POLITECNICO DI TORINO

Master's Degree in Mechatronics Engineering

Master's Degree Thesis

Test and Development of the CAN peripheral in SPC58NN84x 32-bit Power Architecture microcontroller

Supervisors

Prof. Paolo BERNARDI

Candidate

Zeyad MOHAMEDIN

July 2025

Summary

In the ever-evolving landscape of embedded systems, the SPC58NN84x microcontroller stands as a notable advancement. This microcontroller, a successor to the SPC5x family, offers enhanced features and increased throughput while maintaining a favorable cost-to-feature ratio and notable improvements in power efficiency and performance. With three processor cores, two checkers, and an embedded e200z0 core within the Hardware Security Module, the SPC58NN84x finds application in a wide range of contexts, including automotive powertrain controllers, integrated chassis control, and safety-critical systems.

This thesis addresses a pivotal step in harnessing the potential of the SPC58NN84x microcontroller by integrating it with the Micrium operating system. The process involves fundamental module implementations within the microcontroller, culminating in the creation of an abstraction layer and APIs for the operating system. The study's primary focus is on the CAN (Controller Area Network) module, encompassing hardware and software configurations. Specific components include CAN and UART hardware setups, software configurations for UART and CAN modules, and the development of a bare-metal application. This application facilitates CAN communication tests, control through UART utilizing the Direct Memory Access (DMA) algorithm, and CAN operating mode manipulation via UART commands. Additionally, essential Micrium OS APIs are implemented, forming the bedrock of hardware abstraction.

The conclusion underscores the centrality of embedded systems in contemporary technological progress while acknowledging the escalating complexity of programming them. Operating systems offer an effective remedy by elevating the abstraction level, thereby simplifying embedded system programming. This thesis serves as a pivotal phase within a larger project centered around comprehensive testing of the SPC58NN84C microcontroller and its integration with the Micrium operating system. The study encompasses microcontroller setup, UART-controlled tests, CAN peripheral communication tests, bare-metal application coding, result verification through various tools, and the implementation of fundamental operating system functions.

Detailing the intricate connectivity arrangements, including necessary adaptors

and motherboards, the thesis offers both graphical and textual elucidations. The CAN test comprises a bare-metal application incorporating primary functions such as initialization, message reading, transmission, and termination, alongside auxiliary functions for MCU configuration, UART management, and debugging enhancement. The Micrium OS abstraction layer parallels these functions with distinct configurations and arguments, supplemented by an additional capability to manage transmitted and received CAN messages.

The thesis examines CAN communication modes across various operational scenarios, facilitated by UART input signals. Comprehensive results are derived through hardware observation, debugging tools, and logic analyzers. This study lays a foundation for realizing Micrium OS-controlled CAN peripherals within the SPC58NN84C MCU, signifying a promising trajectory for future advancements. As the groundwork is established, subsequent developments can transform the SPC58NN84C into an accessible, efficiently-operated Micrium OS-controlled MCU, ushering in new dimensions of programmability and efficiency for programmers and developers.

Acknowledgements

I would like to express my sincere gratitude to all those who have contributed to the successful completion of this thesis.

Foremost, I extend my heartfelt appreciation to Professor Paolo Bernardi for his invaluable guidance, unwavering support, and continuous supervision throughout the course of this research. His expertise, insightful feedback, and dedication to my academic growth have been instrumental in shaping this work.

I am also thankful to the faculty members and staff of the Department of Control and Computer Engineering (DAUIN) and Department of electronics and Telecommunication (DET) at Politecnico di Torino, whose knowledge-sharing environment provided me with a strong foundation to undertake this research project.

Finally, I extend my thanks to all the individuals who participated in this study and provided their insights, which significantly contributed to the results presented in this thesis.

In conclusion, this research would not have been possible without the combined efforts and support of everyone mentioned above. Thank you for your contributions to my academic journey.

Table of Contents

List of Acronyms

List of Acronyms

Acronym	Full Definition
CAN	Controller Area Network
UART	Universal Asynchronous Receiver-Transmitter
OS	Operating System
MCU	Microcontroller Unit
USB	Universal Serial Bus
IDE	Integrated Development Environment
ECU	Electronic Control Unit
GPIO	General-Purpose Input/Output
RTOS	Real-Time Operating System
LIN	Local Interconnect Network
RAM	Random Access Memory
CPU	Central Processing Unit
FIFO	First In, First Out
DMA	Direct Memory Access
I/O	Input/Output
LED	Light Emitting Diode
API	Application Programming Interface

List of Tables

List of Figures

Chapter 1

Introduction

1.0.1 What is SPC58NN84x

The SPC58NN84x microcontroller belongs to a family of devices superseding the SPC5x family. SPC58NN84x builds on the legacy of the SPC5x family while introducing new features coupled with higher throughput to provide a substantial reduction of cost per feature and significant power and performance improvement (MIPS per mW). On the SPC58NN84x device, there are 3 processor cores, 2 checkers, and one e200z0 core embedded in the Hardware Security Module. SPC58NN84x applications include

- Automotive powertrain controllers for six to eight-cylinder gasoline, diesel engines and advanced combustion systems as well as high-end hybrid and transmission control
- integrated chassis control, high-end steering, and braking
- in general any safety-critical application requiring a very high level of safety integrity

1.0.2 What is the development done in this study

This study could be considered one of the first steps to operate the SPC58NN84x microcontroller with Micrium operating system. which is a complex process that includes making the basic Implementation of many modules inside the microcontroller and then using these basic implementations to build the abstraction layer and the APIs of the operating system. This study considers mainly the CAN module, including:

- CAN hardware setup

- UART hardware setup
- UART software configuration
- CAN modules software configuration
- Building a bare-metal application which
  - testing the transfer and receiving of the CAN
  - controlling the CAN through UART which operates in the runtime using the Direct Memory Access (DMA) algorithm
  - controlling the CAN different operating mode using the UART
- the basic implementation of the used APIs by Micrium OS, which composes the hardware abstraction

Chapter 2

Background

2.0.1 Controller Area Network (CAN)

The CAN is an ISO that defines a serial communication bus as a multi-master, message-broadcasting system replacing the complex wiring communication with a two-wire bus. Can communication provides high transmitting rates of up to 1 Mbps for standard CAN communication Unlike typical networks such as USB or Ethernet, CAN does not transmit big blocks of data between two nodes under the direction of a central bus master. In a CAN network, many short messages, such as temperature or number of revolutions per minute, are broadcasted to the whole network, ensuring data consistency in every node in the system The CAN bus protocol is becoming increasingly popular among engineers who design advanced industrial embedded systems and that's because:

- The Decentralization: each node on the CAN bus has complete access to the CAN bus, unlike other communications which are based on assigning the responsibility to just one node, consequently, if this bus node fails all the communication will fail, that's the reason why CAN communication is a perfect communication protocol for safety-critical applications.
- Event-driven: The transmission of CAN messages on the bus is not prescheduled or depends on the time, otherwise, the communication channel is only busy if new data has to be transmitted, allowing for very rapid bus access and making it able to properly handle the asynchronous events.
- Receiver-selective addressing: A CAN network uses a method of receiver-selective addressing to prevent dependencies between bus nodes and maximize configuration flexibility. Every CAN message is available for receipt by every CAN node (broadcasting). A condition is that each CAN communication must be identifiable by message identification (ID) and node-specific filtering.

Despite the increase in costs, this permits the integration of more CAN nodes without modifying the CAN network.

CAN Frames

The data transmitted on the bus are encapsulated in a frame comprised of multiple bits. One or more bits of this frame are grouped to identify one of the frame's characteristics (e.g., the identifier and the length of data being transmitted). The CAN network relies on content-related addressing for communication. Identifiers are not assigned to CAN nodes, but rather to data and remote frames. So, any CAN node can receive all the broadcasted CAN messages. Each receiver is responsible for choosing CAN messages individually. Such flexible receiver-selective addressing needs each receiver to filter the received CAN messages by filtering the identifier of each message.

The CAN frames are classified into three types:

- Data Frame: This is a data frame with a maximum payload size of eight bytes. For this purpose, there is the so-called data field, which is surrounded by numerous other fields necessary to execute the CAN communication protocol.
- Remote frame: a frame type that can be used to request user data, i.e. data frames, from any other CAN node. A remote frame has the same structure as a data frame, except for the missing data field.
- Error frame: The error frame can be used to indicate errors discovered during communication. ongoing erroneous data transmission is stopped, and an error frame is sent. The structure of an error frame is completely different from that of a terminated erroneous data or remote frame. It is made up of only two components: the error flag and the error delimiter.

Since error frames are out of the scope of this thesis, the following paragraphs will contain a description of the data and remote CAN frame types, their bitfield structure, significance, and meaning. CAN Data frames are classified into two types:

- Can frame of the standard CAN
- Can frame of the extended CAN

Standard CAN Frame

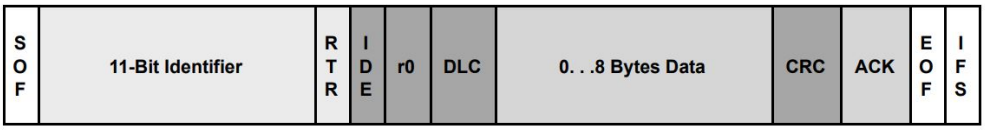

Figure 2.1: Standard CAN frame Structure [1]

Bit field	Length	Explanation
SOF	1	The single dominant start of frame (SOF) bit indicates the beginning of a message and is used to synchronize nodes on a bus after they have been idle
11-Bit Identifier	11	The identification (ID) comes after the SOF. This determines the priority of the data frame (the lower the binary value, the higher the priority), and along with acceptance filtering, it establishes sender-receiver relationships in the CAN network which are stated in the communication matrix.
RTR	1	"Remote transmission request", is used by the sender to inform recipients of the frame's type (a data frame or a remote frame). A data frame is indicated by a dominating RTR bit.
IDE	1	A dominating single identification extension (IDE) bit indicates that the CAN identifier that has been transmitted is a standard CAN identifier
r0	1	Reserved bit (for possible uses in the future)
DLC	4	A data length code is a 4-bit bitfield that defines the length of the transmitted payload. The maximum length of a payload to be transmitted by the CAN frame is 8 bytes.
Data	64	The transmitted message. The maximum length is 64 bits (8bytes).

CRC	16	The cyclic redundancy check is used to detect the errors through calculate the checksum which is the number of the transmitted bits. Based on the CRC, the receivers get a negative or a positive acknowledgment in the ACK slot.
ACK	2	ACK is considered as the feedback which is given from the receiver to the sender confirming to the sender if the message transmitted was free of errors or not. By default the ACK bitfield in the frame transmitted from the sender is recessive, once the receiver receives it without any errors, it overwrites the recessive bit in the original message with a dominant one, otherwise, if there is an error the receiving node leaves this bit recessive and discard the message and After arbitration, the transmitting node repeats the message.
EOF	7	"End of frame" is seven successive bits that terminate the transmission of a data frame.
IFS	7	Inter-frame space Contains the time necessary by the controller to relocate a successfully received frame to the correct position in a message buffer region.

Table 2.1: The description of the bytes comping the standard CAN frame

Extended CAN Frame

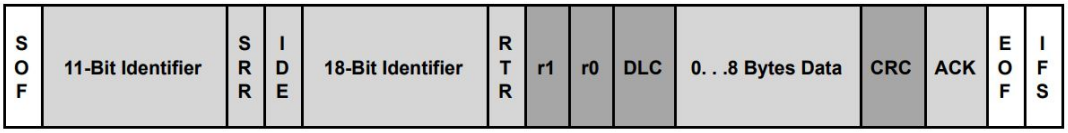

Figure 2.2: Extended CAN frame Structure [1]

Bit field	Length	Explanation
SRR	1	In the extended format, substitutes the RTR bit in the regular message location as a placeholder.
IDE	1	Always recessive in the extended frame, as the recessive bit in the IDE means that additional identifying bits are to come.
18-Bit Identifier	18	Holds the rest of the extended CAN frame identifier.
r1	1	The same as r0, Reserved bit (for possible uses in the future)

Table 2.2: The description of the bytes comping the extended CAN frame

CAN Network

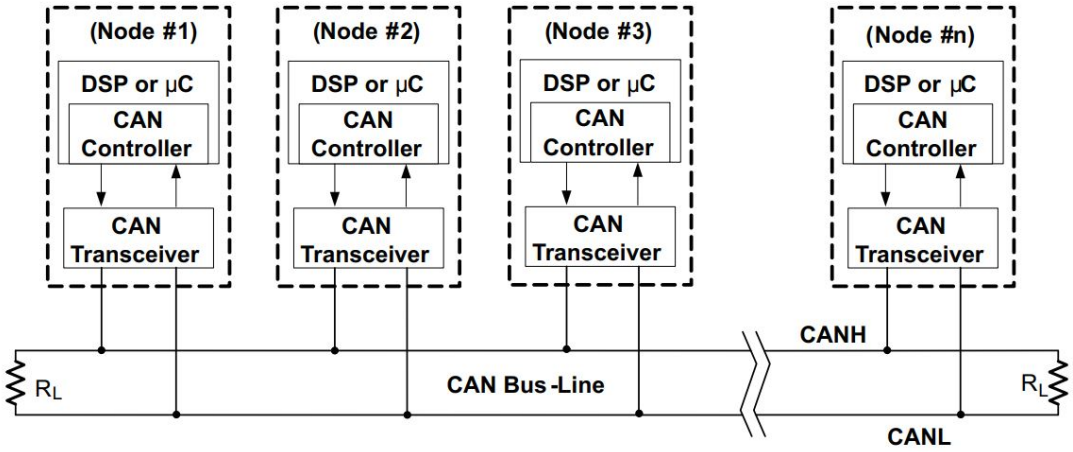

Figure 2.3: CAN network Structure [2]

Figure 2.3 shows the main components of the CAN network which are:

- CAN Nodes
- CAN Bus

The following lines explain each of these components in more detail.

CAN Bus

A CAN network is made up of a variety of CAN nodes that are connected by means of a physical communication medium which is called a CAN bus. In real life, the CAN bus is represented by an unshielded twisted two-wire line with a maximum resistance of 60 milliohms. The maximum available data rate on the CAN bus is 1 Mbps per second. It is permissible to extend the network by a maximum of about 40 meters. Bus termination resistors, which are located at the CAN network's endpoints, play an important role in avoiding transient events (reflections). Physical signal transmission in a CAN network is based on differential voltage transfer (differential signal transmission). This efficiently eliminates the effect of noise signals generated by the external environment, consequently, the CAN bus is made up of two lines: CAN high (CANH) and CAN low (CANL).

CAN Nodes

The CAN node represents the ECU on the can bus that can transmit, receive, and interact with it. To participate in CAN communication, an electronic control unit (ECU) requires a CAN interface composed of a CAN controller and a CAN transceiver.

- CAN controller: performs the communication activities specified by the CAN protocol. The quantity of CAN messages each CAN node sends or receives varies. There are also significant variances between the transmitting and receiving frequencies. These differences have led to the development of two CAN controller architectures: those with and without object storage. CAN controllers may be integrated or utilized as a stand-alone chip component. The microcontroller considers the CAN controller as a memory chip in this instance. The stand-alone option is more customizable, but the on-chip model requires less space and provides faster and more reliable communication between the microcontroller and the CAN controller.

- CAN transceiver: The CAN transceiver interfaces the CAN controller with the communication medium. Typically, the two components are electrically isolated by optical or magnetic decoupling, such that even if an overvoltage on the CAN bus destroys the CAN transceiver, the CAN controller and underlying host stay affected. There are always two bus pins on a CAN transceiver: one for the CAN high line (CANH) and one for the CAN low line (CANL) (CANL). To ensure electromagnetic compatibility, physical signal transmission in a CAN network is symmetrical, and the physical transmission medium consists of two lines. High-speed CAN transceivers are often distinguished from low-speed CAN transceivers. CAN transceivers with a high data rate support up to

1 Mbit/s. Low-speed CAN transceivers handle only up to 125 kbit/s data speeds. In ISO 11898, the maximum number of CAN nodes is stated as 32. The maximum number of CAN nodes varies significantly on the capability of the CAN transceivers used and whether the CAN network is high or low speed.

2.0.2 Bare-Metal application

Embedded devices are widespread. Today, there are more embedded processors in operation than there are inhabitants on the planet, and that means the number of devices has officially overtaken the number of people. Taking into account the increasing number of embedded devices and the connectivity between them. The term "bare-metal systems" refers to the fact that the software on many of these devices runs directly on the hardware and that they are relatively inexpensive. In these kinds of computers, the application operates in privileged low-level software and has direct access to the computer's processor and its peripherals; it does not pass through any of the software layers that are part of the operating system. These bare-metal systems are able to meet strict runtime guarantees on severely constrained hardware platforms. The advantages of the bare-metal applications are:

- Better performance for the same hardware
- Reliable
- For small applications, it is easier and faster

The disadvantages of the bare-metal applications are:

- The exponential growth in complexity that comes with increasing system size and functionality
- Multi-threading is not possible in one core
- Programming and adaptation of basic and standard functions are required for the different types of the hardware

2.0.3 OS-Based applications

Real-Time Operating Systems, also known as RTOS, are computer programs that are intended to work within strict time limits. Users may not be aware of the presence of real-time operating systems (RTOS) in embedded systems even if they are used in a variety of embedded systems. It is usual for a car to contain dozens of

microcontrollers, which is a good illustration of this condition that can be observed in the automotive industry. It is believed that 33 percent of the semiconductors used in a car are microcontrollers, and it is common for the automobile industry to employ them. Because of this, the costs of producing vehicles are reaching proportions that were previously only seen in the aerospace industry. Specifically, one-third of the total cost of a vehicle is spent on the vehicle's chassis, one-third is spent on the powertrain, and one-third is spent on electronics. OS base applications are the applications that operate on top of or are supported by the kernel, which serves as the operating system (OS). The "kernel" is the center or nucleus of the operating system, and it is responsible for controlling practically all the system's components. Therefore, the kernel is involved in every loop that occurs between the hardware and the software. The term "User Space Applications" is used to refer to the typical user-defined programs, and these applications have intermediate layers before they reach the hardware. To gain access to the hardware, they will need to navigate through the kernel space first. In User space applications, only a portion of memory is located above the kernel. The most significant distinction (and benefit) is that user space applications are not dependent on the underlying hardware. The kernel is made up of a number of modules, each of which has the ability to communicate directly with the underlying hardware. It offers the necessary abstraction to hide low-level hardware details to system or application programs, which ultimately led to it being independent of the underlying hardware. Process management, memory management, timers, inter-process communications (IPC), and device drivers for all hardware resources or power management are some of the kernel's primary actions or responsibilities. An application that is running in user space has the potential to have the kernel manage its scheduling and threading (which is an advantage of multi-threading), but on the other hand, it also has the potential to be interrupted at any time when the operating system needs to manage other calls and processes. Therefore, applications that require precise timing in order to access the GPIO are either not suitable or require additional processing. The advantages of the RTOS applications are:

- Priorities and multi-threading
- Community support
- Portable (independent from the hardware)
- Can be scaled

The disadvantages of the RTOS applications are:

- Demand a minimal amount of (powerful) hardware to perform the necessary Kernel Security protection.

- Complicated for use in relatively modest applications
- The learning curve is significantly steeper than for bare metal.
- It's possible that the system needs some updates.

Chapter 3

Setup

3.0.1 The used electronics

To set up a complete system that can send, receive, and interact with the messages sent on the CAN bus we used the following electronic components:

SPC58NN84C3 microcontroller

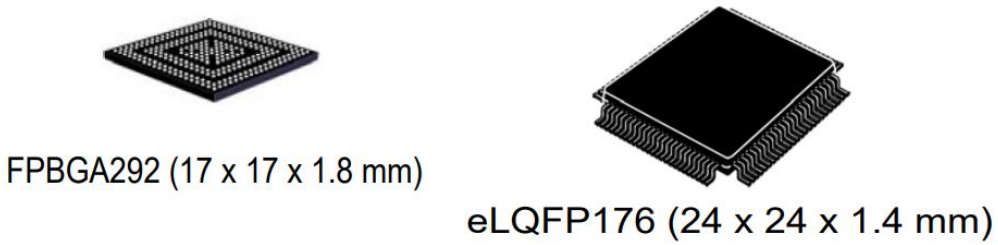

Figure 3.1: SPC58NN84C3 microcontroller [9]

SPC58NN84C3 microcontroller is a microcontroller that belongs to the SPC5x family. The SPC5 series of 32-bit Automotive Microcontrollers are designed to support a vast array of automotive applications, including Gateways, Electro Mobility, and ADAS, as well as Engine and Transmission management, Body, Chassis, and Safety. SPC5 microcontrollers have up to three cores functioning at up to 200 MHz and a junction temperature of 165 degrees Celsius. SPC58NN84C3 microcontroller has a lot of features and specifications, however, the following lines

will mention only some of them which are related to this thesis topic. The features of the SPC58NN84C3 microcontroller are:

- Seven communication interfaces of the LIN and UART communication (LINFlexD)
- Multi-channel direct memory access controllers an eDMA with 64 channels and another with 32 channels
- Seven modular controller area network (MCAN) modules, in addition to one time-triggered controller area network (M-TTCAN), all supporting flexible data rate (CAN-FD)
- Embedded memories in the CAN and eDMA peripherals.

SPC58NN84x has two CAN subsystems implemented:

- CAN Subsystem 0
- CAN Subsystem 1

The Controller Area Network (CAN) subsystem contains modular CAN (M CAN) modules, Time triggered CAN (M TTCAN) modules, and an intelligent CAN RAM controller for at least one subsystem. The CAN RAM controller includes ECC encoder/decoders for the Message RAM data and active transmit message buffer protection, as well as M_TCAN trigger memory from CPU write access. The subsystem corresponds to the little-endian data format. Features:

- Conforms to CAN protocol version 2.0, parts A and B, as well as ISO 11898-1: 2015
- Supports the CAN Flexible data rate (ISO CAN FD) protocol with a maximum of 64 data bytes on M_TTCAN and 64 data bytes on M_CAN.
- In standard CAN mode, bit rates are up to 1 Mbit/s.
- In ISO CAN FD mode, bit rates are up to 8 Mbit/s.
- Improved acceptance filtering
- Direct Message RAM access for Host CPU
- Two clock domains (CAN clock and Host clock)
- Error logging in CAN

- Two configurable Receive FIFOs
- Separate signaling takes place as High Priority Messages are received.
- Generic Slave Interface with 8/16/32 Bits for Connecting to Host CPUs Tailored to Individual Customers
- Configurable Transmit Event FIFO
- Up to 64 Receive Buffers can be used.
- Up to 32 transmit Buffers can be used.
- Configurable Transmit Queue
- Configurable Transmit FIFO
- The M_CAN's message memory is shared by several M CANs.
- Loop-back test mode that may be programmed
- Calibration, as well as debugging, are supported on M_CAN
- Both 11-bit and 29-bit identifiers are supported
- Tx Handler: This component controls the flow of messages between the CAN core and the external Message RAM. It is possible to define a maximum of 32 Tx Buffers for the transmission process. Tx buffers can function in one of three ways: as a standalone Tx Buffer, as a Tx FIFO, as part of a Tx Queue, or as a mix of all three of these uses. A Tx Event FIFO is responsible for storing Tx timestamps together with the Message-ID that corresponds to them. In addition to that, transmit cancellation is supported.
- Rx Handler: This component is responsible for controlling the movement of received messages from the CAN core to the message RAM located outside. The Rx Handler is equipped with two Receive FIFOs, each of which has a capacity that may be customized, as well as up to 64 Rx Buffers that are dedicated to storing of all messages that have been deemed acceptable after being filtered. In contrast to a Receive FIFO, a dedicated Rx Buffer is used just for the storage of messages that have been assigned a particular identification. A timestamp for Rx transmissions is saved. along with each individual message. For 11-bit IDs, a maximum of 128 filters may be created, whereas a maximum of 64 filters can be defined for 29-bit IDs.
- Two different interrupt lines are provided by the module. Separately, each of the interrupt lines can be set to active or inactive status.

Acceptance filters can be configured for standard identifiers or can be configured for extended identifiers. This capability is offered by the M CAN. The acceptance filtering procedure terminates after the first element that matches. These filter components will not be considered for this message. When all identifiers are received, the process of acceptance screening may then begin. After the completion of the acceptance filtering process and the determination of whether or not a matching Rx Buffer or Rx FIFO exists, the Message Handler will begin writing the received message data in blocks of 32 bits to the Rx Buffer or Rx FIFO that was determined to be a match. If the CAN protocol controller has determined that an error condition exists, such as a CRC error, then this message is rejected, which has the following effect on the Rx Buffer or Rx FIFO that was affected. Features of the acceptance filtering:

- Filtering elements are configured as:
  - Filter for a specific identifier
  - Filter for a range of identifiers
  - Classic bit mask filter
- Each filter element may be set to accept or reject incoming data according to the user's preferences.
- Separate activation or deactivation of each filter element is possible at any time.
- Filters are checked-in sequence, and the execution stops once the first matching element is found

UART in SPC58NN84C3 is part of LINFlexD which is a controller that provides support for some basic UART transfers. In LIN/UART mode of operation, the LINFlexD offers support for multiple channels as well as a parametric DMA Tx/Rx interface. UART mode features:

- Full-duplex communication
- 1/2/3 stop bits
- Baud rates up to 25 Mbit/s.
- 12-bit + parity reception
- 4 bytes are reserved for the receiving buffer, while 4 bytes are reserved for the transmission buffer.

SPC58XXADPT292S

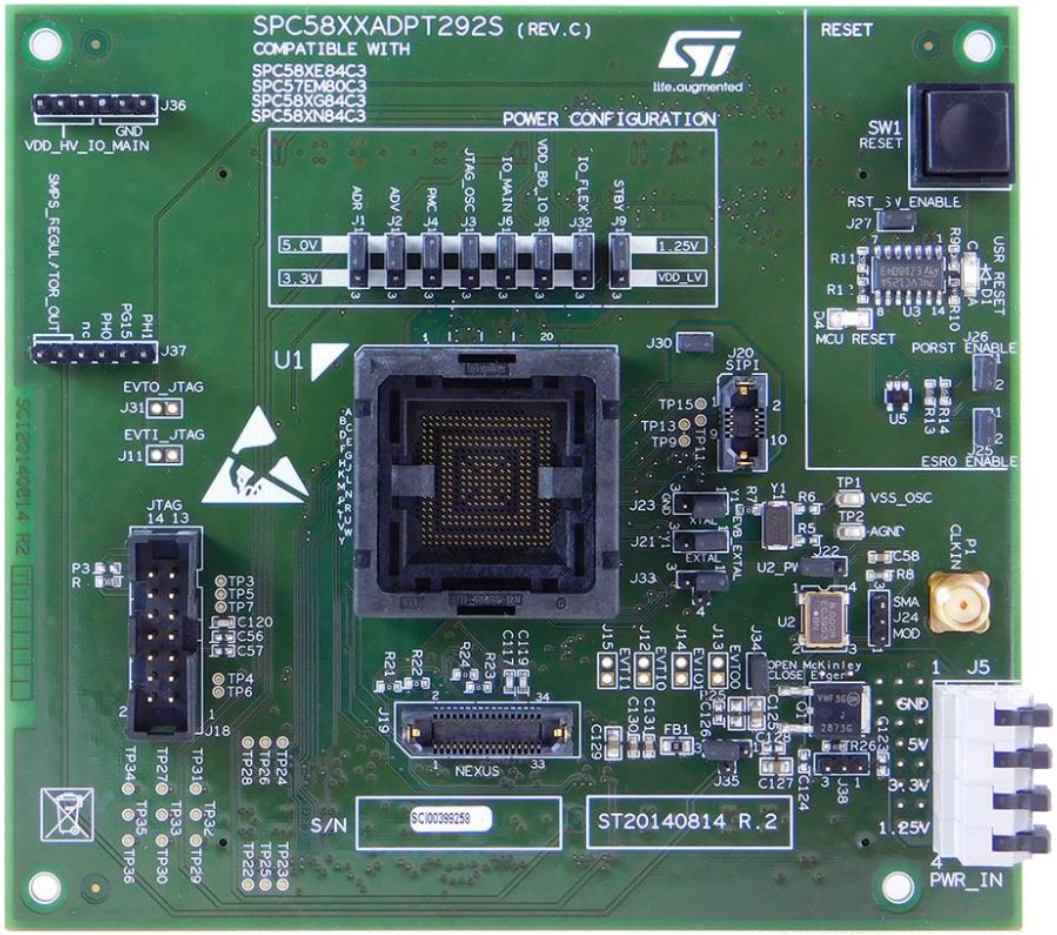

Figure 3.2: SPC58XXADPT292S Front [4]

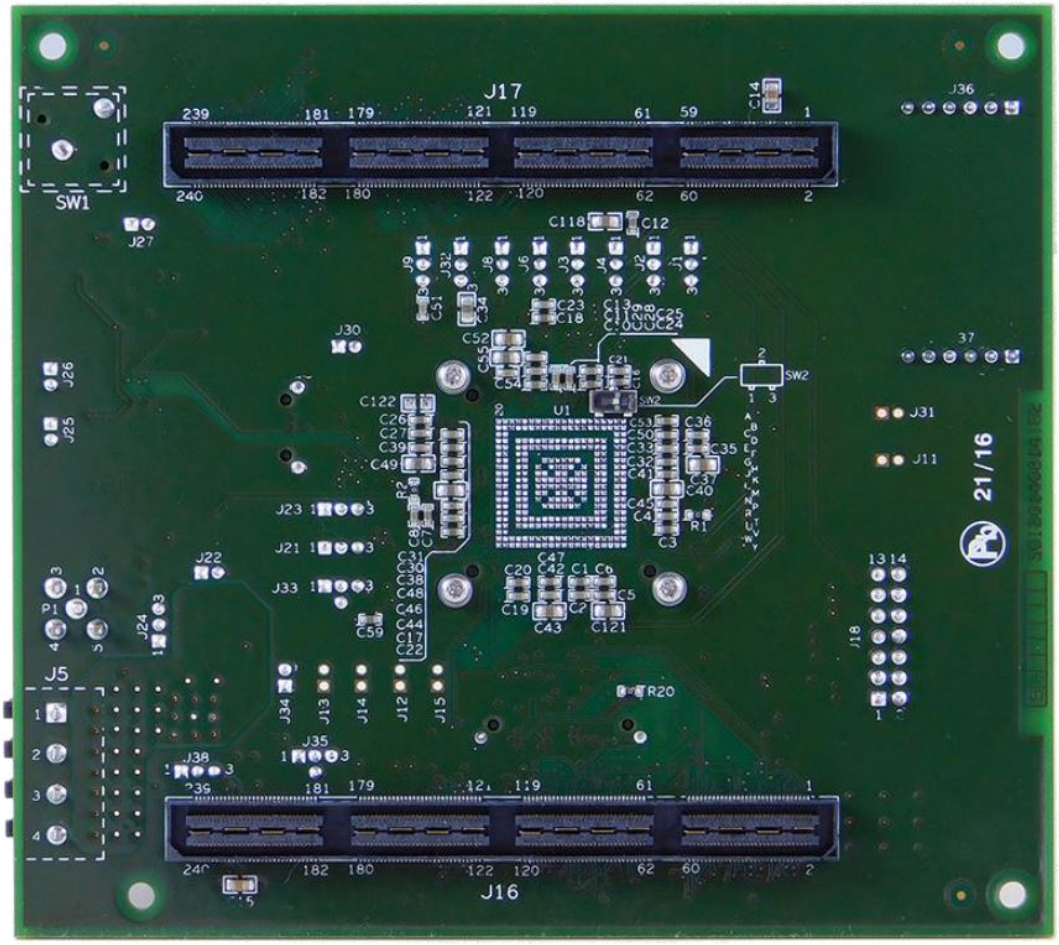

Figure 3.3: SPC58XXADPT292S Back [4]

The SPC58XXADPT292S minimodule is an evaluation board that is compatible with the SPC58XE84C3, SPC57EM80C3, SPC58XG84C3, AND SPC58XN84C3 microprocessors from STMicroelectronics. These microprocessors come in the LFBGA292 package. The SPC58XXADPT292S minimodule was developed to be attached to the SPC57xxMB motherboard. This provides a method for easy customer evaluation of the compatible microprocessors as well as facilitates the development of both hardware and software. High-density connectors provide an EVB-MCU daughter card interface. These connectors also support the developers in evaluating all of the device's peripherals.

SPC7XXMB

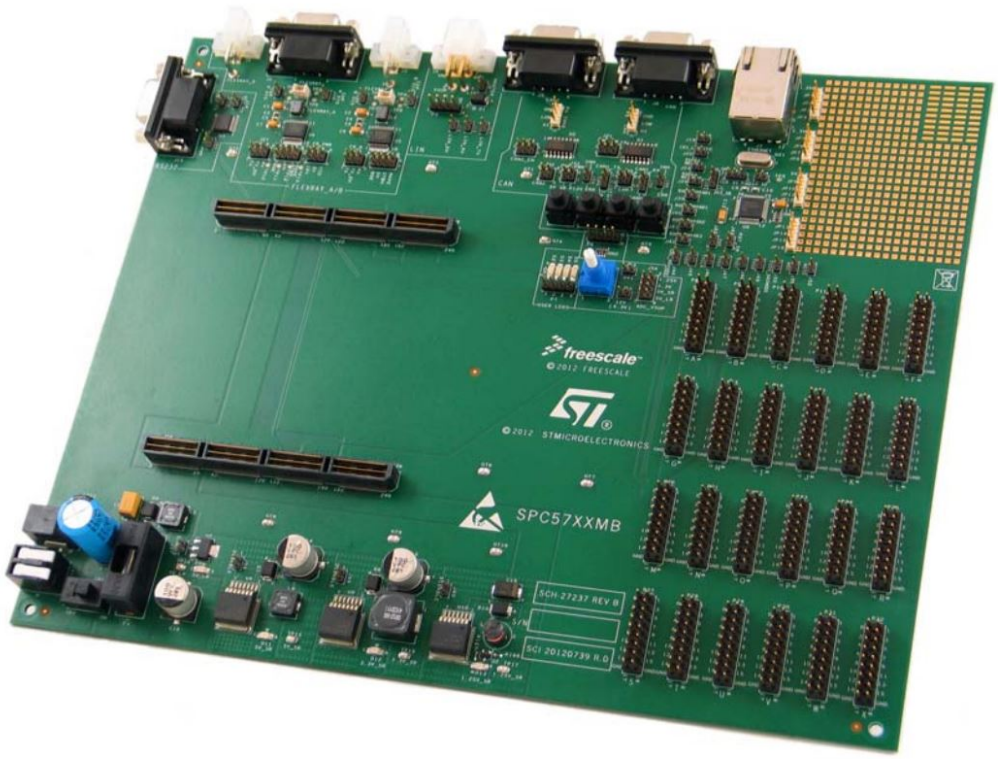

Figure 3.4: SPC7XXMB [10]

The EVB was designed to be used on a workbench or in a laboratory to provide a mechanism for easy customer evaluation of SPC57xx microprocessors and to assist the development of both hardware and software. The EVB was designed as a modular development platform to provide the greatest amount of flexibility as well as ease of use. There is no microcontroller unit (MCU) on the main board of the EVB. Instead, the MCU is connected to a daughter card which is an MCU (occasionally referred to as an adapter board). By using this approach, it will be possible to use the same EVB platform for several MCU variants and packages that belong to the SPC57xx family. The EVB and MCU daughter cards are connected to one another through a high-density connection interface.

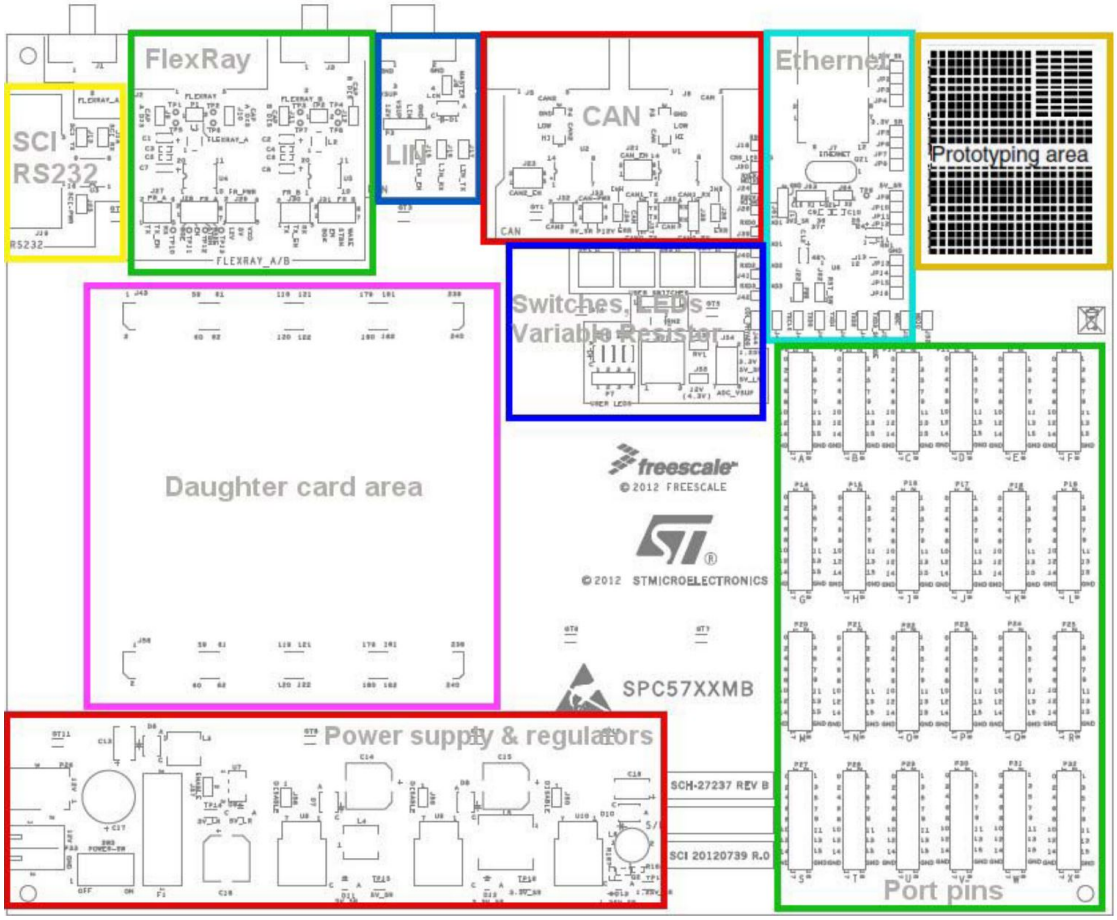

Figure 3.5: SPC7XXMB structure [5]

SPC57XXMB is considered as the window for the MCU to the external world, it provides the MCU with many I/O pins, connectors, switches, power supply and many other functional interfaces. These interfaces are grouped together into many group called functional groups and each one of them has its own role to make the MCU capable of communicating with the external world. The motherboard can provide a lot of key features such as:

- Through the utilization of MCU daughter cards, it provides support for a variety of SPC57xx MCUs.
- Physical interface of RS232/SCI as well as a female connector of DB9 type.
- LINFlexD interface.
- Two independent and configurable CAN interfaces

- 4 switches
- 4 Connectable LEDs

Ft232R

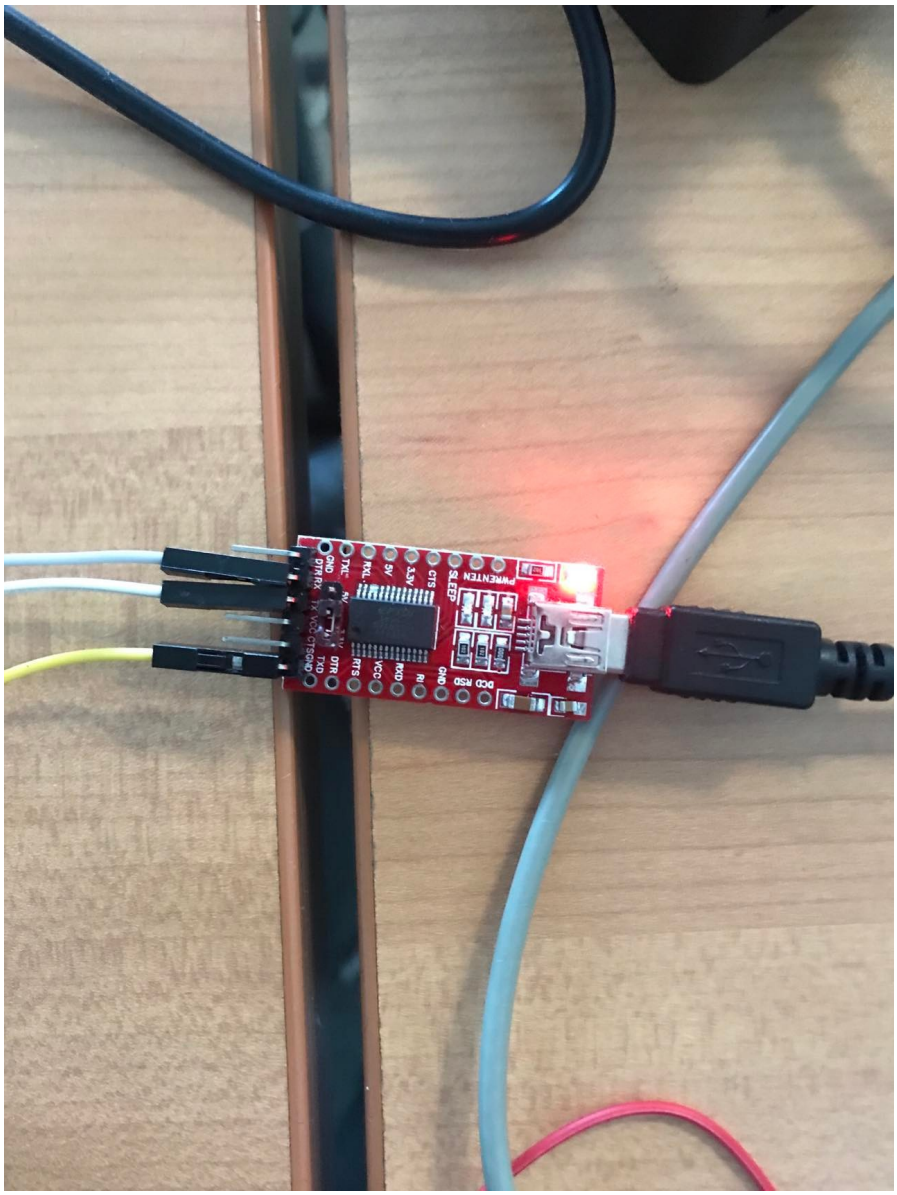

Figure 3.6: Ft232R

FT232R is a USB to a serial interface that includes an option of clock generator output. FT232R provides communication between the MCU and the personal PC through the UART. It is supported by many GUIs which can facilitate the development and debugging of the MCU.

Logic analyzer

Figure 3.7: The Logic analyzer

A logic analyzer is an electronic component that is used for capturing and displaying numerous signals generated from a digital system or digital circuit. It is a very helpful tool for debugging as well as verifying digital signals.

3.0.2 Connections

SPC58NN84C3 – SPC58XXADPT292S – SPC57XXMB connection

The result from the connection between SPC58NN84C3 MCU, SPC58XXADPT292S and SPC57XXMB can be considered as the one unit (Test unit), on which all the tests and development in this project are being done.

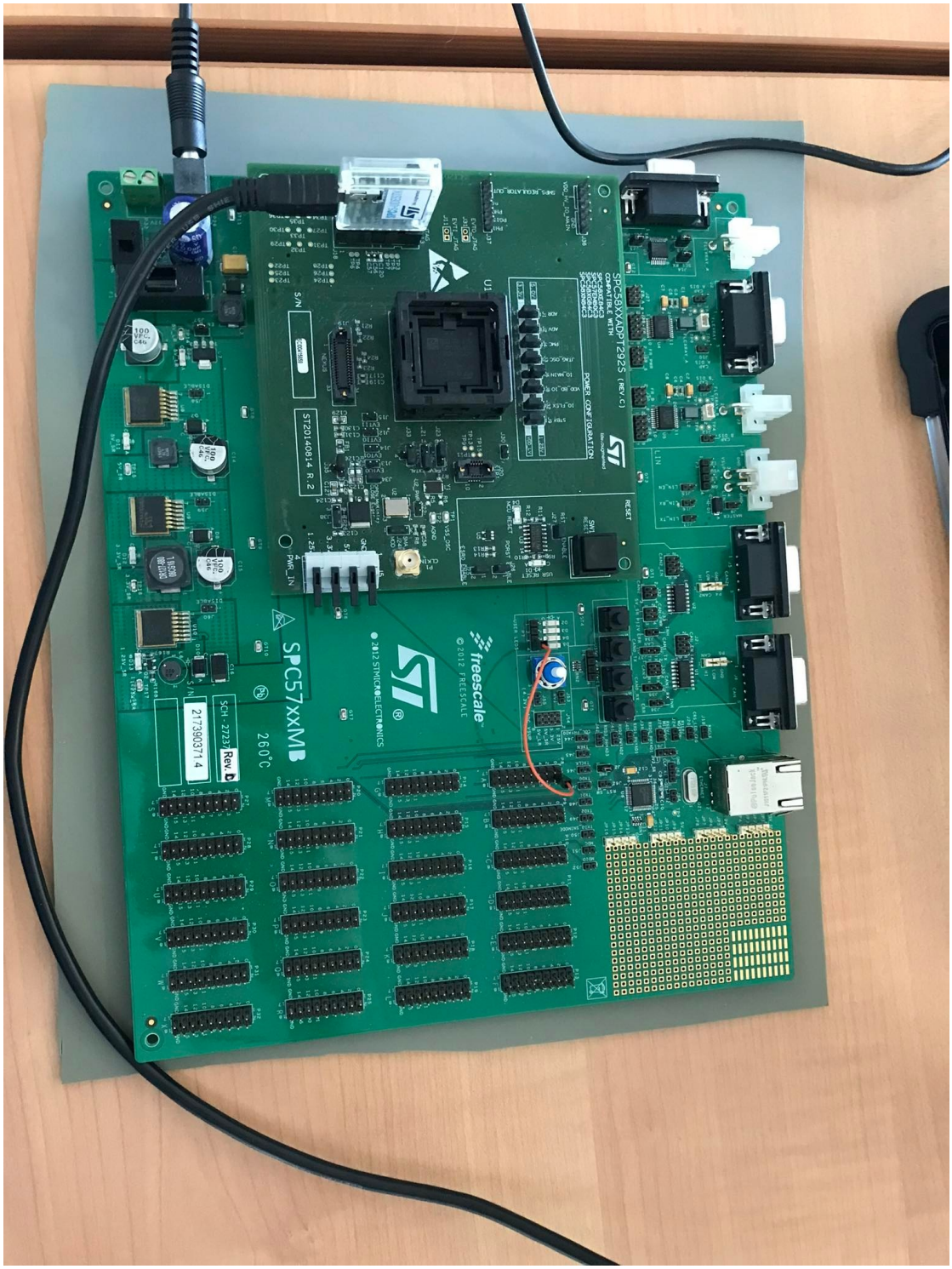

Figure 3.8: The test unit

As shown in figure 3.8, SPC58NN84C3 is placed in the middle of SPC58XXADPT292S and fixed by jaws, while SPC58XXADPT292S and SPC57XXMB are connected to one another through a high-density connection interface as shown in the figure 3.9

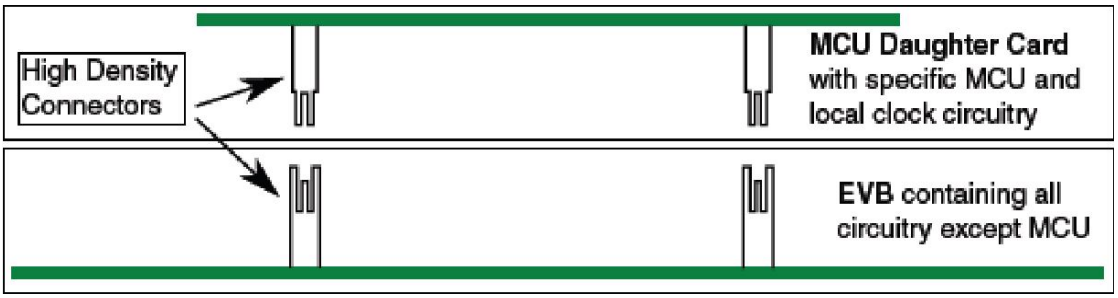

Figure 3.9: High Density connection [5]

Test Unit – Test Unit Connection

From CAN bus point of view, every test unit is considered as a CAN node which has to be connected to the CAN bus to be able to communicate with the other CAN nodes. A node could be connected the CAN bus by connecting the CAN high, CAN low and CAN ground of the node to CAN high, CAN low and CAN ground of the CAN bus respectively. Since there are only two can nodes in this project, so that, by connecting the two nodes we are creating a new bus composed of these two nodes.

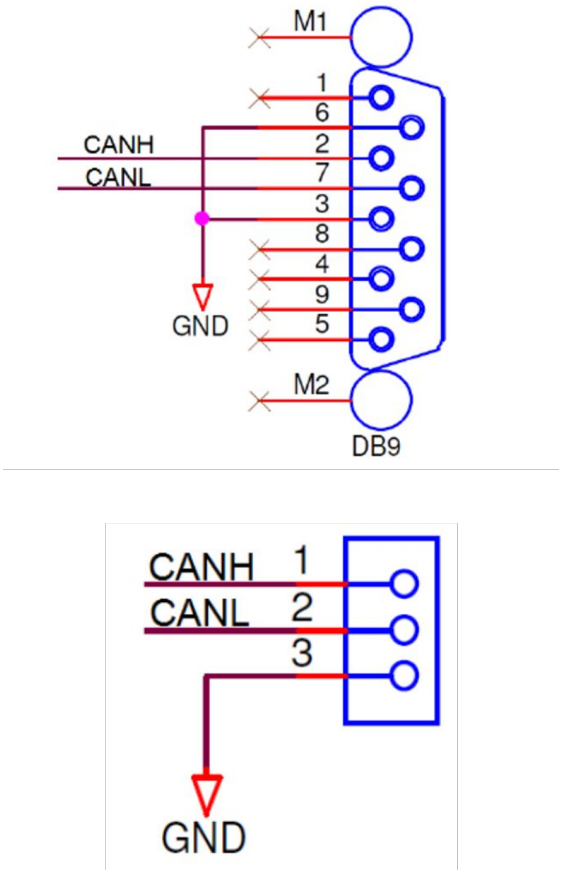

Figure 3.10: DB9 Three-pin connector [5]

Figure 3.10 shows that to connect the test unit to the bus we have two options:

- Connecting the two units through a DB9 connector.

- Three-pin header interface connector.

In this project the Three-pin header interface connector is used, in addition, a LED is connected to the receiver of the CAN controller, which blinks when any message is received by the controller.

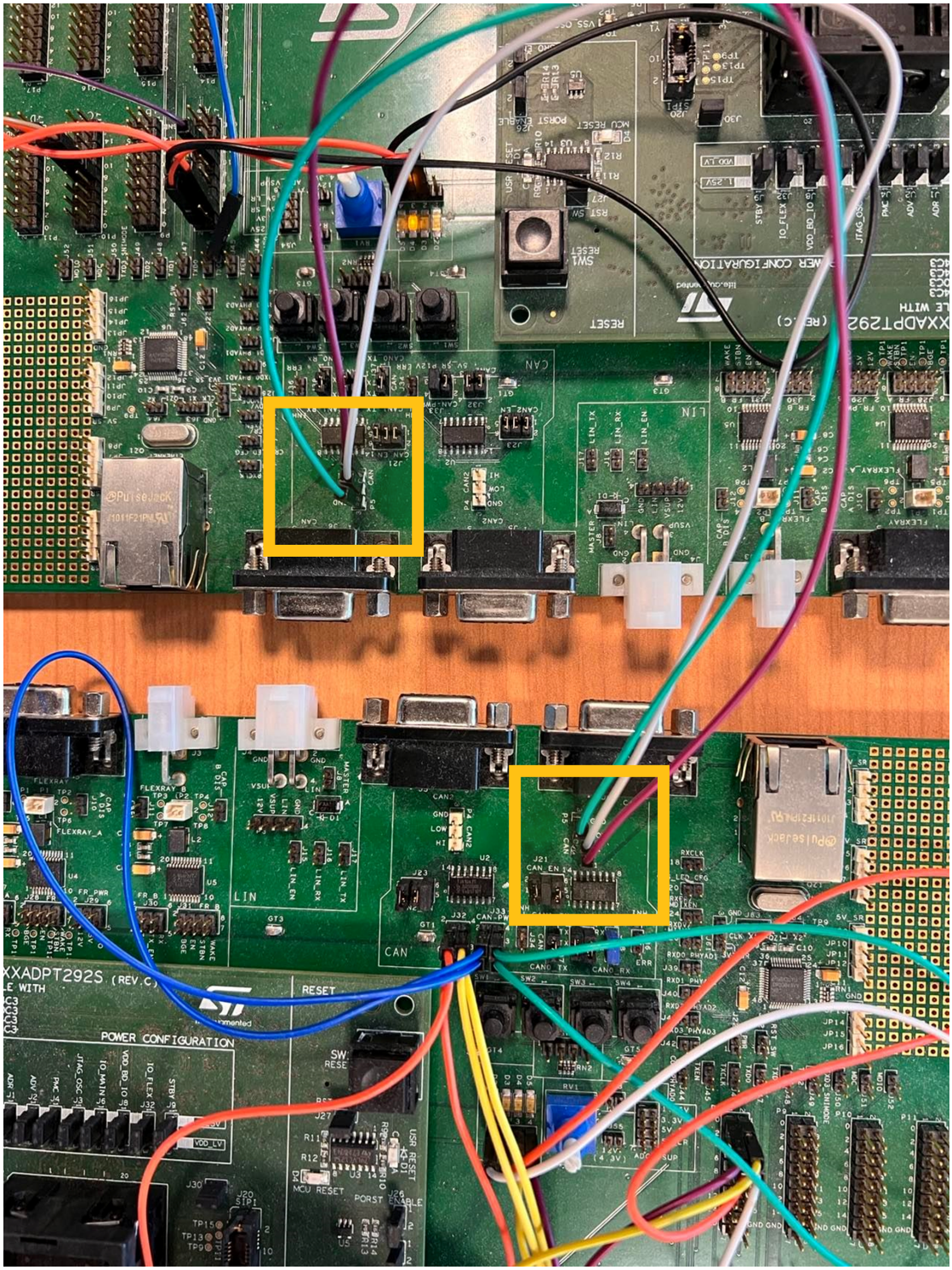

Figure 3.11: Test units connection

In figure 3.11, the connection between the three-pin header interface connectors of the two test units is highlighted.

Test Unit – UART Module Connection

The test unit is connected to the UART module by connecting the configured pin to be the receiving pin of the UART in the ''port pin functional group" of the test unit to the transmission pin of the UART module and by connecting the configured pin to be the transmission pin of the UART in the ''port pin functional group" of the test unit to the receiving pin of the UART module while having the same ground.

Figure 3.12 shows a schematic for this connection and the transmission-receiving connection is highlighted.

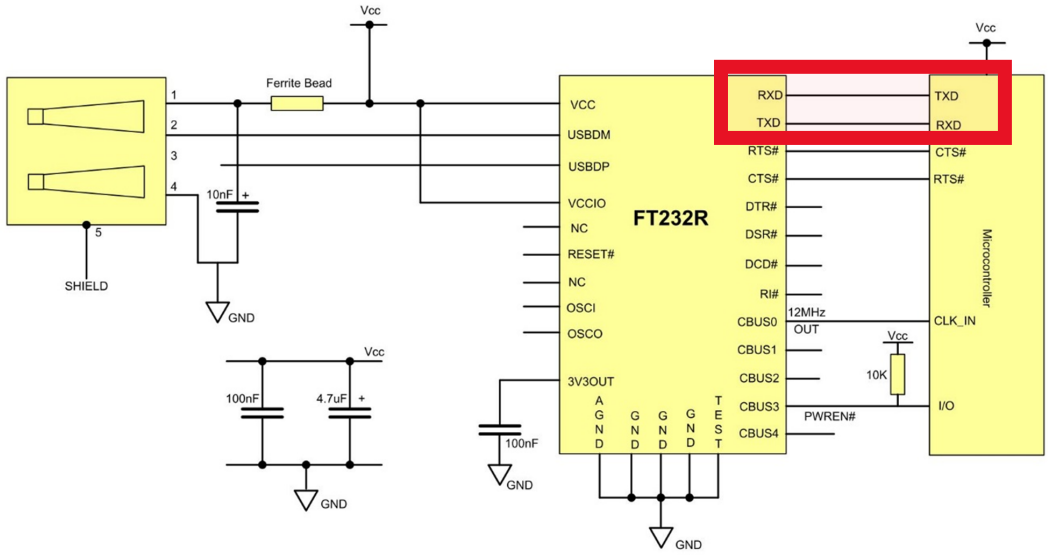

Figure 3.12: Test unit - UART connection [6]

3.0.3 Configurations

Hardware configurations

To configure the test unit many hardware configurations has to be implemented, however, in this study, only the configurations related to the CAN module are going to be mentioned. As shown in figure 3.5, the two CAN subsystems of the test unit could be configured independently, and each of these CAN subsystems can be switched on in the test unit by connecting the configuration jumpers, taking into consideration that, the different connections of these jumpers control the operating

mode of the CAN subsystem. Table 3.13 shows the different implementations of these jumpers, and correspondingly the different operating modes.

Jumper	Label	Description
J23	CAN2_EN	PHY U2 configuration 1-2: WAKE to GND 3-4: STB to 5V 5-6: EN to 5V
J32	CAN2	1-2: PHY TX to MCU 3-4: PHY RX to MCU
J33	CAN-PWR	1-2: 5.0V_SR to PHY U2 VCC 3-4: 12V to PHY U2 VBAT
J34	INH/ERR	PHY U2 signal out 1: ERR 2: INH
J21	CAN_EN	PHY U1 configuration 1-2: WAKE to GND 3-4: STB to 5V 5-6: EN to 5V
J35	CAN-PWR	1-2: 5.0V_SR to PHY U1 VCC 3-4: 12V to PHY U1 VBAT
J37	TTCAN_TX/MCAN1_TX	PHY U1 TX to MCU 1-2: TTCAN TX 2-3: MCAN1 TX
J38	TTCAN/MCAN1	PHY U1 RX to MCU 1-2: TTCAN RX 2-3: MCAN1 RX
J36	INH/ERR	PHY U1 signal out 1: ERR 2: INH

Figure 3.13: Test unit pin connections [5]

Figure 3.14 shows a schematic for the detailed connections of the can Module inside SPC57XXMB.

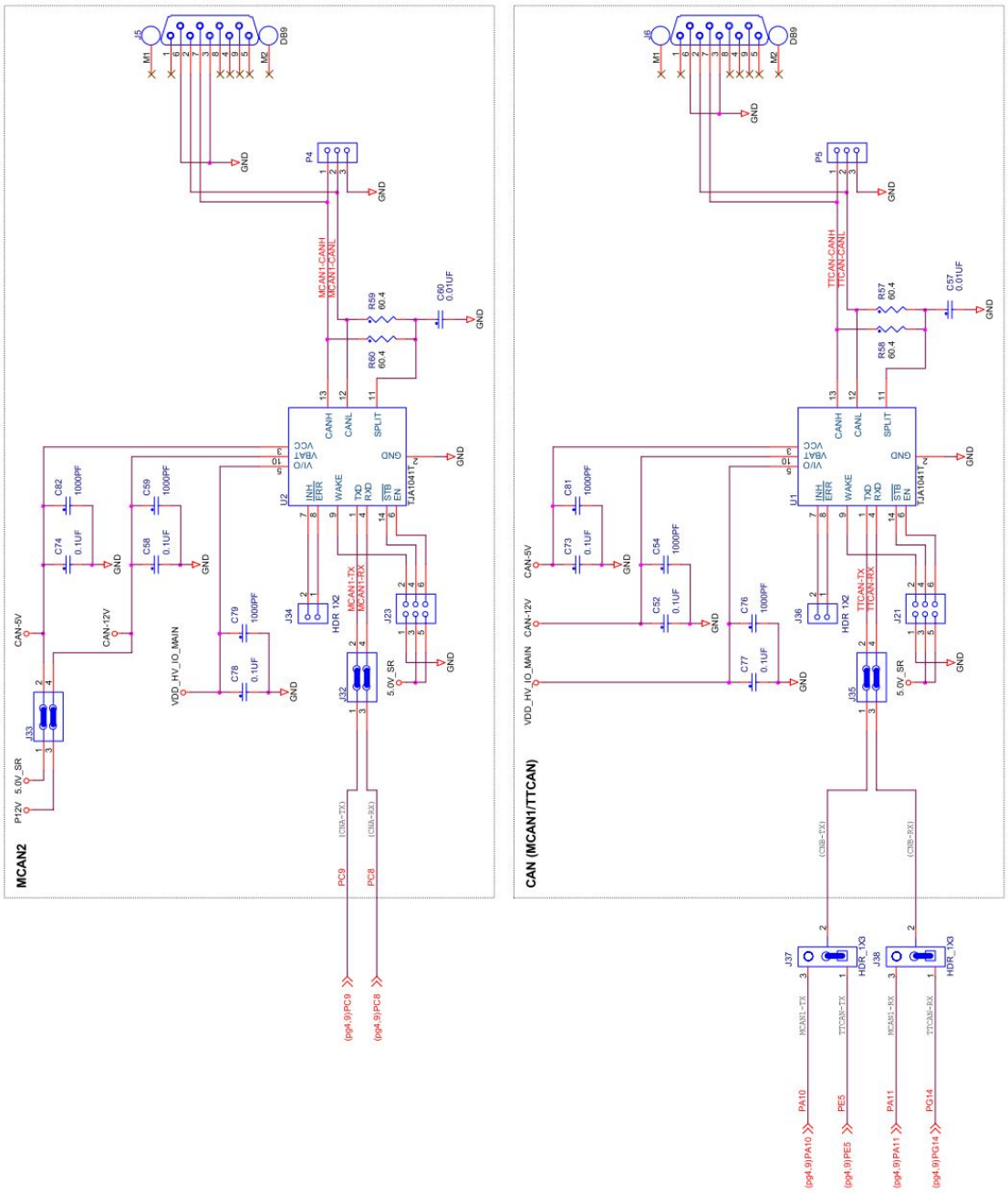

Figure 3.14: The schematic of the detailed connections of the can Module inside SPC57XXMB [5]

Software configurations

Every CAN subsystem in SPC58NN84C3 can operate in many operating modes, which are:

- Normal mode
- External loopback mode
- Internal loopback mode
- CAN FD operation
- Transceiver delay compensation
- Restricted operation mode
- Bus monitoring
- Disabled automatic retransmission
- Power-down (sleep mode)

Since this study concerns only the first phases of testing and developing the can peripheral in SPC58NN84C3, only normal mode, internal loopback mode, and external loopback mode are going to be mentioned. The main configurations of the CAN peripheral can be configured through a special configurator implemented inside the SPC5 studio. Table 3.1 specifies the needed configurations to operate the CAN peripheral in different operating modes.

Configuration	Value	Explanation
Loopback	No loopback External loopback Internal loopback	Switches between the different operating modes according to the selection
Endianness	Big_Endianness	The arrangement of the bytes that make up a word of digital data stored in a computer's memory is referred to as endianness.
Clock Prescaler NSJW NTSEG1 NTSEG2	1 3 10 3	These four configurations together define the baud rate, these mentioned values provide a bit rate of 500000

Interrupt	LINE0 LINE1 DISABLE	If line 0 or line 1 are selected the interrupt line becomes active and the CPU will stop and receive the can frame once it is received through the interrupt, otherwise if it is disabled, the CPU will carry on until it goes to the address of the received message in the memory and read it. This makes the interrupt faster and more complex
Callback	"the identifier of the interrupt call back function"	Specified only in case of an active interrupt, otherwise, it should be left empty
Number of RX buffers	1	Can be changed in case of more than RX buffer used
RX buffer filters	Filter type: Standard Filter value: 0x7f0 Rx buffer number: 0 OR Filter type: Extended Filter value: 0x8901234UL Rx buffer number: 0	This configuration set up the filter of the received frames. Frames that don't include these filter IDs will be rejected by the CAN controller

Table 3.1: The configurations to set up in SPC5 studio in order to operate the CAN peripheral in different operating modes.

Note: the CAN subsystem to be configured should be enabled from the configurator before using the table 3.1. UART Serial communication can be configured through the same configurator in the SPC5 studio as in table 3.2.

Controller/LED	Notes	

Baud rate	38400	The pace at which information is transmitted across a communication channel is referred to as the baud rate.
Mode	8BITS_PARITY_ NONE	A number's evenness or oddness can be described using the concept of parity. The parity bit is a way for the receiving UART to tell if any data has changed during transmission
API behavior	Asynchronous	Asynchronous communication indicates that there is no clock signal to synchronize the bits that are sent from the transmitting device to the receiving end of the connection.
TX callback RX callback	Identifier of the UART transmission/receiving a call back function	Within the interrupt service routine of the UART transmitter is where you will find the call to the callback function. After the UART module has finished transferring a character, the interrupt service routine of the UART transmitter is run once. Naturally, the callback function is also run once at this point

DMA Enable	Enabled	This configuration is a check box that is checked to enable the DMA feature of the serial communication

Table 3.2: The configurations to set up in SPC5 studio in order to operate the UART.

Each of the modules, such as CAN modules and UART modules, as well as the LEDs that are connected to the test unit, need to be connected to either one or two pins in the pin port section of the test unit, one of them serving as the transmission pin and the other pin serving as the receiving pin. The port pin section is shown in Figure 3.5 For the CAN modules and the UART modules, these pins allow them to send and receive messages to and from external devices, consequently, allowing the developers to control the performance of these modules, while for the LEDs, these pins connect the LEDs to the MCU which could control them through the code implemented on the MCU, consequently, it provides instantaneous and visual feedback to the developer which could facilitate the developing and troubleshooting process, for example, the LED could be programmed to blink whenever a message is received in the memory through the UART, consequently, during the troubleshooting process, if the whole system doesn't work, this LED could be checked immediately, if it blinks it means that there is a transmission, if not, it means that there is an issue in the UART which has to be solved. This solution is a way faster solution for the developers than going inside the memory to check the transmission. Table 3.3 shows the connection and the configurations of the used pins in this study.

Controller/LED	Notes

The transmission pin of CAN subsystem 1	• As shown in figure 3.14, the CAN subsystem 1 transmission controller is implicitly connected to some pins in the pin port section, in this study we choose PA [10] • Although the CAN controller is connected implicitly to the pin, however, the pin PA [10] should be configured as a CAN transmission pin to be able to interpret the CAN message correctly • Pin PA [10] could be connected to an external device (e.g., logic analyzer) to display and troubleshoot the transmitted message as well as facilitate the CAN behavior development.
The receiving pin of CAN subsystem 1	• As shown in figure 3.14, the CAN subsystem 1 receiving controller is implicitly connected to some pins in the pin port section, in this study we choose PA [11] • Although the CAN controller is connected implicitly to the pin, however, the pin PA [11] should be configured as a CAN-receiving pin to be able to interpret the CAN message correctly. • Pin PA [11] could be connected to an external device (e.g., logic analyzer) to display and troubleshoot the received message as well as facilitate the CAN behavior development

The transmission pin of CAN subsystem 2	• As shown in figure 3.14, the CAN subsystem 2 transmission controller is implicitly connected to some pins in the pin port section, in this study we choose PC [9] • Although the CAN controller is connected implicitly to the pin, however, the pin PC [9] should be configured as a CAN transmission pin to be able to interpret the CAN message correctly. • Pin PC [9] could be connected to an external device (e.g., logic analyzer) to display and troubleshoot the transmitted message as well as facilitate the CAN behavior development.
The receiving pin of CAN subsystem 2	• As shown in figure 3.14, the CAN subsystem 1 receiving controller is implicitly connected to some pins in the pin port section, in this study we choose PC [8] • Although the CAN controller is connected implicitly to the pin, however, the pin PC [8] should be configured as a CAN-receiving pin to be able to interpret the CAN message correctly. • Pin PC [8] could be connected to an external device (e.g., logic analyzer) to display and troubleshoot the received message as well as facilitate the CAN behavior development.

UART transmission pin	• the UART transmission controller is implicitly connected to some pins in the pin port section, in this study we choose PF [3] • Although the UART controller is connected implicitly to the pin, however, the pin PF [3] should be configured as a UART transmission pin to be able to interpret the UART message correctly. • Pin PF [3] could be connected to an external device (e.g., FT232R module) to display and troubleshoot the transmitted message as well as facilitate the UART behavior development
UART receiving pin	• the UART receiving controller is implicitly connected to some pins in the pin port section, in this study we choose PF [2] • Although the UART controller is connected implicitly to the pin, however, the pin PF [2] should be configured as a UART transmission pin to be able to interpret the UART message correctly. • Pin PF [2] could be connected to an external device (e.g., FT232R module) to display and troubleshoot the transmitted message as well as facilitate the UART behavior development.

LED D2	• LED D2 is connected to the pin PA [1] • Unlike the CAN controllers and UART controllers, any pin in the port pin section can be configured as a general-purpose input-output pin that switched on and off the led • Unlike the CAN controllers and UART controllers, any pin in the port pin section can be configured as a general-purpose input-output pin that switched on and off the led
LED D3	• LED D3 is connected to the pin PA [2] • Unlike the CAN controllers and UART controllers, any pin in the port pin section can be configured as a general-purpose input-output pin that switched on and off the led • Pin PA [2] must be connected to the pin of the LED D2 shown in the schematic of the figure 3.5. Figure 3.15 shows the physical connection between the LED D2 and the pin PA [2] on the test unit

Table 3.3: The connection and the configurations of the used pin

Each of the CAN subsystems, the UART controller, and the LEDs could be configured only with the implicitly connected pins in the test unit. Table 3.4 specify the list of the configurable pin for each of them

Controller/LED	Notes

The transmission pin of CAN subsystem 1	PA [10] PC [2] PD [3] PF [7] PF [8] PH [8] PH [13] PL [7] PL [9]
The receiving pin of CAN subsystem 1	PA [11] PC [1] PD [2] PF [3] PF [6] PH [10] PH [12] PJ [2] PL [6] PL [10
The transmission pin of CAN subsystem 2	PA [2] PC [9] PH [7] PJ [6] PM [10]

The receiving pin of CAN subsystem 2	PA [1] PA [11] PA [13] PB [9] PC [8] PF [5] PH [9] PJ [7] PL [3] PL [0]
UART transmission pin	PA [10] PB [10] PD [14] PD [15] PE [10] PF [3] PH [2] PJ [6] PL [1]

UART receiving pin	PA [11] PB [11] PD [14] PD [15] PE [10] PF [2] PJ [7] PL [0]

Table 3.4: The configurable pins for the UART and each CAN subsystem

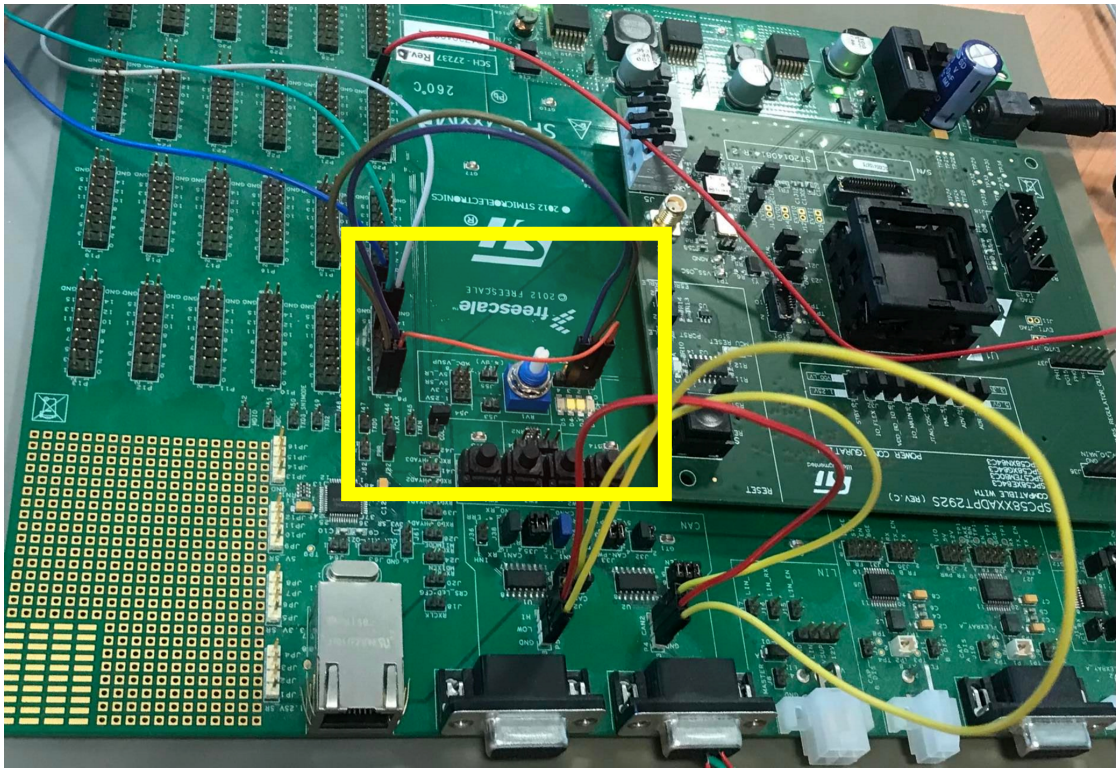

Figure 3.15: LEDs pin connection

Chapter 4

Proposed Method

The bare metal application introduced in this study is composed of data types, definitions, and many functions, these functions can be classified into three types of functions which are:

- The functions which are used to switch on the CAN and to transmit and receive the messages through it, and these functions are:
  - CAN initialization function
  - CAN start function
  - Enabling interrupts
  - CAN Transmission function
  - CAN message reading function
  - CAN Stop
- The functions which are used to switch on the UART and to transmit and receive the messages through it, and these functions are:
  - UART initialization function
  - UART start function
  - UART transmission / receiving call back functions
  - UART message reading function
  - UART message writing function
- Auxiliary functions, are the functions used to improve the performance of any of the CAN or the UART
  - Operating mode control function
  - The function that controls the blinking of the LEDs
  - A comparing function that compares two inputs

4.0.1 CAN initialization function

Expressed as the “can_lld_init()” function in the bare-metal application. No arguments are passed to the “can_lld_init” function. The CAN initialization means that:

- Initialize the clock for the CAN subsystems
- Initialize the CAN controllers with a NULL configurations
- Initialize the shared ram for the CAN subsystems.

The “can_lld_init()” function is called inside another function, which is “componentsInit()”. Initialization functions are called only once in the application and for all the CAN controllers.

4.0.2 CAN Start function

Expressed as the “can_lld_start(CANDriver *canp, const CANConfig *config)” function in the bare-metal application. The “can_lld_start” function receives the following arguments

- A pointer to the targeted CAN driver object.
- The corresponding configurations.

The CAN start means that:

- Assign the selected configurations to each of the corresponding CAN controllers, for example, the baud rate, the operating mode, and the endianness.
- Set the pointers of each CAN to their corresponding values in the memory
- Calibrate the CAN clock
- Access the registers which controls the operating mode and set it according to the selected operating mode in the configurations
- Enable/Disable the CAN FD according to the selected configurations.
- Set the baud rate according to the selected configurations
- Configures the transceiver compensation delay if enabled in the configurations
- Set the driver’s receiving and transmitting buffer size to max 64 bytes.

- Initialize the global filters
- Initialize the standard filters
- Configure the standard filters
- Initialize the extended filters
- Configure the extended filters
- Initialize the receiving method which could be one o the following
  - first input first output (FIFO)
  - receiving buffer
- Initialize the receiving method which could be one o the following
  - first input first output (FIFO)
  - receiving buffer
  - Queue
  - Mixed FIFO
  - Mixed Queue
- Enables the interrupt for RX buffer or FIFO
- Enable BUS OFF interrupt

Start functions are called for each of the enabled CAN controllers only once in the application.

4.0.3 Enabling interrupts

Expressed as the "irqIsrEnable()" function in the bare-metal application. No arguments are passed to the "irqIsrEnable" function. This function globally enables interrupts.

CAN transmission function

Expressed as the "can_lld_transmit (CANDriver *canp, uint32_t msgbuf, const CANTxFrame *ctfp)" function in the bare-metal application. The "can_lld_transmit" function receives the following arguments

- A pointer to the CAN driver object

- The method of transmitting e.g., FIFO.
- A pointer to the CAN frame to be transmitted

The "can_lld_transmit" function performs the following operations

- Check if the frame size is correct
- Set the transmission method according to the second passed argument
- Check if the message box is free to transmit
- Write the transmission buffer
- Send the message

This function is called every time It is needed to send a message through the CAN bus. If CAN frame is sent successfully, "can_lld_transmit" returns zero otherwise it returns other values.

4.0.4 CAN message reading functions

The CAN controller doesn't need a function to be called to receive a message from the CAN bus, Automatically, all the CAN nodes which are connected to the CAN bus receive all the CAN frames, however, these frames on the CAN bus must carry specific identifiers configured in the CAN filter to be allowed to pass through the filter to the CAN memory. The CAN reading functions are called to read the already received CAN messages, not to receive them. The CAN is able to read the received messages through two different functions which are:

- The interrupt's call back function
- The message reading function

The Interrupt's call-back function

The call-back function is a function that takes place whenever a defined operation has been performed. In the case of the Interrupt, the call-back function is a function that is called whenever an interrupt is activated due to the reception of a new message. The Interrupt of the message reception must be enabled, and the call back function header should be defined in the CAN configuration section. The interrupt call-back function is expressed as the "mcanconf_rxreceive (uint32_t msgbuf, CANRxFrame crfp)" and "mcanconf_rxreceive1 (uint32_t msgbuf, CANRxFrame crfp)" functions in the bare-metal application. Every Interrupt, consequently, every call-back function, corresponds to a specific configuration, so that, the number of

the call-back functions must be equal to the number of the defined configurations, not the enabled CANs. For instance, if only two configurations are used to configure four can drivers, the number of the defined call-back functions is going to be two, not four. The “mcanconf_rxreceive” and “mcanconf1_rxreceive” receives the following arguments:

- Receiving buffer number or FIFO
- A Pointer to the CAN frame to be read

The message reading function

The message reading functions are ordinary functions that are executed when they are called according to their order of execution in the code. The message reading function is expressed as the “can_lld_receive(CANDriver *canp, uint32_t msgbuf, CANRxFrame *crfp)” function in the bare-metal application. The “can_lld_receive” receives the following arguments:

- A pointer to the CAN driver object
- Receiving buffer number or FIFO
- The address of the CAN receiving frame to which the received message will be copied.

To enable message reading by “can_lld_receive” function, the receiving interrupt must be turned off and the call-back header has to be removed from the configurations of selected CAN. The CAN driver which is configured to use the interrupt cannot read any CAN messages using “can_lld_receive” function. If CAN frame is sent successfully, “can_lld_receive” returns zero otherwise it returns other values.

4.0.5 CAN stop

Expressed as the “can_lld_stop (CANDriver *canp” function in the bare-metal application. The “can_lld_stop” function receives only the pointer to the targeted CAN driver as an argument The “can_lld_stop” function performs the following operations

- Stop the CAN
- Disable the interrupts.
- Erase the shared RAM associated with the targeted CAN driver

- Set the configurations of the CAN to NULL again

This function is not mandatory to be used in all the applications, as it is not always required to stop the CAN.

4.0.6 UART initialization function

Expressed as the "sd_lld_init()" function in the bare-metal application. No arguments are passed to the "sd_lld_init" function. The UART initialization means that:

- Initialize the clock for the UART driver.
- Initialize the receiving, transmission, and the error interrupts.
- Initialize the DMA for each UART driver in which the DMA is enabled.

The "sd_lld_init()" function is called inside another function, which is "componentsInit()". Initialization functions are called only once in the application and for all the UART drivers.

4.0.7 UART starting function

Expressed as the "sd_lld_start (SerialDriver *sdp, const SerialConfig *config)" function in the bare-metal application. The "sd_lld_start" function receives the following arguments

- A pointer to the targeted UART driver object.
- The corresponding configurations.

The UART start function do as follows:

- Assign the selected configurations to each of the corresponding UART drivers, for example, the baud rate, and if the DMA is enabled or not.
- Access the registers which control the operating mode and set it according to the selected operating mode in the configurations
- Calibrate the UART clock

Start functions are called for each of the enabled UART driver only once in the application.

4.0.8 UART transmission / receiving call back functions

One of the most important advantages of the DMA is that it allows the transfer of the data without using the CPU, consequently, this makes the data transfer simpler and let the CPU more available for other operations, however, the DMA uses an interrupt just for notifying the CPU that the data transmission / receiving has been done, and since an interrupt is used a call back function must be introduced to handle it. UART transmission call-back function is Expressed as the "sddmatxcb(SerialDriver *sdp)" function in the bare-metal application, while, UART receiving call-back function is Expressed as the "sddmarxcb(SerialDriver *sdp)" function in the bare-metal application. The "sddmatxcb" and "sddmarxcb" functions receives only the pointer to the targeted CAN driver as an argument.

4.0.9 UART reading function

UART is a serial communication protocol, which allows a microcontroller to be connected to many other devices. In this study, this device is a laptop operated by a "windows" operating system. To set up a UART communication with a "windows" operated device, it is needed to use software that emulates the terminal and provides a proper user interface that displays the received messages from the UART and facilitates sending messages to the UART e.g., "PuTTy". UART reading function is Expressed as the "sd_lld_read(SerialDriver* sdp, uint8_t* buffer, uint16_t len)" function in the bare-metal application. UART reading function reads the message transmitted by the device, i.e., The laptop. The "sd_lld_read" function receives the following arguments

- A pointer to the targeted UART driver object.
- The data buffer receiver
- The number of bytes received

4.0.10 UART writing function

UART writing function is Expressed as the "sd_lld_write(SerialDriver* sdp, uint8_t* buffer, uint16_t len)" function in the bare-metal application. UART writing function transmit a message from the MCU to the device, i.e., The laptop, usually this message is going to be displayed on the user interface of the used software. The "sd_lld_read" function receives the following arguments

- A pointer to the targeted UART driver object.
- The data buffer receiver

- The number of bytes received

Figure 4.1: a message of number zero transmitted from the MCU to the "windows" device.

4.0.11 Operating mode's control function

This function allows controlling the operating mode of the CAN through the UART communication, consequently, the CAN operating mode can be switch between the three main operating modes (i.e., external loopback, internal loopback, and no loopback) through the runtime, otherwise, to change the operating mode it is needed to stop the program, modify the configurations and start the program again from the scratch, which is not a reliable way to develop. The "UART_LBCK" function receives the following arguments

- A pointer to the targeted CAN driver object.
- The address of the UART data buffer receiver

The UART controls the CAN operating mode as follows

- If the transmitted message is 0, the CAN is going to operate in the normal operating mode.

- If the transmitted message is 1, the CAN is going to operate in the external loopback operating mode.
- If the transmitted message is 2, the CAN is going to operate in the internal operating mode.
- otherwise, the UART has no effect on the CAN.

4.0.12 The comparing functions

The comparing function is a function that receives two vectors as inputs, compares every two elements in them, and returns different values depending on the similarity between the values of the two passed vectors. The comparing function is expressed as "msg_cmp" function in the bare-metal application. In this study, the comparing function is used to compare the transmitted and the received frames from the same CAN subsystem in the external and internal loopback operating modes, while in the normal operating mode, the received message is compared to pre-defined values. This method allows not just checking the transmission and the receiving of the messages through the CAN network, but also, it provides a way to check if the content of these messages is transmitted and received correctly without any corruption. "msg_cmp" function receives the following arguments

- The identifier of the transmitted frame in case of loopback operation or the identifier of the reference frame in case of normal operation
- The identifier of the received frame
- The number of messages to be compared in the message array vector of the CAN frame.

4.0.13 The function that controls the blinking of the LEDs

The function that controls the blinking of the LEDs is expressed as "pal_lld_togglepad(port, pin)" function in the bare-metal application. "pal_lld_togglepad" function receives the following arguments

- The port at which the pin which controls the LED is located
- The pin to which the LED is connected.

The ports and the details of the pins are illustrated in chapter 3

4.0.14 Algorithm

The bare-metal application that checks the functionality of the whole system is based on a loop that keeps transmitting two different messages on the bus, one message holds a standard identifier and the other holds an extended identifier. Each of these messages is composed of two parts:

1. 4 Bytes which contains a fixed message that doesn't change with every cycle in the loop
2. 4 Bytes which contains a counter that increments by 1 with every cycle in the loop, this part is useful to make sure that the updated version of the message is being transmitted, not only the first one.

Every CAN subsystem connected to the bus is called a "node". In this study, we have two nodes, node A and node B. Node A is transmitting and receiving these two messages while node B is just a receiver. Whenever a message is received by any of these nodes the LED is going to blink once. Suppose that the LED connected to node A is called LED A and the LED connected to node B is called LED B. In this study, SPC58NN84C3 works in three different operating modes which are:

- Internal loopback
- External loopback
- No loopback

Table 4.1 describes the state of each LED in the different operating modes.

Operating mode	LED A	LED B
Internal loopback	Not blinking	Not blinking
External loopback	blinking	blinking
NO loopback	Not blinking	blinking

Table 4.1: LEDs Statues in the different operating modes

The algorithm which is implemented in the bare-metal application is as follows

- Start and initialize the CAN

- Start and initialize the CAN
- Define the to-be-sent frame
- Assign the message to the to-be-sent frame
- Infinite for loop
  - Update the to-be-sent frame with the updated value of the counter
  - Check the UART messages received
    - If a new message contains the value "Zero", the operating mode will be "No loopback"
    - If a new message contains the value "One", the operating mode will be "External loopback"
    - If a new message contains the value "Two", the operating mode will be "Internal loopback"
    - If there are no new messages received, The operating mode will remain unchanged
  - Transmit the messages from node A on the bus
  - Check the messages received by the CAN node A and the CAN node B, if the received message contains an ID that is allowed to be received by the filter, the corresponding LED will blink and the message will be shown on the debugger and the comparing variable will turn into 1, otherwise, nothing will happen and the comparing variable will be zero
  - The counter will be incremented by 1
  - Repeat

Figure 4.2 is a flow chart that explains the algorithm of the bare-metal application

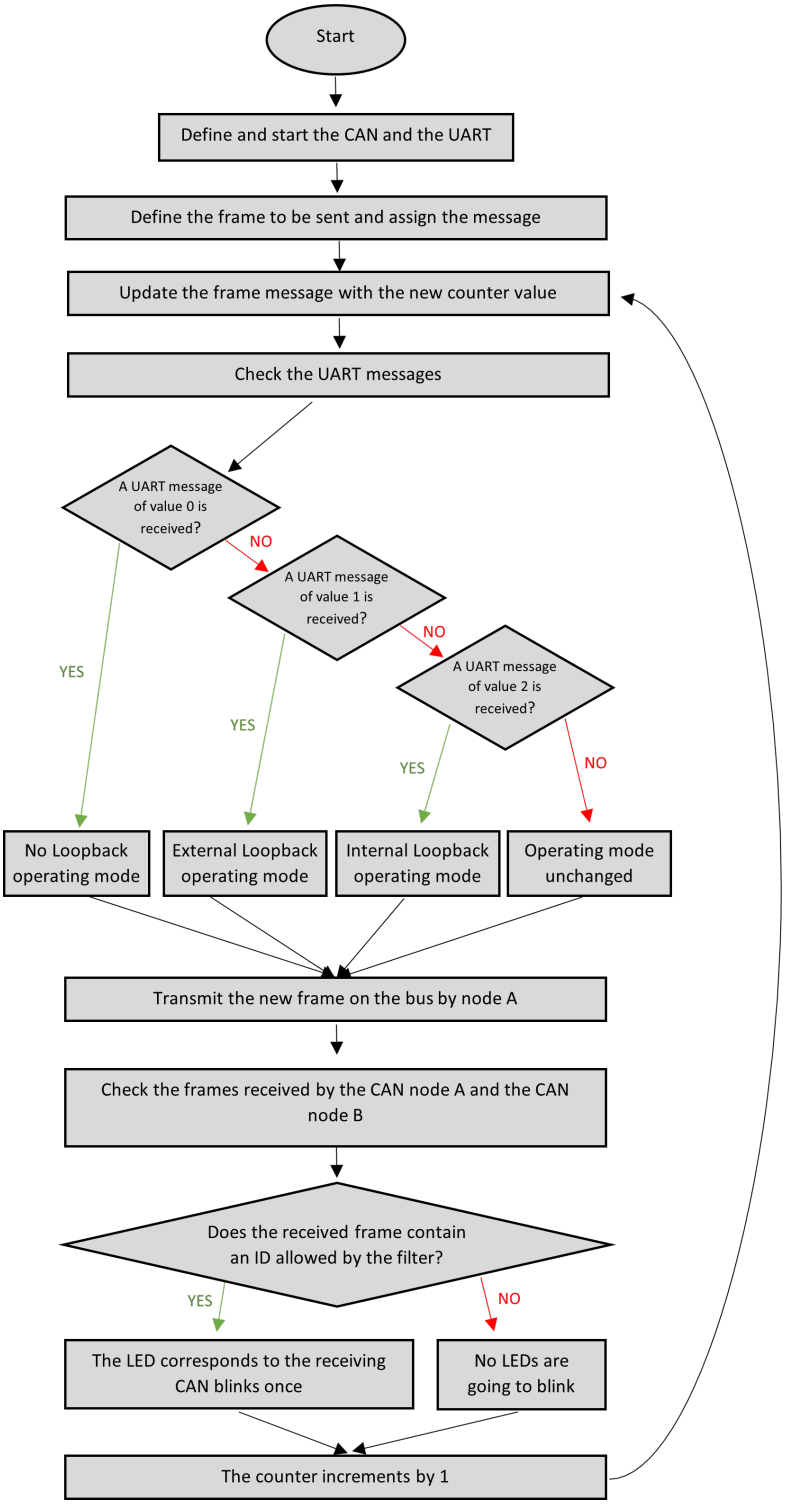

Figure 4.2: The algorithm flowchart

4.0.15 First Step into the MICRIUM OS application

Hardware abstraction is one of the essentials that should be present in the OS-operated MCUs, it is considered the connection between the OS and the various MCU types. The hardware abstraction can be implemented in Micrium by defining the basic six functions which build the driver layer. For the hardware abstraction between SPC58NN84X and Micrium OS, these functions are listed and explained in table 4.2, while table 4.3 explains the passed arguments to these functions.

Function	Explanation
SPC58NN84X_CAN_Init	This function is responsible for initializing the CAN controller that has been chosen based on the name of its bus device. This function should also clear all message buffers (if any are available) and put the CAN controller in an off state.
SPC58NN84X_CAN_Open	This function identifies the specified CAN device as used, thus locking the device. The value that is returned is the identification of the device, and it is necessary to make use of this identifier for all subsequent activities involving this device.
SPC58NN84X_CAN_Close	This function frees up the specified CAN device so that it can be utilized again; more specifically, it disables the device lock
SPC58NN84X_CAN_IoCtle	This function exerts control over the CAN device that has been provided. The control operation that the user wants to do can be defined by the parameter function. lock
SPC58NN84X_CAN_Read	The most recent CAN frame that was received from the CAN controller is read by this function.
SPC58NN84X_CAN_Write	A CAN frame is written into the CAN controller in preparation for transmission via this function.

Table 4.2: Micrium OS basic CAN functions explained

Function	Explanation
SPC58NN84X_CAN_Init	• Bus device name
SPC58NN84X_CAN_Open	• the name of the bus node that must be utilized by the interrupt function in order to gain access to the CAN bus layer (not used by now) • device name that identifies the device that is contained within the controller • the operating mode used i.e., External, internal, and no loopback modes
SPC58NN84X_CAN_Close	• device identifier, returned by SPC58NN84X$_{C}AN_{O}pen$
SPC58NN84X_CAN_IoCtle	• device identifier, returned by SPC58NN84X_CAN_Open • function code that defines the operation that will take place. The values of this parameter are CAN_FRAME_TX, CAN_FRAME_RX, CANFD_FRAME_TX,, CANFD_FRAME_RX • optional functional argument so far is implemented only for the transmission and receiving process and in this case, this value is the address of the frame to be received or the frame to-be-transmitted

SPC58NN84X_CAN_Read	• device identifier, returned by SPC58NN84X_CAN_Open • Pointer to CAN frame • Size of buffer
SPC58NN84X_CAN_Write	The same arguments as SPC58NN84X_CAN_Read

Table 4.3: Micrium OS basic CAN functions arguments

Note: These functions are not completely ready to be used with Micrium OS yet, they still need some additional developments to completely follow the standards of the OS, however, this development so far can be considered as the cornerstone of the hardware abstraction layer. These functions can replace the main functions in the bare-metal application, however, The main difference is that the arguments passed to these functions follows the standard of Micrium OS.

Chapter 5

Experimental results

This section includes the experimental results of operating the test unit in a complete CAN communication. The results of testing the can communication could be tracked in three different ways which are:

- Tracking the LEDs blinking on the test unit
- Tracking the logic analyzer
- Tracking the debagger connected to the test unit

The results shown on each of these tracking methods will be explained in the three operating modes which are:

- Internal loopback operating mode
- External loopback operating mode
- No loopback operating mode

5.0.1 Results on SPC58NN84x HW

This tracking method shows only if the complete communication is performed correctly and the CAN node is receiving messages which match the filter, it is a fast way to show if there is a problem that hinders the communication, however, it cannot help detect this problem.

Internal loopback case

In the case of the internal loopback, the message transmitted doesn't go from the CAN controller to the CAN bus, it doesn't even reach the pins or the CAN

transceiver, the transmitted message is internally from the transmission part to the receiving part inside the CAN controller. The internal loopback is used only to test the connection between the interface and the internal chip is working properly, so, in this test since we have no transmitted messages on the CAN bus, no LEDs will blink. Figure 5.1 shows that there are no blinking LEDs in case of internal loopback

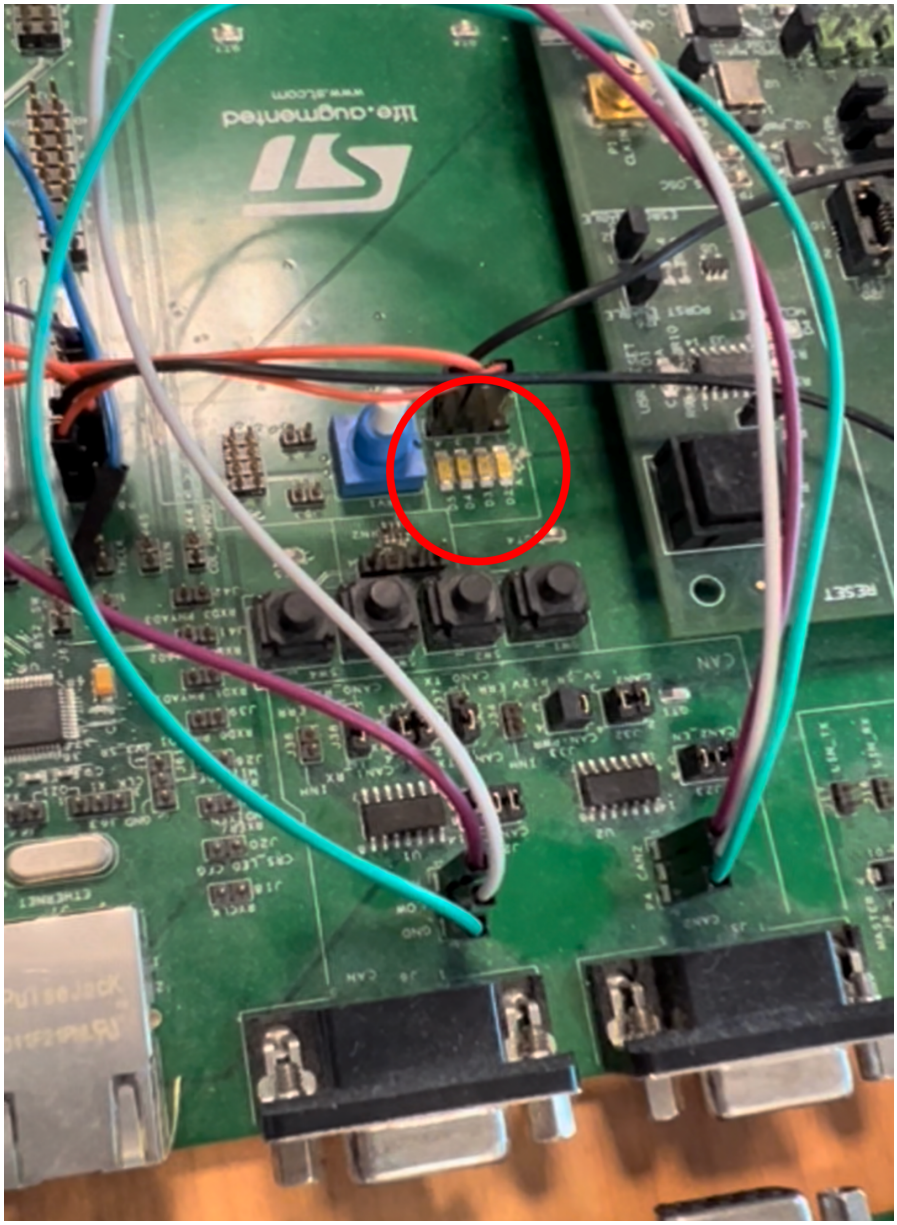

Figure 5.1: HW results in case of internal loopback

External loopback case

In case of the external loopback the message transmitted goes from the CAN controller to the CAN bus, then this transmitted message is received by both nodes A and B, and if it matches the filter of both, the corresponding LED will blink, so, in this test since both filters contains the transmitted message ID, both LEDs will blink. Figure 5.2 shows the blinking of both LEDs in case of external loopback.

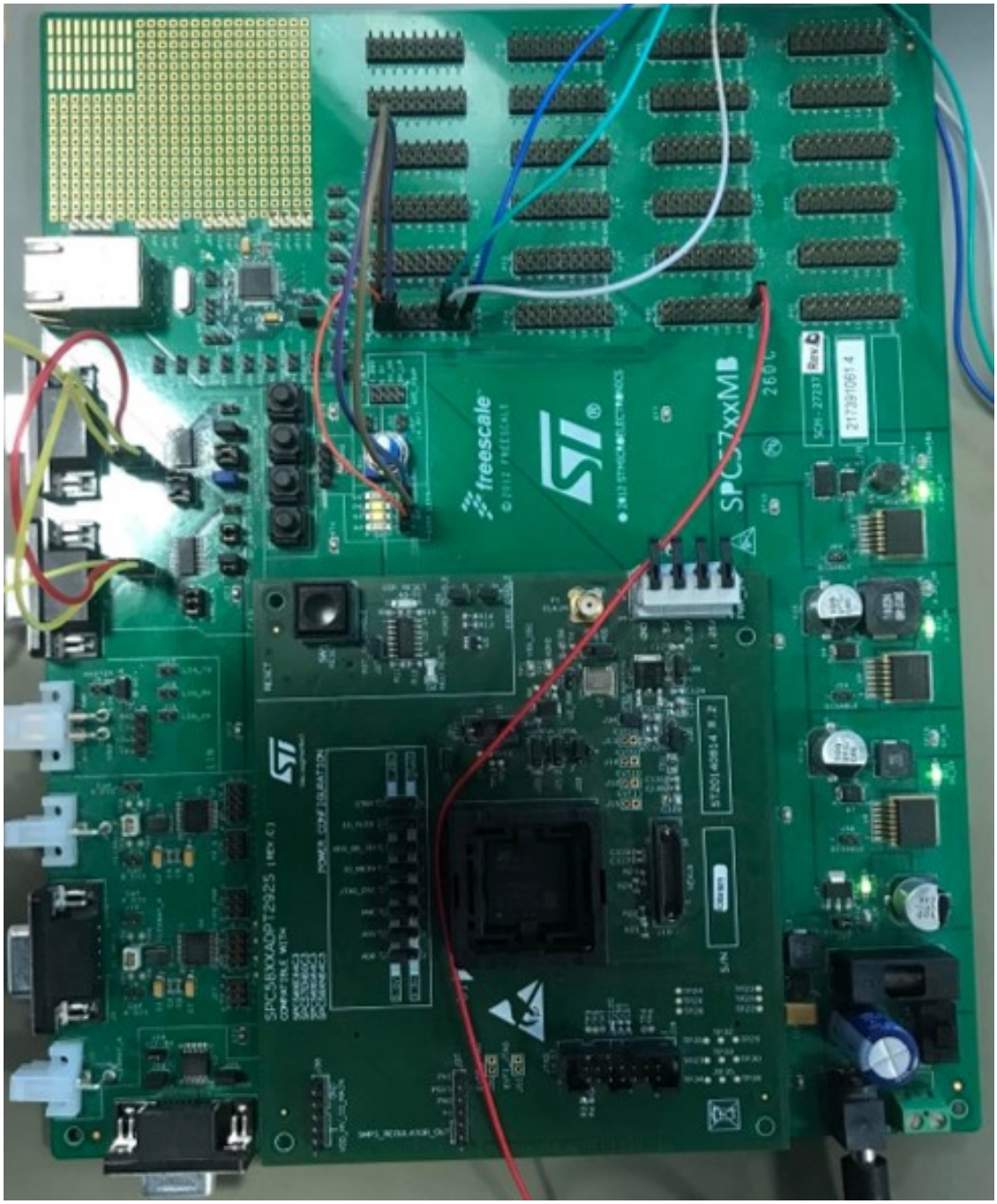

Figure 5.2: HW results in case of external loopback

No loopback case

No loopback is the normal operating condition in which the message transmitted goes from the CAN controller to the CAN bus, then This transmitted message is received by node B only, and if it matches the filter of node B, the corresponding LED will blink, so, in this test since the filter of node B contains the transmitted message ID, the LED corresponds to LED B will blink. Since the test unit contains two independent CAN subsystems, one of them may operate only for receiving as node B, while the other one may operate for the transmission as node A as shown in figure 5.3.

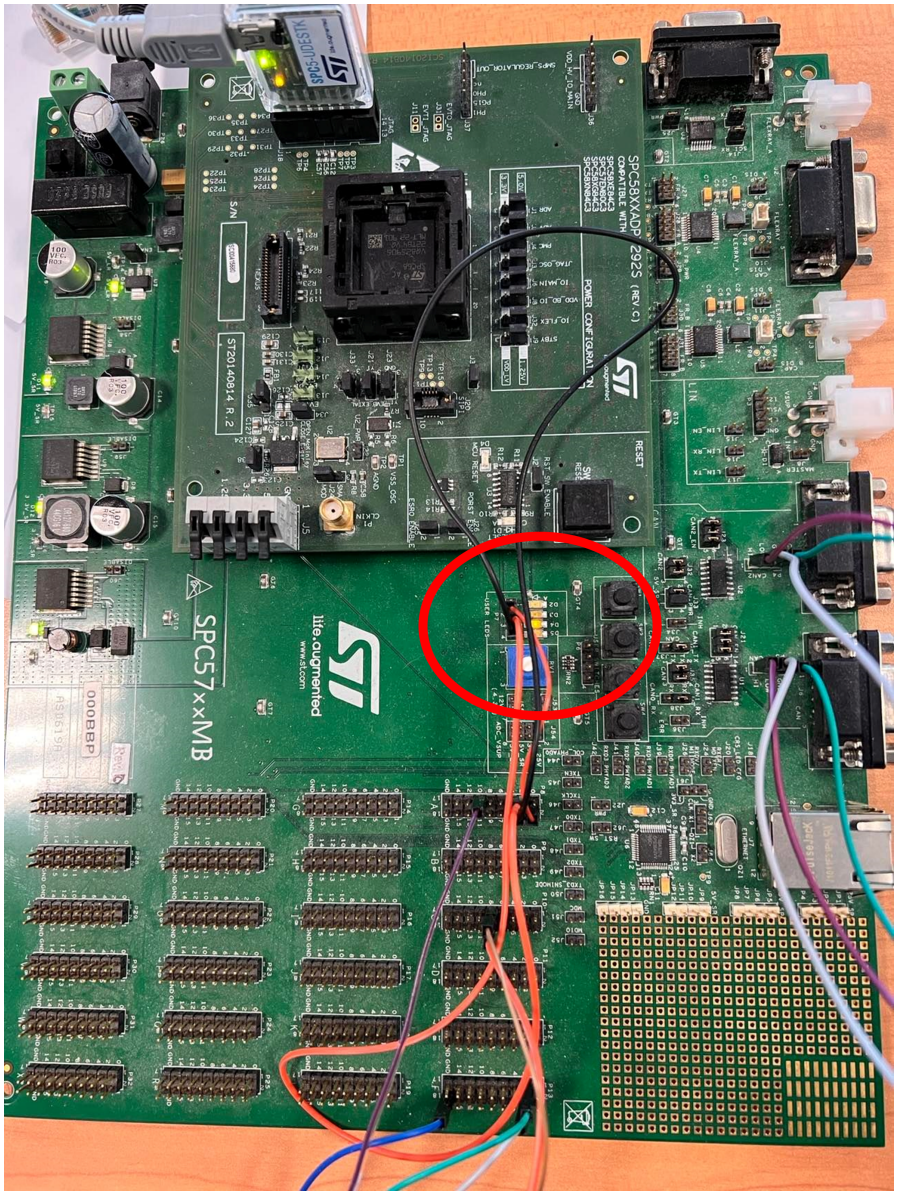

Figure 5.3: HW results in case of No loopback

Also, node B can be another test unit as shown in figure 5.4.

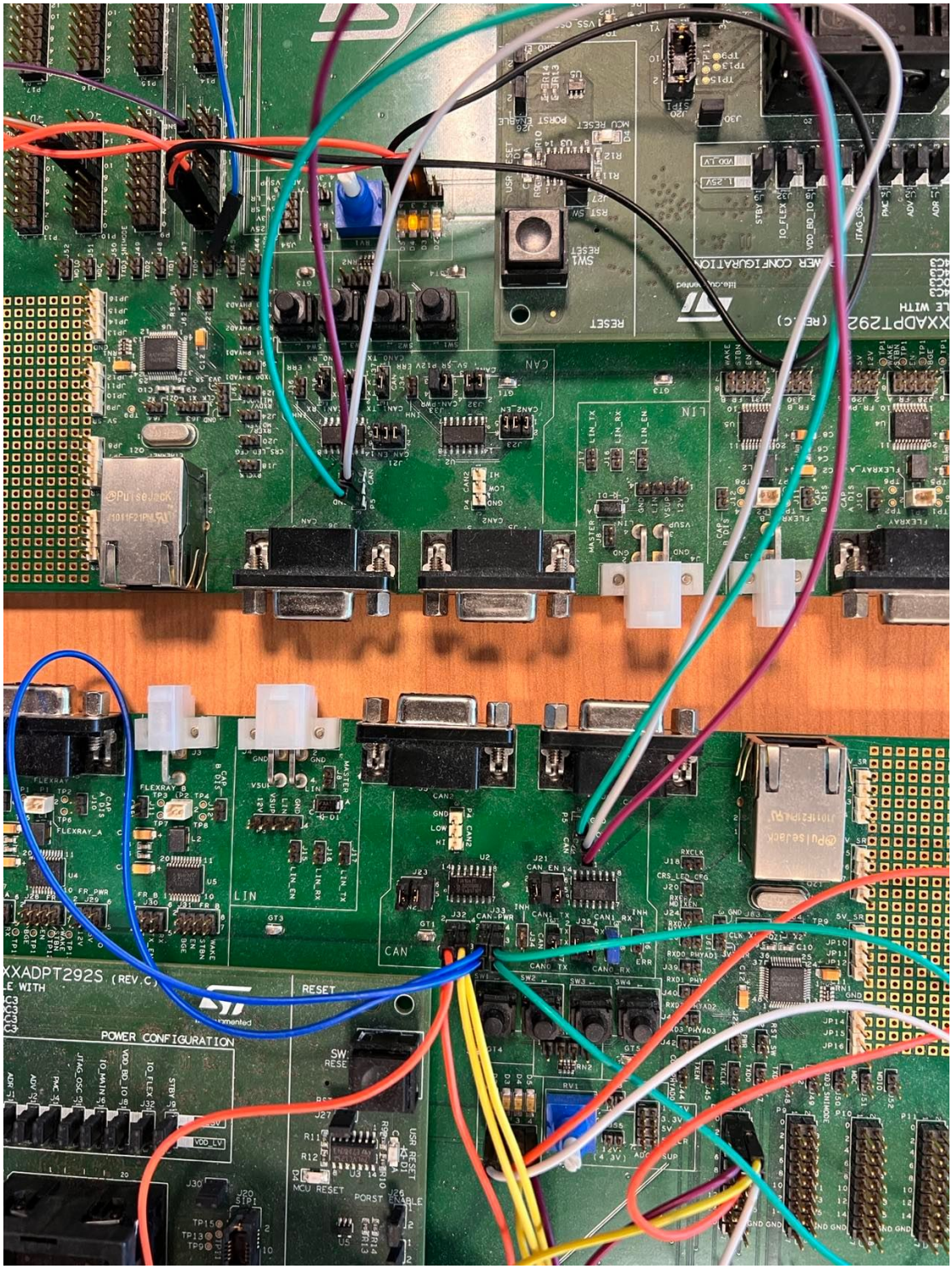

Figure 5.4: HW results in case of No loopback using 2 connected test units

5.0.2 Results of the logic analyzer

The logic analyzer is a very helpful tracking and troubleshooting method, unlike the LEDs tracking, the logic analyzer can be used for troubleshooting, it is able to display the details of the communication and the message content in terms of bits or bytes. The results shown in this section are for a CAN communication at a baud rate of 500bit/s

Figure 5.5 shows how the signals look like on the logic analyzer, the CAN communication signals are shown as multiple lines separated from each other, and going closer inside these lines, the structure of the signal becomes more and more clear.

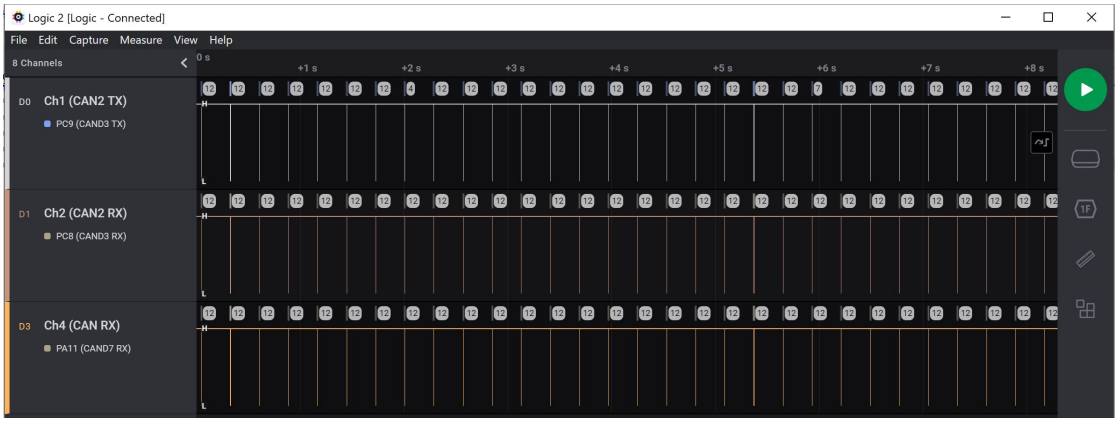

Figure 5.5: frames transmission and receiving signals on logic analyzer

the structure of the transmitted or the received CAN frames can be noticed in figure 5.6, they are composed of a sequence of bits, and every bit has a specific significance labeled above the bit and explained in the previous sections of this study. The message assigned to each frame can be visualized as a sequence of bytes labeled by the content of each byte.

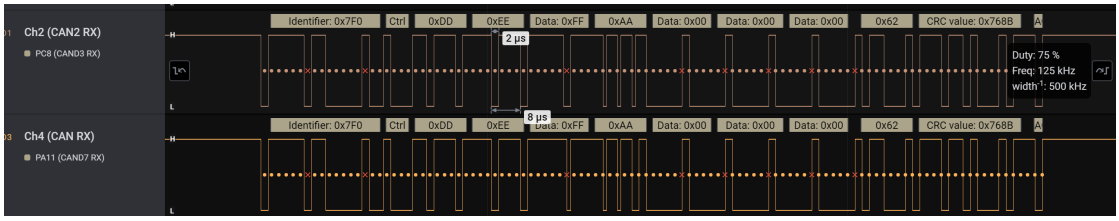

Figure 5.6: The composition of the transferred and received signals on the logic analyzer

The logic analyzer allows also tracking the time between each two sent or received

frames which is around 280ms as shown in figure 5.7, also, if needed, the time to receive every bit or a group of bits is shown in figure 5.7

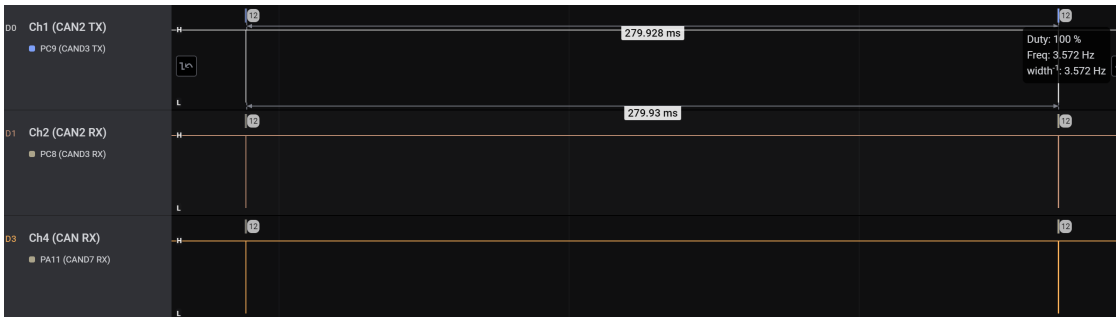

Figure 5.7: the time between two signals on the logic analyzer

5.0.3 Results the Debugger

debugger is a special tool that is connected to the test unit to flash the program, debug it, show the result and the registers states as well as many other uses. The debugger allows tracking if the sent message is the same one received, also it allows checking every single value of the registers or the bits of the frames sent or received and this can be done step by step. Figure 5.8 shows the state of the debugger in case of external loopback, it can be noticed that the payload of the transmitted frame is updated and identical to the payloads received by both CAN nodes A and B, which means that the CAN communication is working properly.

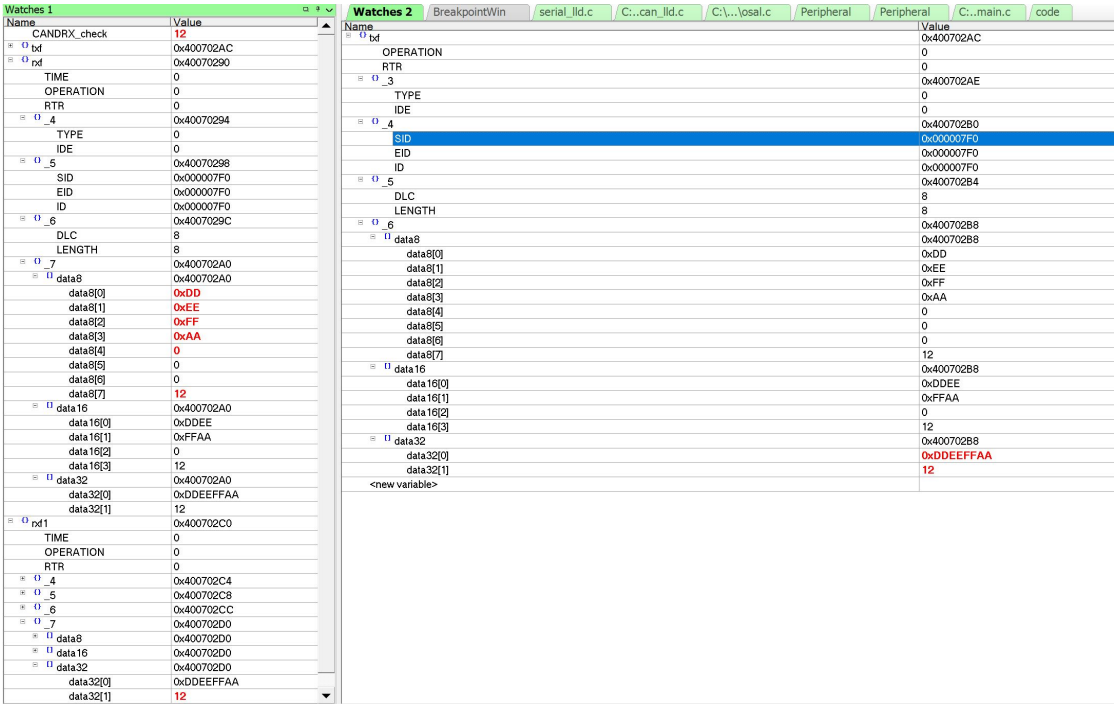

Figure 5.8: The results on the debugger

as a summary:

Evaluation Method	**Internal Loopback**	**External Loopback**	**No Loopback**
LEDs	Functioning correctly	Functioning correctly	Not functioning
Logic Analyzer	Correct signal patterns	Correct signal patterns	No signal
Debugger	No errors	No errors	Errors detected

Table 5.1: Summary of Experimental Results

Chapter 6

Conclusion

Embedded systems is the corner stone of the technological renaissance we are witnessing today, however, the complexity of programming the embedded systems is growing dramatically along with the number of program lines of the source code programmed in these embedded systems, so that, the operating systems has made a perfect solution through raising an abstraction level to facilitate the embedded systems programming.

This thesis is a part of a multi-year project aiming to completely test STM micro-controller "SPC58NN84C" and to raise Micrium operating system on it, while this thesis concerns about setting up the micro-controller for a UART-controlled test, testing the communication of the CAN peripheral, coding a bare metal application which runs this test, verify the results using observation, debugger and a logic analyzer and implementing the main functions on which the abstraction layer of the operating system is built.

The "SPC58NN84C" is a just micro-controller which must be connected to an adaptor like " SPC58XXADPT292S" and a mother board like " SPC57XXMB" which altogether are providing utilizable access to "SPC58NN84C" peripherals and building the a suitable environment for the test. The details of these connections is completely covered in this thesis work both by graphical and written explanations, as well as, the graphical and written explanation of all auxiliary tools connectivity e.g. The UART module and the logic analyzer

The CAN test is based on bare-metal application which is based on five main functions which are CAN initialize, open, received message reading, message transmission and the CAN stop, in addition to the auxiliary functions which configure the MCU, start and control UART ,and enhance the debugging process.

The abstraction layer of Micrium operating system is based mainly on five basic functions as in bare-metal application, however, they have different configurations and arguments. In addition, the abstraction layer has an extra function which is able to control and manipulate the transmitted and received messages by the

CAN. The arguments and the way of using the bare-metal application and the abstraction layer functions are illustrated in details in this thesis which could be considered as a reliable documentation for any further testing or developing study on the CAN peripherals in the future.

The CAN communication in the bare-metal application is programmed to switch between three different operating modes according to the input signal of the UART which are the normal operating mode, external loop back mode ,and internal loop back mode. The results of the testing process in these operating modes could be covered in details using the combination of three different methods, which are observing the hardware, using the debugger, and using the logic analyzer.

This thesis study paves the way for building a Micrium OS operated control over the CAN peripheral in "SPC58NN84C" MCU, so that, further developments will be required to achieve this goal, consequently, through more developments the "SPC58NN84C will eventually be a Micrium OS controlled MCU that could be developed and used by programmers easily and more efficient.

Chapter 7

References

[1] Texas Instruments. (2002, August). Introduction to the Controller Area Network (CAN). In http://masters.donntu.ru/2005/fvti/trofunenko/library/sloa101.pdf

[2] Texas Instruments. (2015, January). Compact CAN-to-Ethernet Converter Using 32-Bit ARM® CortexTM-M4F MCU. In https://www.ti.com/lit/ug/tidu706a/tidu706a.p

[3] Reference manual of SPC58NN84C3
https://www.st.com

[4] The user manual of SPC58XXADPT292S mini-module
https://www.st.com

[5] The user manual of SPC57XXMB
https://www.st.com

[6] FT232R USB UART IC Data sheet
https://www.ftdichip.com

[7] CAN, "Controller Area Network",[Online]
Available: http://www.ti.com

[8] CAN, "A simple Intro",[Online]
Available: https://www.csselectronics.com

[9] SPC584Nx, SPC58ENx and SPC58NNx Data sheet

[10] SPC57XXMB Data brief