Modified Condition/Decision Coverage respectively, here all combinations that affect the result of the decision expression are tested, means each of the results of all conditions in the logical expression is tested at least once and at the same time each of the conditions independently affects the resulting logical expression. Another technique is Pairwise testing, all combinations are covered in each pair of inputs. Other techniques such as C /DC, CC, DC have only a small test coverage, so they are used for non-critical parts of the application - they work on the principle of meeting/not fulfilling the whole condition (DC - Decision Coverage) or part of the logical expression (CC - Condition Coverage) [21].

The Pairwise testing method was chosen for testing the application, because it is not so demanding like MCC, but at the same time it will cover enough cases to test the entire application. For Pairwise testing, it is first necessary to determine the so-called equivalence classes - the system should behave in the same way for all values from an equivalence class according to the specification (based on which it is tested). Depending on the type of input, it can be defined by interval such as amount, age or by discrete values such as browser type, payment method, menu item [21].

Messages and commands are input for the CAN FD Gateway application, so it is possible to specify equivalence classes. The general messages depend on the identifier, which can be standard, extended, and from the application's point of view, it can run the application by any rule. Parameters such as data and length are one equivalence class, there is no difference whether 8 bytes will be sent, or 64, as well as no difference in what data is sent. In terms of commands - rules, the impact is different. The Trig command has no parameters, but it needs to be tested, the Reset command has a single parameter and that is the type of reset. More interesting commands to test are Block/Pass and Modify. Both have the parameters like identifier, type (unlimited, time-limited, limited by the number of frames)

and duration (zero length, any 16-bit number except 0). For an identifier, equivalence classes can be specified as standard, extended, or already included in a rule, because each class will test a different part of the application. Similarly, a type with three classes always has a different impact on the application. Finally, the Modify command has even more parameters that should be tested. Data and mask are again parameters that do not matter what they will be, resp. their impact is always the same, ie they can be summarized into one equivalence class. In contrast, the DLC and CRC parameters again have a different impact on the application depending on the value of the parameter. DLC could be classified as follows - shortening, non-shortening, lengthening the length of the message. CRC - do not count, correctly or incorrectly calculate. In total, there are only 18 options for the Block/Pass command (which is still well done manually), but the Modify command has 162 combinations [21].

To test the final application, Kvaser Memorator Pro was provided - a logging tool, but also as an active unit on the CAN bus with support for filtering, error detection and generation. It has two CAN FD interfaces (communication speed in the range of 50-1000 kbps) galvanically separated from the board itself and uses USB to connect to a PC. In terms of software, the CAN King programm and the SDK (Software Development Kit) are available for free download on the manufacturer's website for easy implementation of this tool within the application.

Figure 21-Kvaser Memorator Pro

The CAN King program is used to easily monitor the connected CAN network, but the unit can also actively participate in communication and acknowledge received frames and, also send user-defined frames. For initial testing, the CAN King program was used, when both CAN interfaces were connected using a terminated jumper (for physical communication in the CAN network, a 9-pin connector is used and there must also be a CAN_H, CAN_L 120Ω resistor between the wires for termination, otherwise there would be reflections on the line). The connection created the so-called Loopback - when one interface sends a frame, the other receives it and vice versa. In Loopback mode, both interfaces within the network must also be active, because CAN requires each frame to be acknowledged by someone. After connecting and pressing the Start Run button to activate both interfaces with the given settings, it is then very easy to generate a test report. After verifying that the tool works, the next step was to send and receive messages using the SDK, so that in the end it is possible to test the real application.

9.3 Test conclusions.

Firstly, it is necessary to test all the three FDCAN peripherals. So random messages were generated and sent to each FDCAN interfaces using SDK and CAN king. By doing this we can also test whether right LEDs are switching ON and OFF according to timer value set and verify the functionality of systick timer interrupts. Each interface has a pair of LEDs, yellow to indicate any message received and red LED to indicate any error. From the testing methods as discussed above also from previous versions of gateway there are some test cases for block/pass and modify commands[21].

Cases	ID	Type	Length
1	standard	Without rule	0
2	standard	Timewise	Any num
3	extended	Without rule	Any num
4	extended	Timewise	0
5	existing	Number of	0
6	existing	Without rule	Any num
7	standard	Number of	Any num
8	extended	Number of	~0
9	existing	Timewise	~0

Table 14 –Test cases Block/Pass commands

Cases	ID	Type	Duration	DLC	CRC
1	standard	Without rule	0	Shortening	Don’t count
2	standard	timewise	Any num	Extending	Calculate correctly
3	Extended	Without rule	0	Extending	miscalculate
4	Extended	timewise	Any num	Shortening	Don’t count
5	Existing	Without rule	Any num	Same length	Calculate correctly
6	existing	timewise	0	Same length	miscalculate
7	standard	Number of	Any num	Same length	miscalculate
8	Extended	Number of	0	Shortening	Calculate correctly
9	Existing	Number of	~0	Extending	Don’t count
10	Extended	Without rule	~Any num	Same length	Don’t count
11	existing	timewise	~Any num	Shortening	miscalculate

Table 15 – Test cases modify command

All the above test cases were generated for testing and to verify the previous gateway module which is using CANFD protocol, it is enough to test any one or few cases from each table to make sure the functionality of this gateway module. Firstly, the reset command is checked, then followed by block/pass commands here, a rule is set to block a message with extended ID or standard ID then using SDK the messages are sent to the module. Similarly pass command is also checked. Then for modify command a message with DLC of 64B is sent to the gateway.

The gateway is set to modify this message to 8B for only 3 messages of this type. This ensures the functionality of the gateway for modify command. The same test case is also done for extended ID. Since trig command is used to set the gateway according to the rules set by modify and block/pass commands its functionality is also verified simultaneously. Read command is checked by sending can messages to both FDCAN1 and FDCAN2 one after the other using SDK.

Read command enables message mirroring of control FDCAN or FDCAN3 we can read the messages sent to FDCANs in the output window of the CAN King software. The message with statistical data is not transmitted, the transmission must be activated by the Stat command. if the sending of statistics has been activated the module regularly sends a frame every 100 ms with the number of messages forwarded in both directions.

During testing there were some strange behavior of the gateway due to some undeclared parameters in FDCANs initialization structure. Even though the importance of this parameter were least important it is necessary to define some value and should be declared in the structure of FDCAN handle. Therefore, above results verify the functionality of the software module implemented in the developed hardware for this diploma work.

10. Conclusion

The proposed application uses a defined Gateway management protocol. 3 networks are defined - 2 between which frames are forwarded and the third, which controls the rules for processing (the rules are of the type blocking, transmission, message modification).

The resulting application on real hardware which has been specifically designed for this work to operate in lower voltage and it has also passed all tests and in the future, it is expected that it will be used in the CAN FD network for which the solution is designed. Thus, all points of the assignment are met and the application, in laboratory conditions, seems to be fully functional.

References

[1]. Wikipedia. CAN bus. 2020. URL: https://en.wikipedia.org/wiki/CAN_bus (cit. 23.12.2020).

[2]. Wikipedia. Vehicle bus. 2020. URL: https://en.wikipedia.org/wiki/Vehicle_bus (cit. 23.12.2020).

[3]. Reutlingen Robert Bosch GmbH. *CAN FD - CAN with Flexible Data-rate*. 2012. https://www.cancia.org/fileadmin/resources/documents/proceedings/2012_hartwich.pdf

[4]. Wikipedia. Carrier-sense multiple access. URL: https://en.wikipedia.org/wiki/Carrier-sense_multiple_access (23.4.2020)

[5]. Rohde_Schwarz.com. URL: https://www.rohde-schwarz.com/us/solutions/test-and-measurement/automotive/in-vehicle-networks-and-ecu-testing/overview/in-vehicle-networks-and-ecu-testing-overview_231834.html

[6]. Arm Community. URL: https://community.arm.com/developer/ip-products/system/b/embedded-blog/posts/10-steps-to-selecting-a-microcontroller (12.01.2014)

[7]. ST Microcontrollers.URL:https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

[8]. ST Microcontrollers. URL: https://www.st.com/resource/en/datasheet/stm32g474cb.pdf

[9]. Texas Instruments. URL: https://www.ti.com/lit/ds/symlink/tps62172.pdf?ts=1608959229349

[10]. Texas Instruments: URL: https://www.ti.com/document-viewer/TCAN337G/datasheet/device-options-sllseq75783#SLLSEQ75783

[11]. ST Electronics. URL: https://www.st.com/resource/en/user_manual/dm00629856-stm32cubeide-user-guide-stmicroelectronics.pdf

[12]. ST Resources. URL: https://www.st.com/resource/en/user_manual/dm00104712-stm32cubemx-for-stm32-configuration-and-initialization-c-code-generation-stmicroelectronics.pdf

[13]. ST Resources. URL: https://www.st.com/resource/en/user_manual/dm00610707-description-of-stm32g4-hal-and-lowlayer-drivers--stmicroelectronics.pdf

[14]. Science Direct. URL: https://www.sciencedirect.com/topics/computer-science/hardware-abstraction-layer#:~:text=In%20computers%2C%20a%20hardware%20abstraction,or%20from%20a%20device%20driver

[15]. ST Resources. URL: https://www.st.com/resource/en/programming_manual/dm00046982-stm32-cortex-m4-mcus-and-mpus-programming-manual-stmicroelectronics.pdf

[16]. ST Resources. URL: https://www.st.com/resource/en/reference_manual/dm00355726-stm32g4-series-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

[17]. Cypress Traveo. URL: https://www.cypress.com/file/354851/download

[18]. ST Resources. URL: STM32G4 Series of mixed-signal MCUs with DSP and FPU instructions - STMicroelectronics

[19]. Reutlingen Robert Bosch GmbH. CAN FD - CAN with Flexible Data-rate. 2012. URL: https://www.can-cia.org/fileadmin/resources/documents/proceedings/2012_hartwich.pdf

[20]. J. Novák. Lecture KRP – Automotive CAN 2019. URL: https://moodle.fel.cvut.cz/pluginfile.php/216202/mod_resource/content/2/Automotive%20CAN%20English.pdf

[21]. CAN Gateway. URL: https://dspace.cvut.cz/handle/10467/76275

[22]. Text book. Control Area Network by Konard Etschberger. ISBN: 978-3000073762

[23]. Copper Hill technologies, URL: https://copperhilltech.com/blog/controller-area-network-can-bus-bus-arbitration/#:~:text=Not%20only%20is%20the%20CAN,the%20bus%20is%20available%20again.

[24]. CSS Electronics. URL: https://www.csselectronics.com/screen/page/can-fd-flexible-data-rate-intro

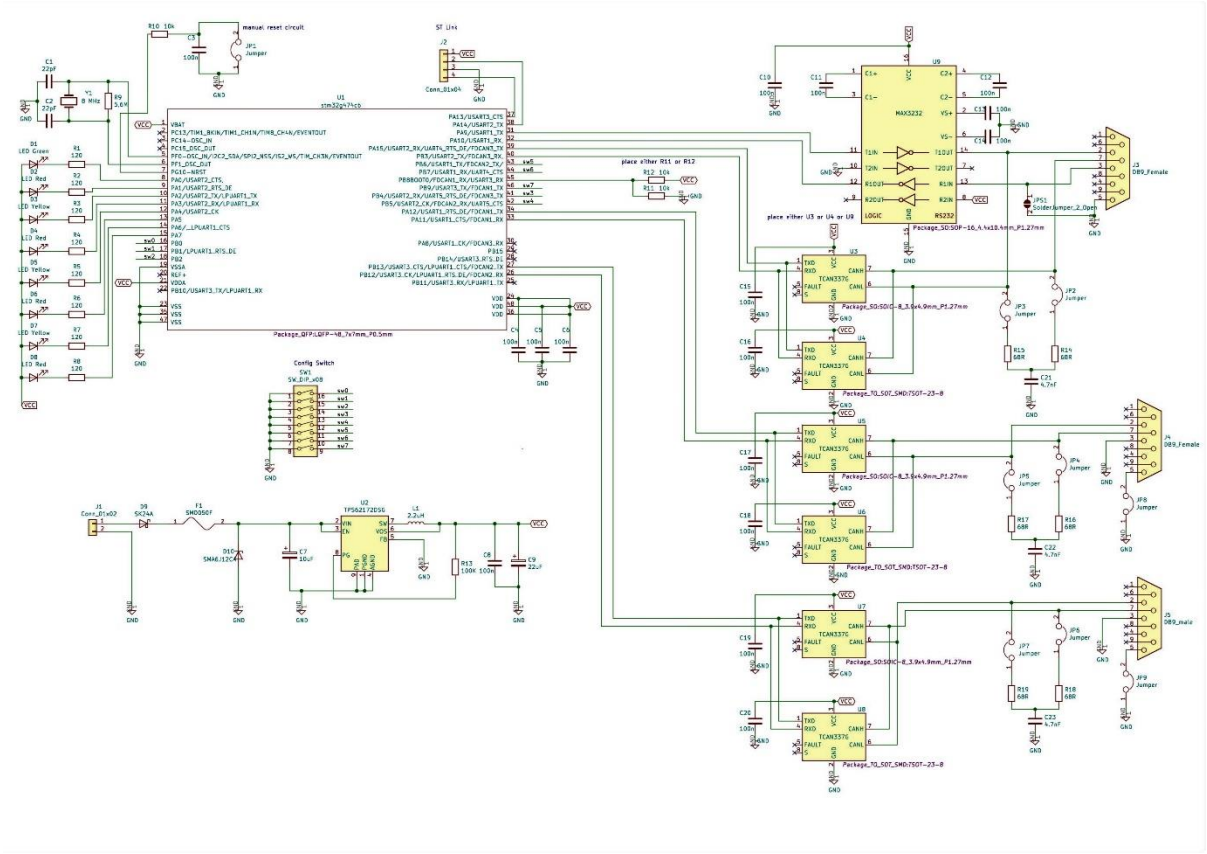
[25]. Udemy Online Course. URL: https://www.udemy.com/course/microcontroller-programming-stm32-timers-pwm-can-bus-protocol/learn/lecture/11647222?start=225#content

[26]. STM32 Cube layer. URL: http://www.emcu.it/STM32Cube/STM32Cube.html

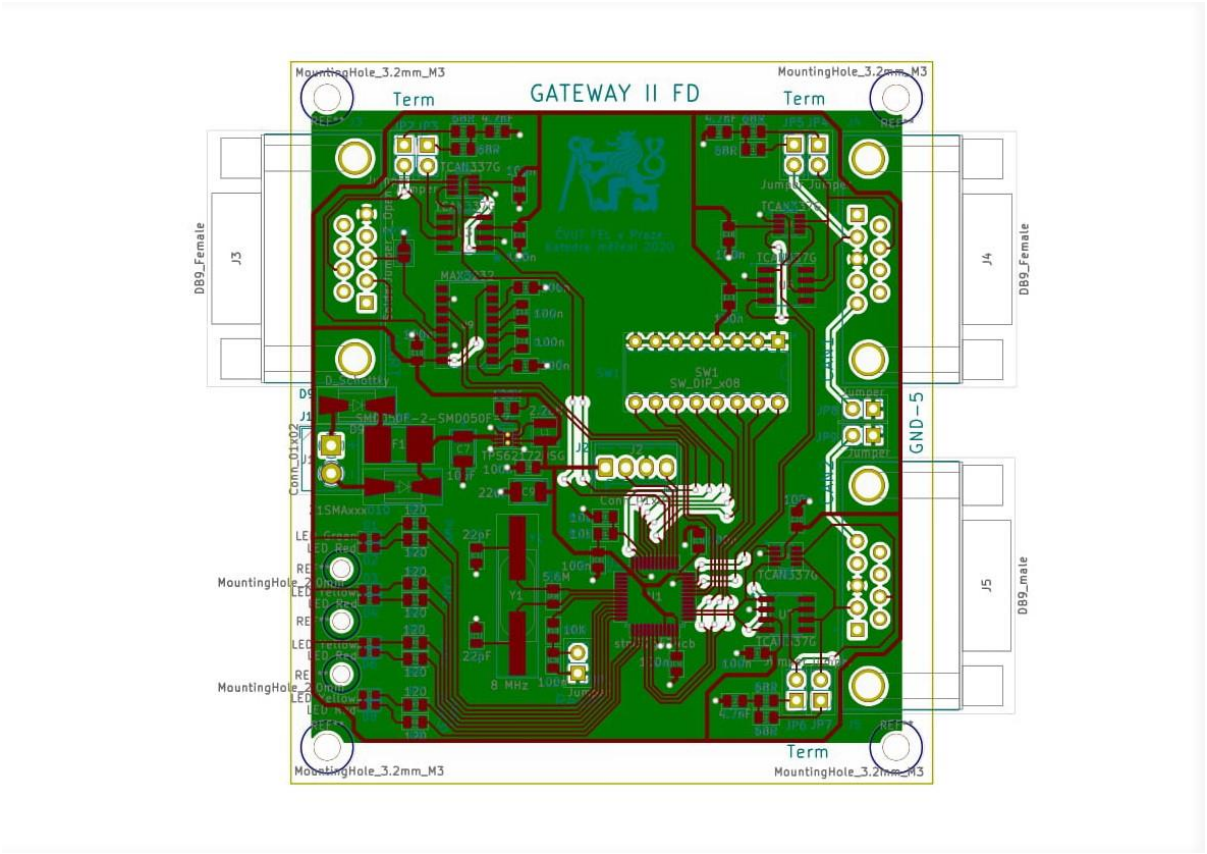
[27]. CAN & CANFD online resource. URL: https://elearning.vector.com/mod/page/view.php?id=363

Appendix

A. Schematic



B. PCB layout


GATEWAY II FD
Term
Term
Term
Term
MountingHole_3.2mm_M3
MountingHole_3.2mm_M3
MountingHole_3.2mm_M3
MountingHole_3.2mm_M3
DB9_Female
DB9_Female
DB9_male
J3
J4
J5
GND-5
SW1
SW_DIP_x08
TCAN337G
8 MHz