POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

Real-Time Control and Communication in Hybrid Vehicle Power Trains

Supervisors

Prof. Sarah AZIMI

Prof. Luca STERPONE

Candidate

Kevin GIORDANO

July 2024

Summary

The electrification of vehicles is advancing rapidly, with DC-DC converters playing a central role in their electrical architecture. Semiconductor technologies such as gallium nitride (GaN) MOSFETs enable high-power, high-temperature operation, enhancing vehicle performance. However, these advancements necessitate improvements in microcontrollers managing these components, including faster computation, advanced pulse-width modulation (PWM) signal generation, and real-time synchronization. This thesis analyses the AURIX TC39 microcontroller in order to develop software for managing a DC-DC converter in a hybrid vehicle architecture. This architecture comprises one inverter connected to the electric motor and two DC-DC converters, one of which interfaces with a battery and the other with a fuel cell. The research starts by implementing the Universal Asynchronous Receiver-Transmitter (UART) communication, which is crucial for the inverter to send reference currents to the DC-DC converters. The Direct Memory Access (DMA) is employed for the purpose of facilitating the efficient transfer of data within the system. To guarantee the reliability of the communication process, a synchronisation mechanism has been implemented to discard messages in the event of a transmission failure. This approach not only enhances the performance of the system by reducing the load on the CPU during data transfer operations, but also ensures the integrity and reliability of communications by preventing the propagation of corrupted or incomplete data. A further aspect of the communication system between the three microcontrollers is the use of Controller Area Network (CAN). This protocol is widely used in the automotive field due to its high degree of reliability. In the subsequent system, CAN is employed to facilitate the exchange of diagnostic, error and status messages between the electrical components within the architecture. In the following, the behaviour of the AURIX TC39 microcontroller

in external input clock mode was analysed by studying its internal modules that deal with clock generation and distribution. The analysis that follows was carried out with a view to the future, as three AURIX TC39 micro controllers are expected to share the same clock source to optimise their synchronisation. Furthermore, the Generic Timer Module (GTM) is employed for the generation of PWM signals and the determination of sampling instants in order to manage the switching poles of the DC-DC converter. This is achieved using the Timer Output Module (TOM). To improve the correct behaviour of the components inside the electrical architecture, the counters responsible for generating the PWM signals are synchronised to the inverter component by means of a reset signal. The implementation of dead time and the optimization of synchronisation between microcontrollers are achieved through the use of the Dead Time Module (DTM) and Timer Input Module (TIM). As a result, the Generic Timer Module (GTM) within the AURIX TC39 has proven to be an optimal component for the developed applications, offering high-speed signal generation and synchronisation features to enhance the behaviour of the system in automotive applications. Finally, the developed control algorithm was tested on the microcontroller to verify compliance with the execution time constraints imposed by the requirements, thus ensuring efficient management of the DC-DC converter within the electrical system of the vehicle.

Table of Contents

List of Figures

Chapter 1

Introduction

Currently, DC-DC converters play a fundamental role in vehicular electrical architecture. Thanks to advancements in semiconductor component technologies, the complete electrification of vehicles is becoming increasingly feasible. In particular, the use of GaN (gallium nitride) or SiC (silicon carbide) MOSFETs is becoming more widespread in vehicular components, as these materials allow operation at higher power levels and temperatures, ensuring excellent performance.
This development necessitates significant improvements in the micro-controllers managing these components, requiring high computational speeds, elevated sampling frequencies, advanced PWM signal generation capabilities, synchronization among micro-controllers and highly deterministic real-time software behaviour. On the market, there are various micro-controllers suited for such applications, thanks to their multi-core architectures that allow increasingly complex software to be executed while optimizing execution times. Among these, the AURIX TC39 by Infineon and the SPC5 family by STMicroelectronics stand out, widely used in engine, transmission, body and chassis control.
In this context, this thesis focused on the analysis of the AURIX TC39 microcontroller to develop management software for a DC-DC converter used in an architecture consisting of an inverter and two DC-DC converters: one managing power flows to a battery and the other managing power flows to a fuel cell, all centred in a hybrid vehicle.

The proposed configuration is depicted in Figure 1.1. Each electrical component is managed by an AURIX TC39. The first step of the research was to implement

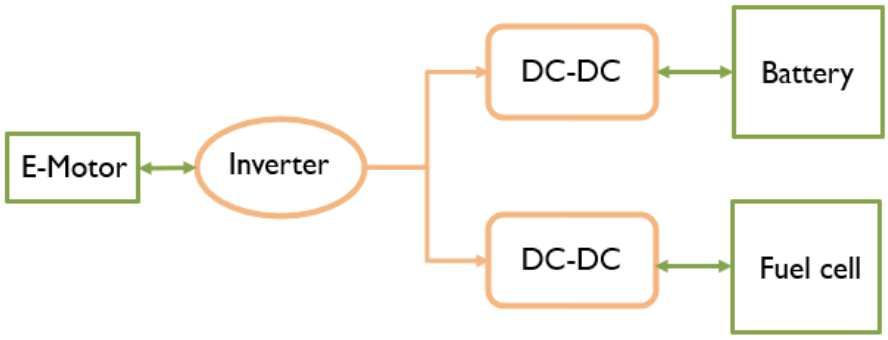

Figure 1.1: Hybrid Vehicle Architecture.

Universal Asynchronous Receiver-Transmitter (UART) communication by exploiting the ASCLIN module in the micro-controller. This communication is critical for the management of the DC-DC converters, as it is used by the inverter to send reference currents to the DC-DC converters every 50 µs. For the implementation of the UART communication, the Direct Memory Access (DMA) mechanism was used to transfer the received data from the receive FIFO memory to the destination buffer, thus optimizing the CPU load. In addition, a synchronization mechanism between transmitter and receiver was implemented to make the system more robust in case of communication interruptions.
Additionally, the Controller Area Network (CAN) was studied and implemented on the same micro-controller, managed by the MCMCAN module. This CAN network is utilized by the three components to send diagnostic messages, enhancing communication reliability and integration within the vehicle's network.
In this context, both modules ASCLIN and MCMCAN are configured to guarantee an efficient communication management. The two communication networks were tested, simulating messages exchange between AURIX TC39 modules, verifying that the communication requirements are fulfilled in terms of Baud rate and correctness acquisition and transmission of the messages.

The next step was to analyse the behaviour of the microcontroller in external input clock mode, with the future aim of implementing a configuration where the three micro-controllers share the same clock to optimise their synchronisation, as shown in Figure 1.2. For this analysis, both the clock acquisition system and the system clock generation mechanism were examined to see if the microcontroller was capable of generating a correct system clock of 300MHz from an external clock of 8MHz. However, the analysis showed that the minimum required input clock frequency was 10MHz. Consequently, the study was adapted to take this new frequency into account, which introduced a new requirement and led to a revision of the previous requirements.

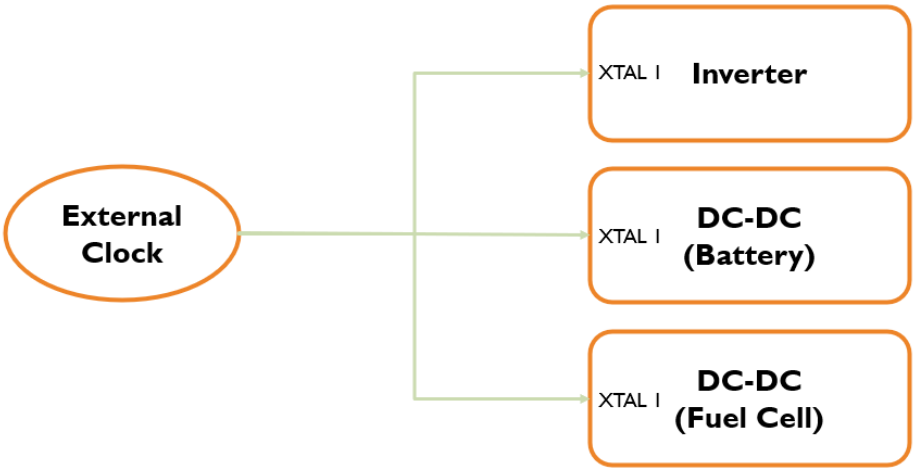

Figure 1.2: Clock Sharing configuration between the three AURIX TC39s

The Generic Timer Module (GTM) was employed to generate the eight pulse-width modulation (PWM) signals required to manage the eight metal-oxide semiconductor field-effect transistors (MOSFETs) at a frequency of 100 kHz.

Furthermore, the Enhanced Versatile Analog-to-Digital Converter module was enabled to determine the sampling instances performed by the Timer Output Module (TOM) sub module, exploiting the cascading connection of TOM channels. Additionally, the implementation of dead time in the eight PWM signals was achieved through the use of the Dead Time Module (DTM). In order to further optimise the synchronisation among the three micro-controllers and to prevent phase shift phenomena between the TOM counters, a configuration was implemented where the inverter sends a reset signal to the GTM counters of the micro-controllers managing the DC-DC converters. In order to achieve this, the Timer Input Module (TIM) sub module was employed, which acquires the reset signal and transmits it via hardware to the TOM counters, thereby ensuring more deterministic system behaviour.

As a result, the GTM results as a suitable module to be used for a real time management for PWM generation for power structure control task, thanks to the TOM architecture which allows synchronization between the counters inside the module.

In conclusion, the developed control task was implemented in the AURIX TC39, which is executed at a frequency of 100 kHz. Its function is to calculate duty cycles for the PWMs generated in the successive period based on the reference currents received via UART and on the samplings performed. In addition, it is necessary to ensure that the duty cycles are updated synchronously and independently for each counter at the time they are reset. Consequently, the Sub Timer Output Module (TOM) was identified as an appropriate solution for the described application, as it is capable of fulfilling all the necessary requirements.

Finally, it was demonstrated that the AURIX TC39 microcontroller is capable of executing the tasks required within the imposed dead times, ensuring deterministic behaviour, thus proving suitable for real-time applications.

Chapter 2

State of art

2.1 PWM definition

Pulse Width Modulation (PWM) is a method frequently employed to generate an analogue output signal from a digital input. This method involves a digital signal that alternates between two logical values, 1 and 0, which typically correspond to 0V and 5V. As illustrated in Figure 2.1 the principal attribute of PWM is the capacity to establish an equivalent mean value by adjusting the percentage of time the signal remains ON throughout the entire period, which is also known as the duty cycle.

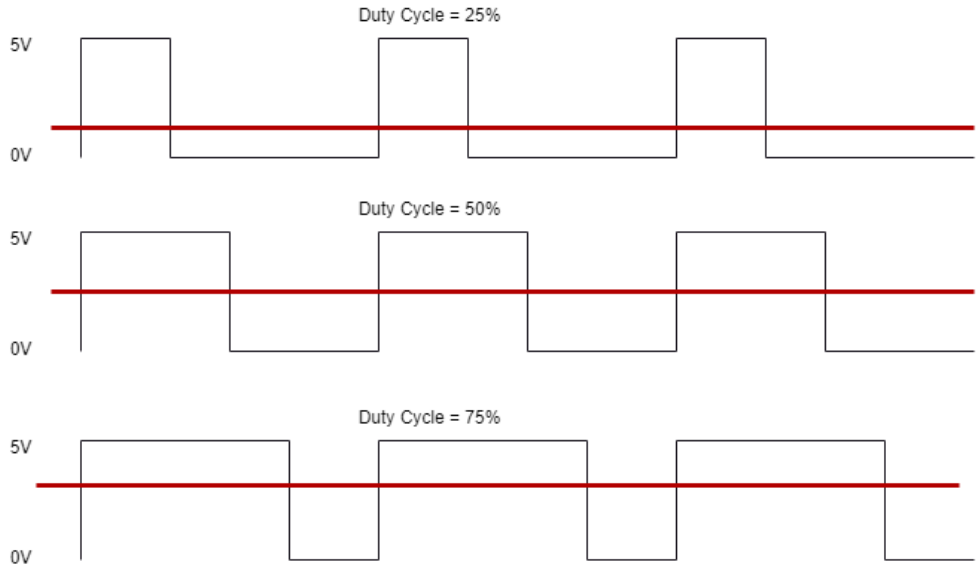

Figure 2.1: Hybrid Vehicle Architecture.

The definition of the duty cycle is expressed by the following equation:

$$DC = \frac{Ton}{Ttot} * 100$$

.

In this context, the variable Ton represents the time at which the signal level is at a value of 1, while Ttot denotes the total period. Due to its inherent properties, PWM is frequently employed in micro-controllers to control the operation of electrical components such as MOSFETs, thereby facilitating the management of power flows.

2.2 PWM modulations

Depending on the type of application, different PWM modulation techniques can be used, each with specific characteristics determined by the counter configuration for signal generation, the need for signal symmetry, and the efficiency required by the controlled system.

2.2.1 Symmetrical PWM

Symmetrical PWM modulation uses a triangular signal as a carrier to set the period of the PWM. The duty cycle is defined by a threshold typically set by a compare register of the timer module. This approach produces a PWM signal that is symmetrical with respect to the center of the period, as shown Figure 2.2. With this modulation, a center-aligned series of multiple PWMs can be generated if all counters are in phase and with the same period.

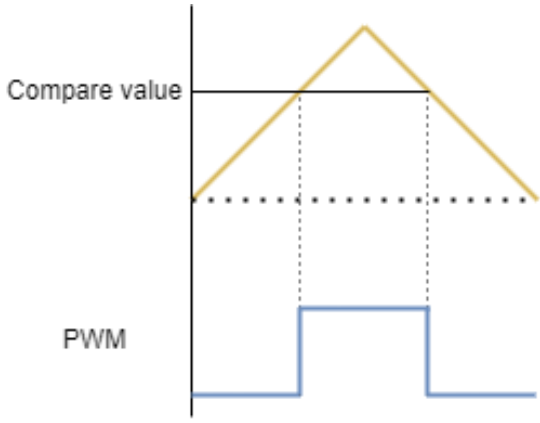

Figure 2.2: Symmetrical PWM.

2.2.2 Asymmetrical PWM

Asymmetric PWM modulation, uses a saw tooth signal as a carrier to define the period. The duty cycle is determined through a threshold value, as illustrated in Figure 2.3. This technique allows for multiple edge-aligned PWMs, which can be left-aligned or right-aligned if the counters generating them are configured with the same period and are in phase with each other.

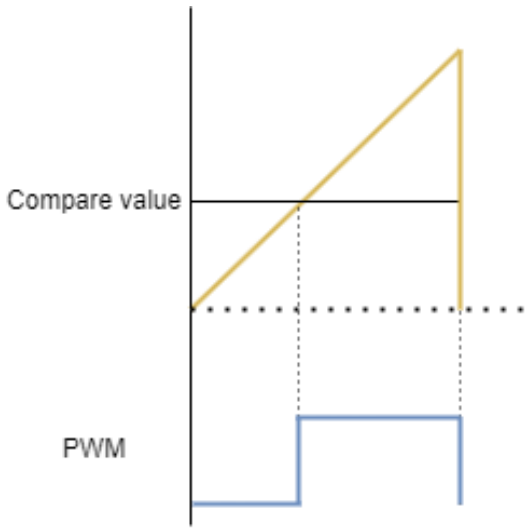

Figure 2.3: Asymmetrical PWM.

2.2.3 Synusoidal PWM

Sinusoidal PWM modulation (SPWM) is a technique based on a sinusoidal modulating signal, widely used in industrial [1]and alternate current motor applications despite limitations due to low switching frequencies that cause low order harmonics in the output signal[2]. In this modulating technique, a sinusoidal modulating signal *Vref* is compared with a triangular carrier *Vc*.

The frequency of the output signal produced depends on the frequency of the modulating signal vref, while the frequency of the triangular carrier signal determines the duty cycle of the output signal[3]. The switching state of the output signal is shown if Figure 2.4 and is defined according to the following rules:

- $V_{out} = 1$ when $V_{ref} > V_c$
- $V_{out} = 0$ when $V_{ref} < V_c$

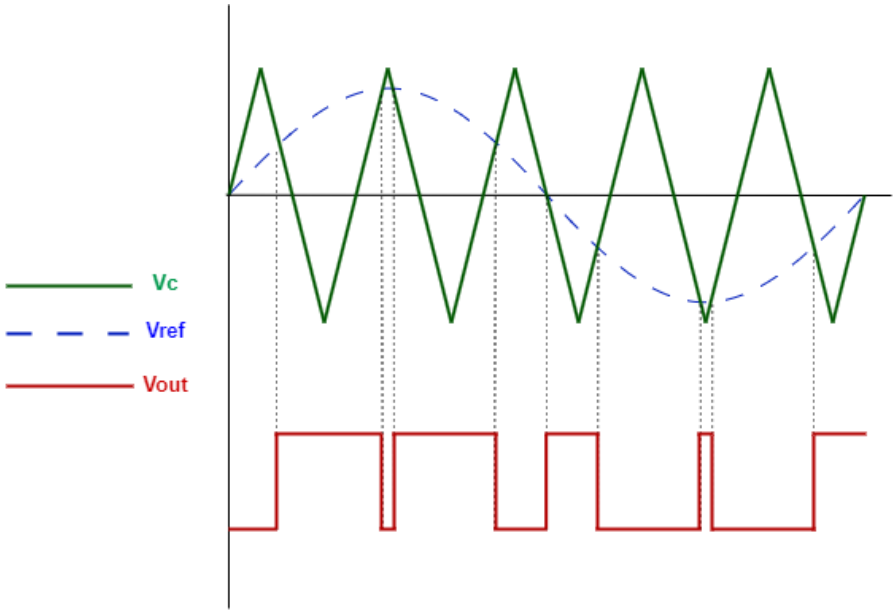

Figure 2.4: Synusoidal PWM.

The maximum amplitude of the sine wave must be less than the maximum amplitude of the triangle wave to obtain linear modulation.

2.2.4 A practical example of PWM for DC-DC Converter control

[4]The following study is based on the analysis of Dual PWM control on a conventional boost converter, as shown in Figure 2.5. The control sees two PWMs controlled simultaneously, divided into high (PWMH) and low (PWML) frequency. The two PWMs are driven by the voltage Vc, which is the resultant of the difference between the reference voltage Vref and the feedback voltage fb, obtained by dividing the voltage Vo between the two resistors *R1* and *R2*.

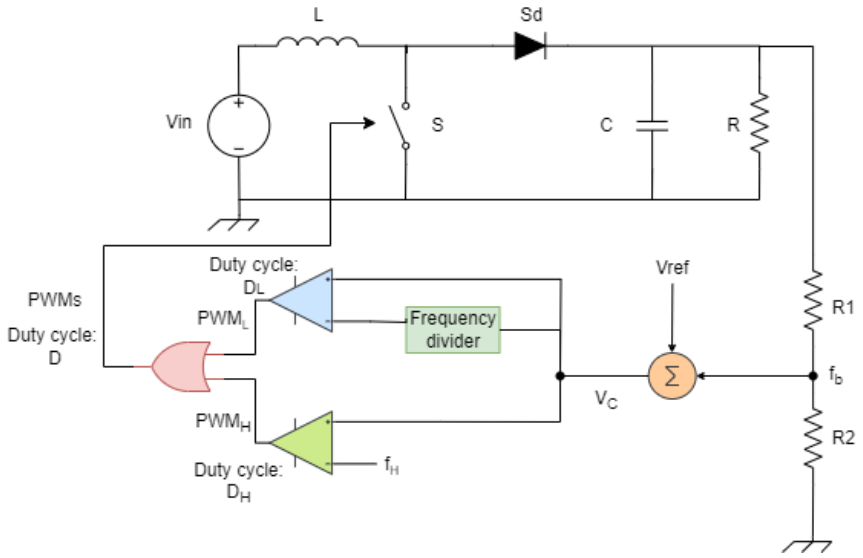

Figure 2.5: Representation of the circuit for realising Dual PWM Control

The goal of such a PWM technique is to set the (1-DL)*TL duration of the low-frequency PWM as a multiple of the period of the high-frequency PWM, resulting in the following relationship:

$$\frac{(1 - DL) * T_L}{T_H} = n$$

with $\boldsymbol{n}$ representing the number of rising and falling edges of the high-frequency PWM over time (1-DL)*TL.

These two PWMs are combined in OR logic generating the resulting PWMs that will control the converter switch. It is obtained that the period of the resulting PWM is equal to TL, while the parameter n defines the number of operating phases in a period TL, as shown in Figure 2.6. In conclusion, this study shows that the Dual PWM technique brings advantages in terms of DC-DC implementation cost and performance, as it allows quadratic voltage gain while maintaining a conventional DC-DC converter architecture.

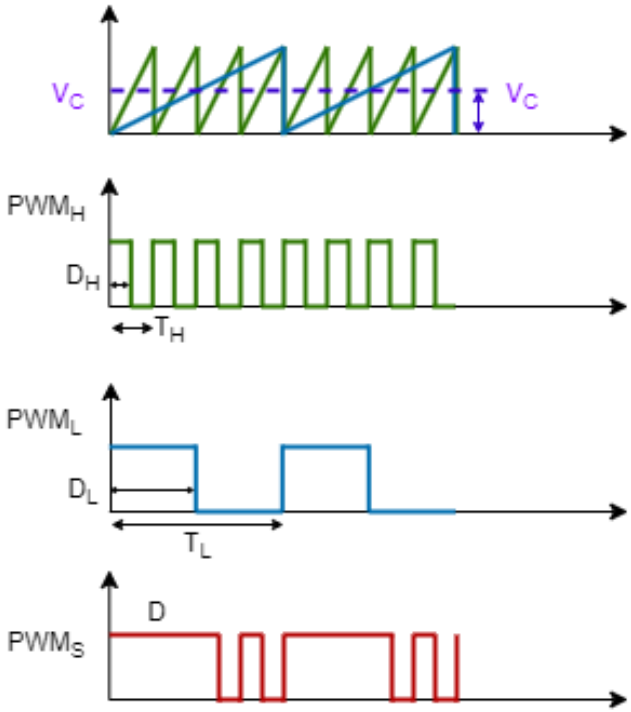

Figure 2.6: Dual PWM control output given by the combination of PWM_H and PWM_L signals

2.3 The role of Dead Time in PWM

The role of dead time in Pulse Width Modulation (PWM) is crucial in controlling high-power MOSFETs for current management. A typical application is the use of switching circuits in DCDC converters, which consist of a high-side and a low-side MOSFET. Both MOSFETs are driven by complementary PWM signals, for this reason it is essential to avoid simultaneous activation of the two MOSFETs to prevent the phenomenon known as **Shoot-through**, which would result in a short circuit between the input voltage and ground.
In order to avoid the shoot-through phenomenon, a dead time is used, which consists of setting both gate signals of the two switches to zero, thus preventing both gates from being active for certain periods of time[5].
One consequence of dead time is distortion in the duty cycle; therefore, it's important to consider these effects. Figure 2.7 illustrates two complementary signals with dead time.

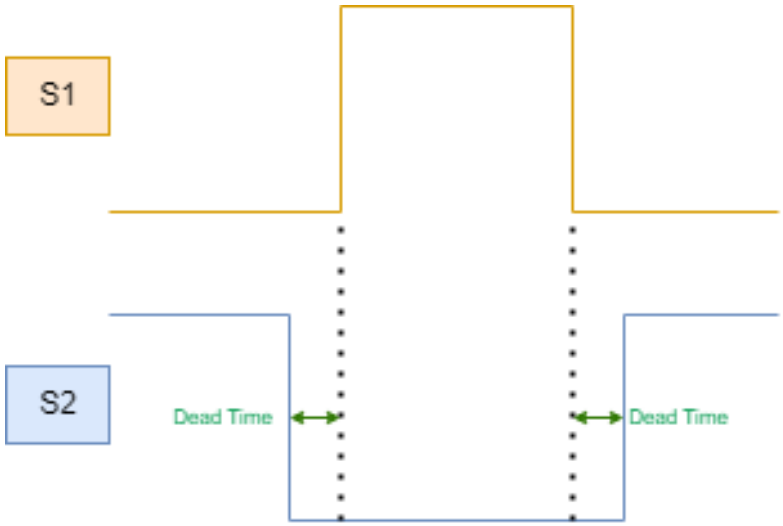

Figure 2.7: Representation of two complementary PWM signals with Dead Time

2.3.1 Dead Time Implementation Techniques in Aurix TC39

[6] In this research activity, two techniques for implementing dead time in the AURIX TC39 microcontroller are investigated. Both techniques rely on the use of the Generic Timer Module (GTM), a module widely used in the automotive sector[6], and the ATOM sub-module for PWM generation.

Dead Time implementation with PwmAtomHl.h library

The first technique employs the PwmAtomHl.h library, which allows for high-level manipulation of PWM characteristics, including dead time. The advantage of this configuration is the ability to implement complementary PWMs without using additional modules besides ATOM. However, the disadvantage is the high number of counters required, as each counter is associated to a single PWM.

Dead Time implementation with Dead Time Module

The second technique examined in the research utilises the Dead Time Module (DTM) present in the GTM, internally connected to the TOM and ATOM modules. This module is capable of generating a pair of complementary PWMs with dead time from a single input PWM. This feature represents a significant advantage, as it halves the number of counters required to generate the PWMs compared to the first method discussed.The only drawback is the introduction of a delay of three clock pulses between the input PWM and the two output PWMs, which remains constant over time, ensuring deterministic behaviour in the system, consequently, it is suitable for use in the following application.

Chapter 3

Background

3.1 AURIX TC39 and Triboard TC3x9

3.1.1 AURIX TC39 Micro-Controller

The AURIX TC39 microcontroller, depicted in Figure 4.1, manufactured by Infineon, is employed extensively in the industrial and automotive sectors due to its multi-core architecture, rendering it well-suited for real-time systems. One of the principal

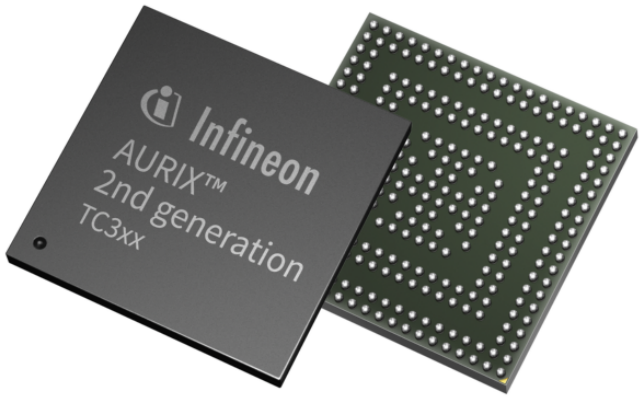

Figure 3.1: AURIX TC39 Micro-Controller[7]

characteristics of the TC39 is its core structure. In fact, the microcontroller comprises six Tricore cores operating at up to 300MHz, thereby conferring high performance and parallelised computing capabilities. Another significant attribute

of the AURIX TC39 is its memory configuration, which includes a flash memory of up to 16MB and a static RAM of up to 6.9MB. This enables the execution of highly complex applications in real time. These features render it particularly suitable for advanced automotive applications, such as driver assistance systems (ADAS), power train and chassis management. This is also due to the modules integrated within it, such as the MCMCAN module, which allows the Controller Area Network (CAN) protocol to be integrated, or the Asynchronous/Synchronous Interface (ASCLIN) module, which provides a series of protocols such as UART and LIN, thus guaranteeing robust connectivity. Another significant attribute of the AURIX TC39 microcontroller that renders it suitable for this type of application is the presence of the Generic Timer Module (GTM), a module that is capable of managing timing operations and generating PWM signals, thereby enhancing the performance of power control systems.

3.1.2 Triboard TC3x9 Evaluation Board

Figure 3.2 illustrates the Triboard TC3x9 evaluation board, which is a widely used tool in the field of software development. The board provides a series of components that permit the simulation of a wide range of automotive scenarios.

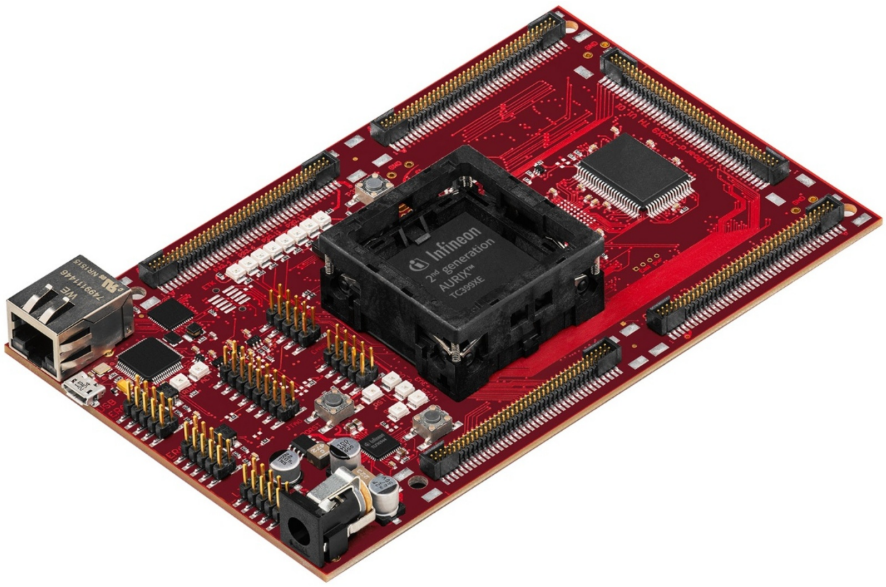

Figure 3.2: Triboard TC3x9 Evaluation board

The Triboard TC3x9 evaluation board includes the following components:

- **High Speed CAN Transceiver TLE9251VSJ**, illustrated in Figure 3.3.
- **Two 10-pin (2x5) Header** for CAN High Speed Transceiver (CAN0 and CAN1).
- **Infineon's Multi Voltage Safety Micro Processor Supply TLF35584QV**[8]

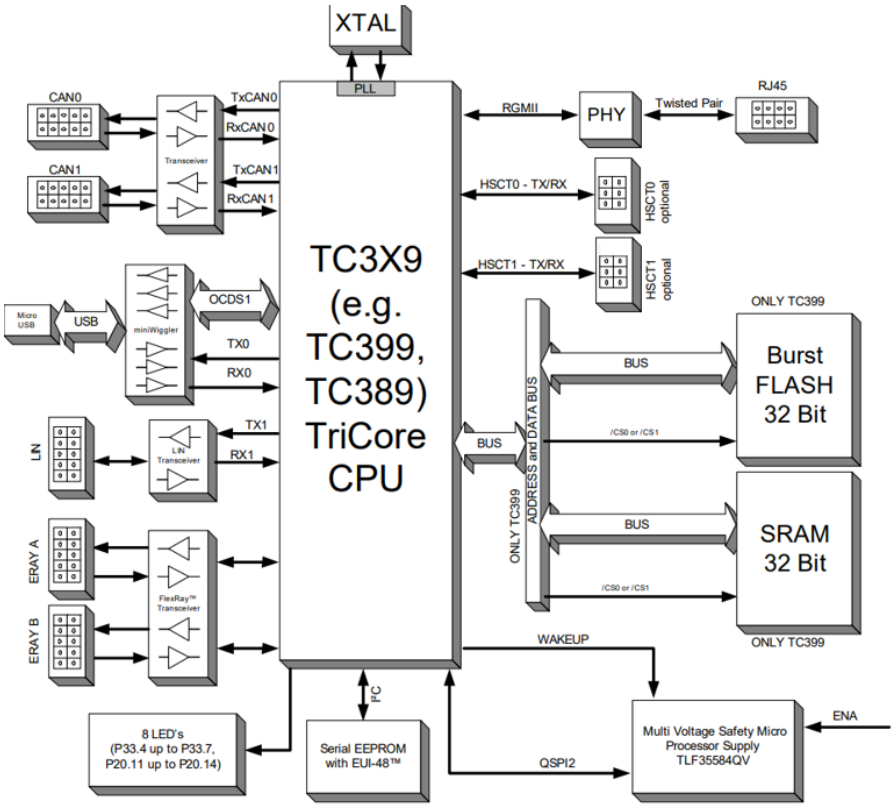

Figure 3.3: Triboard TC3x9 Architecture

3.2 Universal Asynchronous Receiver / Transmitter (UART)

The UART serial communication protocol is a widely used method of communication between electronic devices over short distances. Its implementation is relatively inexpensive, although the transmission speed is not particularly high. The protocol can be used in various modes, including Half-Duplex and Full-Duplex[9]
As an asynchronous protocol, the UART does not require the use of a clock signal to synchronise the transmitter with the receiver, which allows for a very simple communication interface. This is demonstrated in Figure 3.4. Since only two

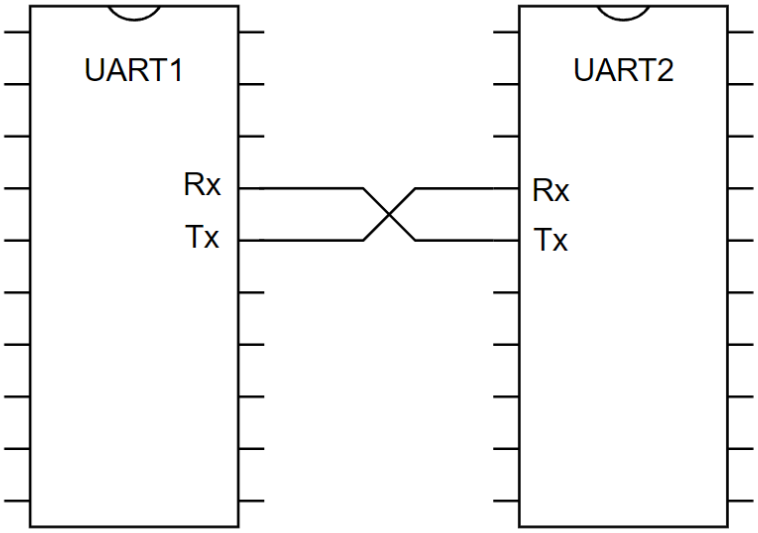

Figure 3.4: Full-Duplex UART Communication

pins are required for communication between two electronic components, one for transmission and one for reception, the interface is straightforward. Internally, in the electronic device, there is a module for managing the communication protocol. This is because the data to be transmitted or received is managed on a parallel bus by the processor, while the data in the communication line is transmitted and received in serial mode.

Figure 3.5 shows a representation of the serial transmission, while Figure 3.6 shows the management of data in parallel by the processor.

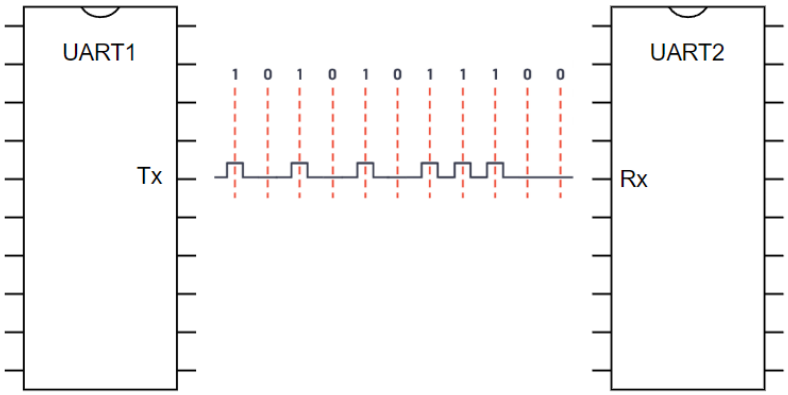

Figure 3.5: Serial Communication representation

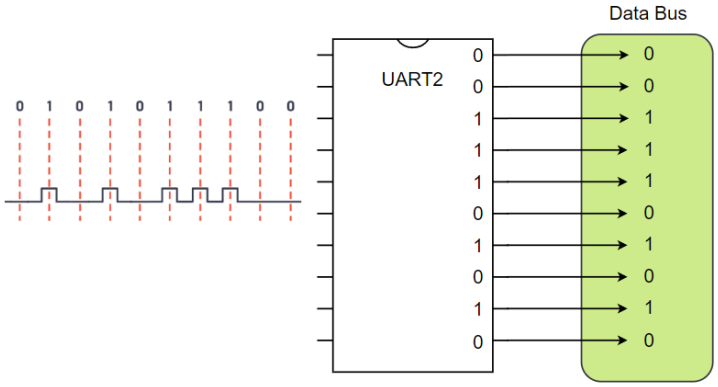

Figure 3.6: Serial-to-Parallel conversion

The parameters characterising the data frame in the UART protocol, depicted in Figure 3.7, include:

- **Baud rate**: indicates the transmission speed of a single bit. For communication to take place correctly, it is essential that the transmitter and receiver are configured with the same baud rate. Once communication commences, the initial bit is employed to notify the receiver of the commencement of transmission. This bit has a duration of one clock cycle and is typically associated with the logic low level.
- **Data frame**: contains the packet of information to be transmitted. Typically, the data frame is 8 bits long, but can vary from 5 to 9 bits.
- **Parity bit**: the UART protocol is highly susceptible to electromagnetic interference, which can result in the transmission of erroneous bits. To enhance the resilience of communication, the parity bit is employed. The parity bit is defined during transmission by counting the number of bits 1 present in the data frame. If this number is even, the parity bit will be 0. The receiver compares the parity bit with the data frame it received. If, once the number of 1 bits is counted, they are even, then the data frame has been acquired correctly.
- **Stop bit**: This bit is used to indicate the end of transmission. This is achieved by the transmitter bringing the communication line from a low to a high state for one or two clock cycles, depending on the configuration.[10]

Figure 3.7: Serial Communication representation

The UART protocol is therefore widely employed in the automotive and industrial sectors, due to its simplicity and low implementation costs. It only requires two wires for full-duplex communication and does not necessitate the use of a shared clock between the two communicating systems. Nevertheless, there are a number of limitations to the UART protocol, including a restricted size of the data frame and a lower transmission speed than other communication protocols.

3.3 Asynchronous/Synchronous Interface (ASCLIN)

The Asynchronous/Synchronous Interface (ASCLIN) module is integrated in the AURIX TC39 microcontroller in order to implement serial communication protocols, including UART, LIN and SPI. Figure 3.8 shows the internal division of the module into blocks.

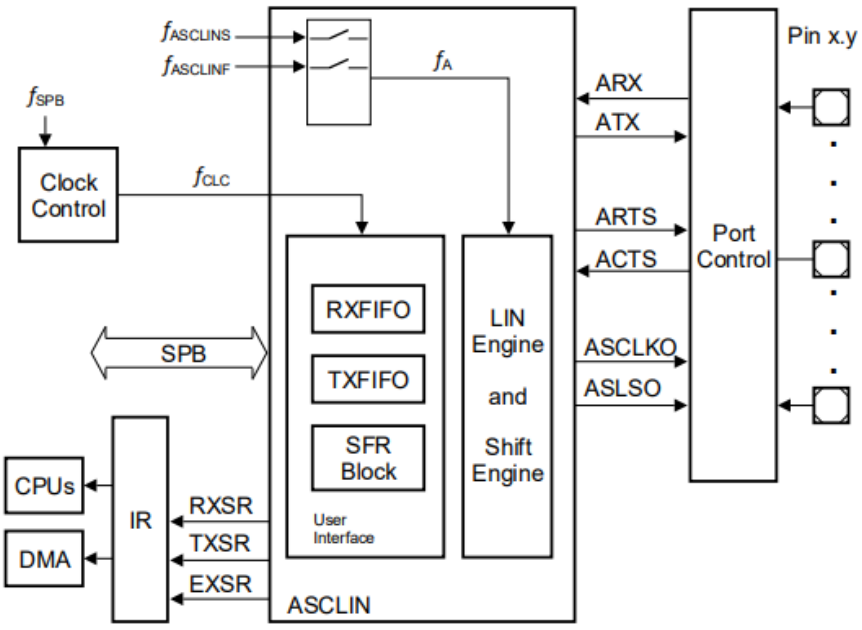

Figure 3.8: Block Diagram of the ASCLIN module[11]

The module's principal characteristics include the presence of FIFOs with a capacity of 16 bytes in both the transmission and reception modes, as well as the capacity to generate interrupts in accordance with the FIFOs' filling levels, thus enabling the transmission and reception mechanisms to be managed in a flexible manner[11].

3.3.1 Asynchronous Serial Control (ASC) features for UART implementation

In the AURIX TC39 microcontroller, the Universal Asynchronous Receiver-Transmitter (UART) communication protocol is implemented via the ASCLIN module. Referring to the Figure 3.8, the clock frequency of the module is defined by fA, which determines the maximum configurable baud rate. For $f_A = 100$ MHz, the maximum baud rate of the UART is 6.25 MBaud, while for $f_A = 200$ MHz, the maximum baud rate is 25 MBaud[11].

3.3.2 Transmission and Reception Mechanism via FIFO

The presence of FIFO memory allows for the flexible management of the transmission and reception of serial packets. This enables interrupt generation to execute the transmission or reception of a single message or data packet. The interrupt generation can be configured in three different modes, which are listed below.

Single Move Mode

In the context of transmission, each time the Transmission FIFO (TXFIFO) is emptied of a byte, an interrupt is generated that fills the TXFIFO with the remaining bytes to be transmitted. This mechanism is depicted in Figure 3.9, which illustrates the generation of interrupts in the Single Move Mode.

In the context of reception, each time the reception FIFO (RXFIFO) is filled with a byte, an interrupt is generated that moves the received byte to a destination buffer, as illustrated in the Figure 3.10.

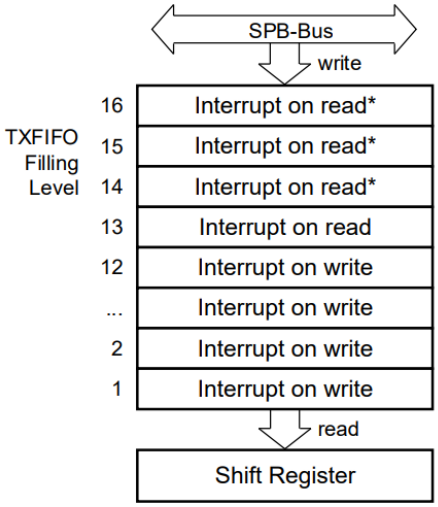

Figure 3.9: Interrupt Generation in the Single Move Mode[11]

In both the transmit and receive modes, data can be transferred via direct memory access (DMA).

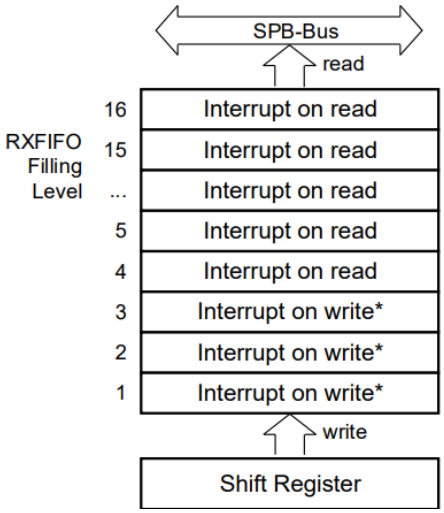

Figure 3.10: RXFIFO - Interrupt Triggering in the Single Move Mode[11]

Batch Move Mode

In the transmission phase, the TXFIFO is filled with the remaining bytes to be transmitted only if the fill level of the TXFIFO is below a user-defined threshold, as illustrated in Figure 3.11, which depicts the interrupt generation in the batch move mode.

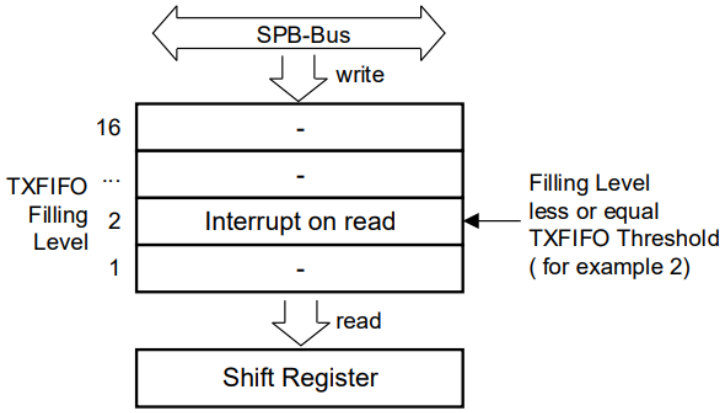

Figure 3.11: TXFIFO - Interrupt Generation in the Batch Move Mode[11]

In the event of reception, the RXFIFO is only cleared if the fill level exceeds a user-defined threshold, as illustrated in the Figure 3.12. It should be noted that this mode does not support direct memory access (DMA) for data transfer.

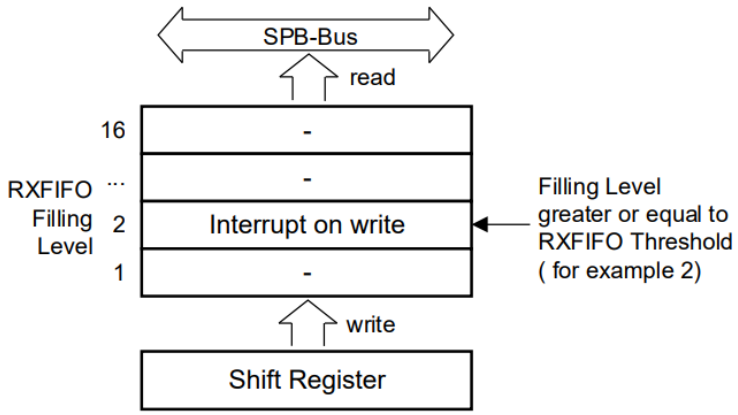

Figure 3.12: RXFIFO - Interrupt Triggering in the Batch Move Mode[11]

Combined Move Mode

In this operational mode, an interrupt is generated when the fill level falls below the threshold and the TXFIFO is emptied by one byte in transmission, as illustrated in Figure 3.13, which depicts the interrupt generation in the Combined Mode.

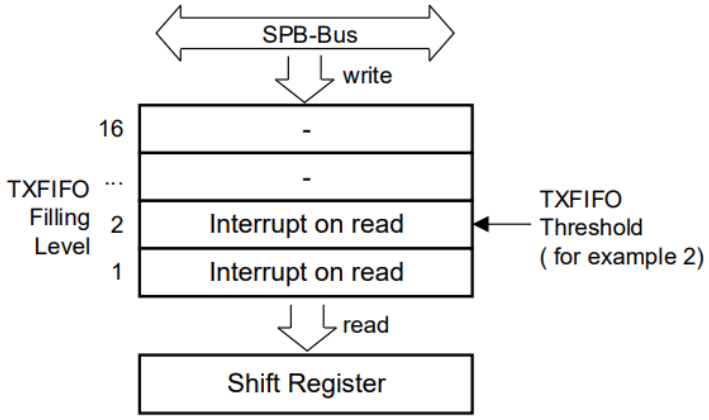

Figure 3.13: TXFIFO - Interrupt Generation in the Combined Mode[11]

On the receiving side, the interrupt is generated whenever the fill level exceeds the threshold and the RXFIFO is filled by one byte. This is illustrated in the Figure 3.14

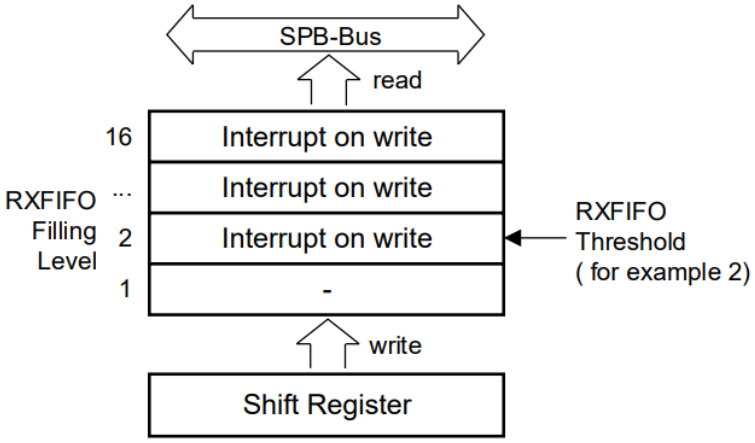

Figure 3.14: RXFIFO - Interrupt Triggering in the Combined Mode[11]

3.4 Controller Area Network (CAN)

The Controller Area Network (CAN) protocol was developed by Bosch in the early 1980s with the objective of enhancing the communication capabilities of electronic devices in vehicles. It is a robust and reliable protocol, in part due to the implementation of the bus, which involves the use of a differential signal transmitted via two twisted pair cables[12].
Figure 3.15 illustrates a typical CAN Bus configuration. This configuration comprises a series of nodes, namely electronic devices, which communicate with each other via a bus defined with two termination resistors of 120 Ω each. The aim of this configuration is to improve the reliability of communication by eliminating signal reflections on the bus.

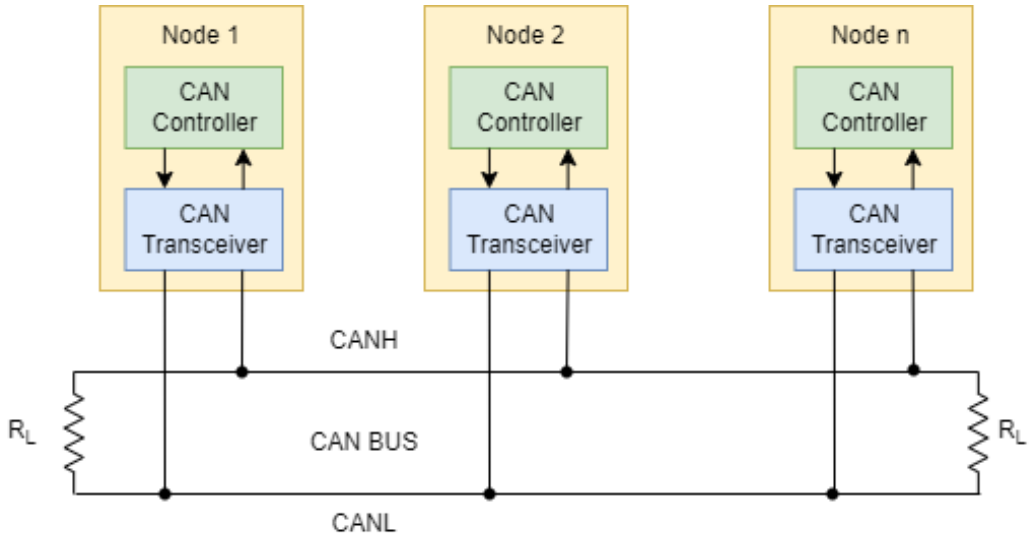

Figure 3.15: Controller Area Network Bus

CAN is distinguished from many other protocols by its Multi-Master architecture, which allows all nodes to transmit and receive messages. In order to address messages only to interested nodes, each message is assigned an ID. During reception, an acceptance filtering mechanism is applied, whereby all nodes receive messages, but only those that are really interested in a certain message acquire it, while other nodes discard the message. Furthermore, the message ID serves to determine the priority of the message. In the event of two nodes attempting to transmit simultaneously, an arbitration technique is employed. This involves both nodes initiating transmission simultaneously, commencing with their respective identifiers. At this stage, the message with the highest identifier, that is, the one containing the highest number of most significant 0s, is declared the winner of the arbitration process and is therefore permitted to continue transmission. In contrast, the message with the lowest priority must wait until the conclusion of the current transmission.
The CAN protocol frame, depicted in Figure 3.16, is comprised of the following fields:

- The **Start of Frame (SOF)** indicates the commencement of transmission, thereby synchronising the nodes on the bus.
- The **Identifier** specifies the priority of each individual message. The smaller the identifier, the higher the priority of the message.
- The **Data Length Code (DLC)** represents the maximum number of bytes that can be transmitted in a single message. The maximum value for this parameter is 8 bytes.
- The **Data Field** contains the information to be transmitted.
- The **CRC** field is a parameter consisting of 16 bits that contains the checksum of the message.

- The **ACK** field contain a bit of value 1, the message will be transmitted again, as it was not correctly received by all nodes.
- The **End of Frame** field marks the end of the message.

Figure 3.16: CAN Message Frame

Figure 3.17 illustrates an example of a CAN message. The figure shows the signal transmitted by the microcontroller (**CAN TX**) and the message in differential form in the CAN BUS converted by the transceiver (**CAN H** and **CAN L**).

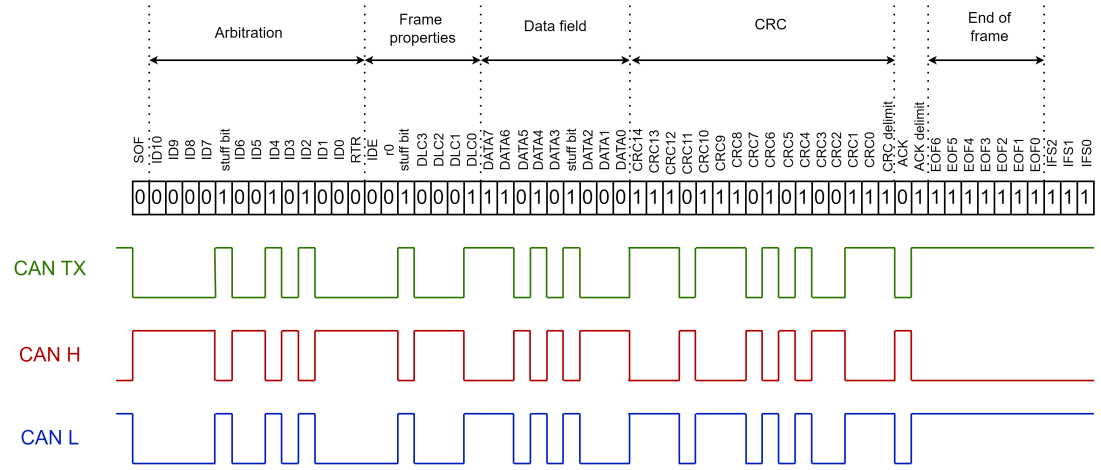

Figure 3.17: Illustration of the CAN message at the various stages. Above is the message in the form of bits. In green is the message converted into a signal, and in red and blue are the messages in the CAN BUS in the form of CAN H and CAN L

3.5 Generic Timer Module (GTM)

The Generic Timer Module (GTM) is suitable for the acquisition of precise data from multiple inputs and the generation of signals for multiple outputs. It is therefore well-suited to automotive powertrain and active safety applications, as well as industrial closed-loop applications. Figure 3.18 shows the main modules contained within the GTM that enable a wide range of timer applications.

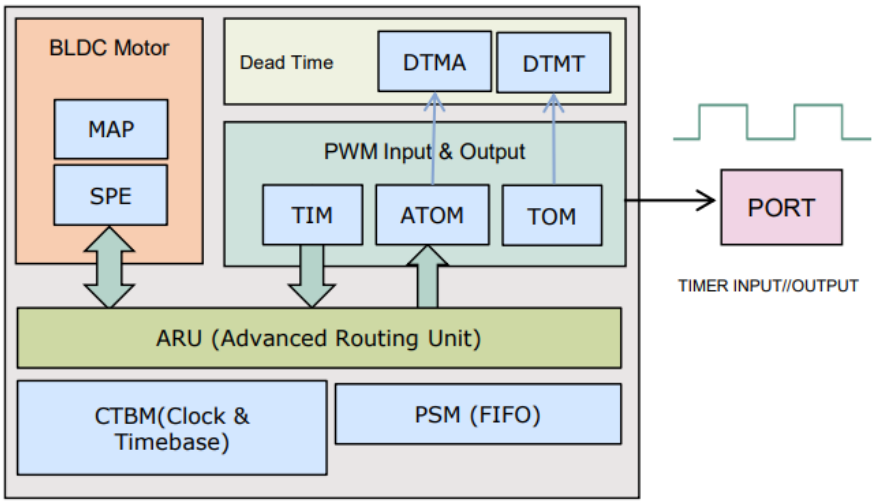

Figure 3.18: GTM Block Diagram[13]

3.5.1 Timer Output Module (TOM)

The Timer Output Module (TOM) is comprised of six clusters, each of which contains 16 cascaded channels, thereby enabling the propagation of diverse types of reset signals between channels. As illustrated in Figure 3.19 , each channel is equipped with a 16-bit counter and two compare registers, CM0 and CM1, which are employed to determine the period and duty cycle of the PWM output.

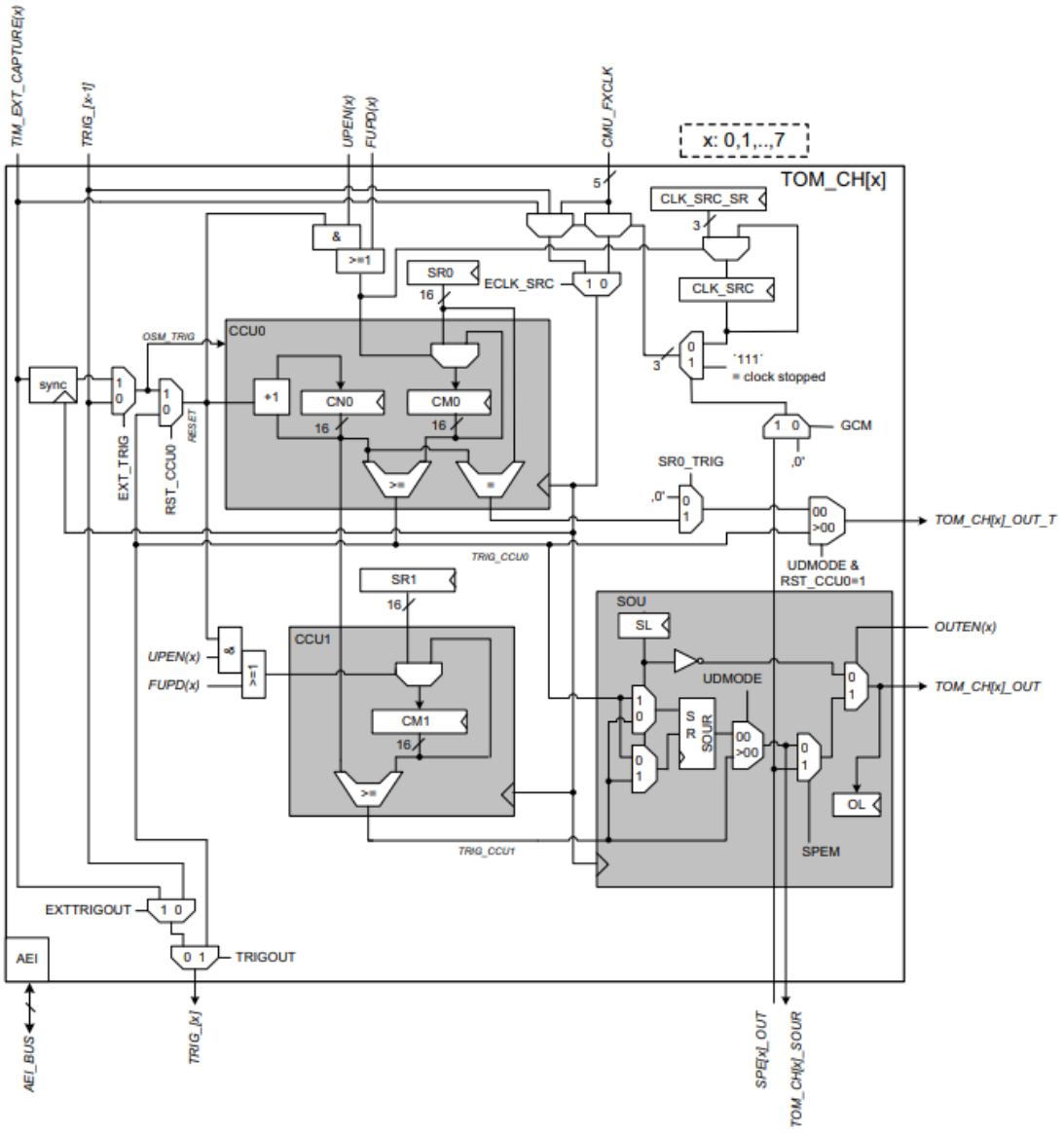

Figure 3.19: TOM Block Diagram[11]

Among the potential parameters that can be modified within the TOM, it is possible to configure the bits of the internal registers to establish one of three reset sources for the counter *CN0*. The specific bits in question are *EXT_TRIG* and *RST_CCU0*, as illustrated in Figure 3.19. The possible configurations are listed below:

- **Internal Reset** produced by condition $CN0 = CM0$.
- Reset **TRIG_[x-1]** generated by the previous channel TOM_CH[x-1].
- Reset acquired from the Timer Input Module (TIM) **TIM_EXT_CAPTURE(x)**.

The interconnection between the various sub-modules of the GTM allows for the propagation of the PWM produced to the DTM module, where two complementary PWMs with implemented dead time can be generated. An alternative is for the signals to be propagated to the EVADC module, where the A/D conversion can be triggered. Figure 3.20 illustrates the internal modules of the microcontroller, demonstrating how they can be connected to the GTM, in particular to the TOM, in order to offer a variety of functionalities.

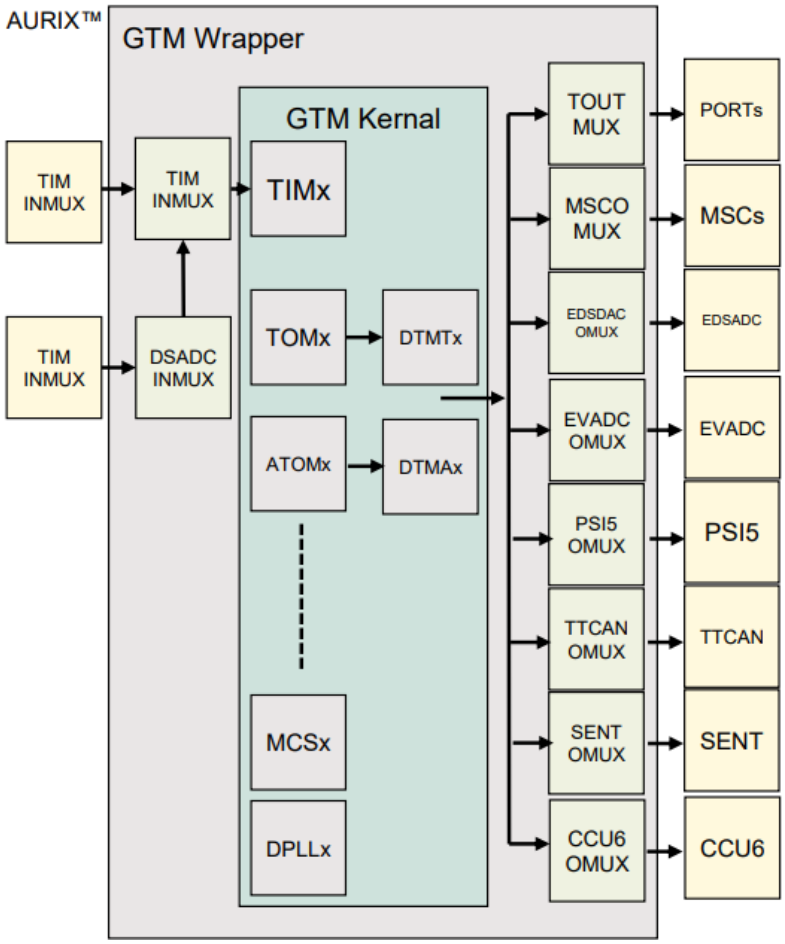

Figure 3.20: Interconnections between GTM and modules in AURIX TC39[13]

3.5.2 Tom Global Channel Control (TGC)

The TGC is a sub-module of the TOM and is responsible for the synchronous and asynchronous control of the TOM channels based on internal or external events. Each cluster of the TOM is comprised of two TGCs, each of which drives up to eight TOM channels. In particular, TGC0 controls from TOM CH0 up to TOM CH7, while TGC1 controls from TOM CH8 up to TOM CH15[11].

Referring to the diagram in Figure 3.21, the functionality provided by the following submodule is shown:

- Mechanism for enabling/disabling the counter of each TOM channel.
- Mechanism for enabling/disabling the output of each TOM channel.
- Force update mechanism to update the duty cycle or period of each TOM channel asynchronously.
- Mechanism for enabling/disabling the synchronous update of the duty cycle and period of each TOM channel

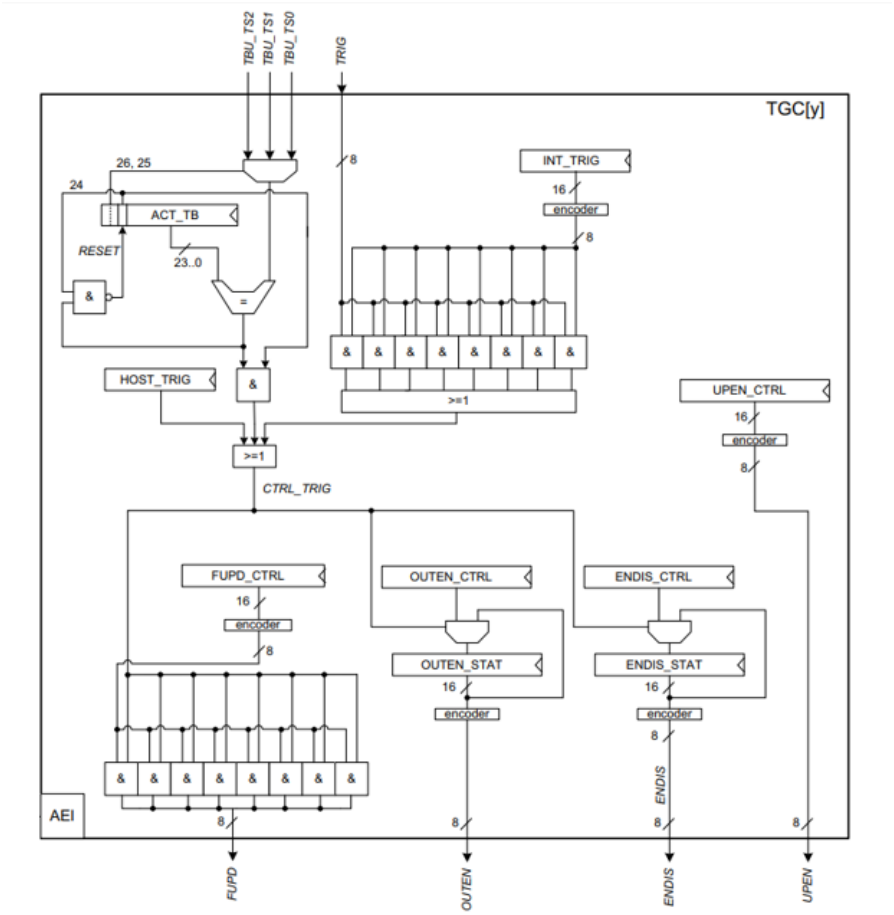

Figure 3.21: TGC Block Diagram[11]

The preceding mechanisms can be activated in two distinct modes:

- **Synchronously**: Based on a signal acquired from the Timer Input Module (TIM) or a Time Base Event
- **Asynchronously**: By writing to the *HOST_TRIG* register.

When one of the pre-defined events occurs, the contents of the shadow registers [**FUPD_CTRL, OUTEN_CTRL, ENDIS_CTRL, UPEN_CTRL**] are written to the status registers [**FUPD_STAT, OUTEN_STAT, ENDIS_STAT, UPEN_STAT**], triggering the desired mechanisms.

Chapter 4

Scope of the research

The following research project is concerned with the analysis of the AURIX TC39 micro-controller, which is employed in the development of software for the hardware control of a DC-DC converter comprising Gallium Nitride (GaN) MOSFETs. The following component is employed in a hybrid vehicle architecture comprising a battery, a fuel cell and a motor, as illustrated in Figure 1.1

4.1 Communication Network: CAN and UART

The initial phase of the research concerns the implementation of communication between the inverter module and the DC-DCs.

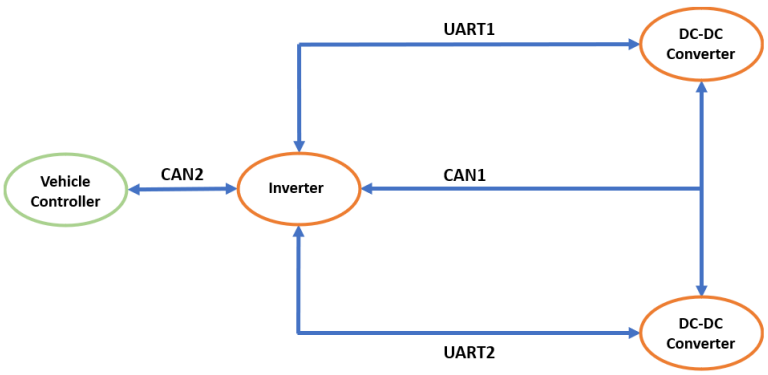

Figure 4.1: Communication Network between the three AURIX TC39

This communication is essential for the exchange of control-critical information such as reference currents and diagnostic messages. A detailed diagram of the communication scheme is presented in the Figure 4.1, which illustrates the two communication protocols, respectively UART and CAN.
The initial protocol subjected to analysis was the Controller Area Network (CAN). Two CAN lines are provided for the implementation of the initial protocol.

- **CAN1**: This bus comprises three nodes: the two DC-DC converters and the inverter. It is used to exchange diagnostic messages, including temperatures, errors and system status. The latter is necessary for controlling the DC-DC converter.
- **CAN2**: This bus comprises the inverter and the vehicle controller, and is used by the vehicle controller to send the motor reference torque.

The second protocol implemented is the Universal Asynchronous Receiver-Transmitter (UART), which is used by the inverter to communicate every 50 µs to the DC-DC converters with the reference currents to be used in the control. Each data packet comprises five bytes, four of which contain data on the reference currents, and one byte contains information on the status register.
In order to enhance the resilience of the communication, a mechanism was implemented that incorporates an additional synchronisation signal to indicate to the receiving unit the commencement of the transmission of the complete data packet. This mechanism is illustrated in Figure 4.2.

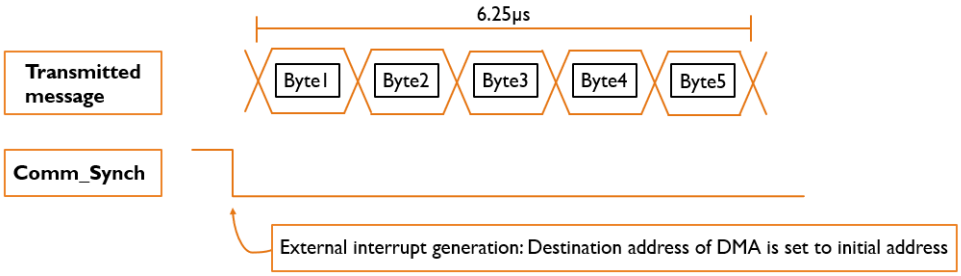

Figure 4.2: UART synchronization mechanism to improve communication robustness

This mechanism allows the receiver to discard partially sent data packets due to communication errors. Furthermore, the Direct Memory Access (DMA) mechanism is employed to transfer data from the receive FIFO to the global buffer, thus reducing the CPU overload. The entire UART communication process is depicted in the flow chart in Figure 4.3.

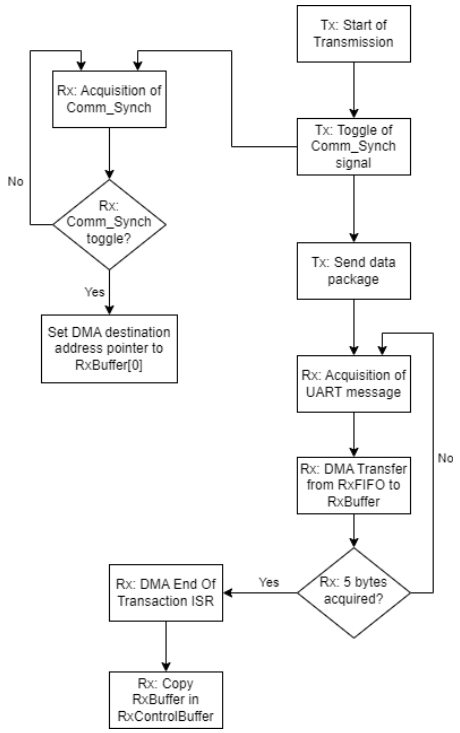

Figure 4.3: Flow Chart describing the UART communication process

4.2 External Clock Task

The research proceeded with the configuration of the microcontroller to generate the system clock from an external clock signal of 8MHz. This configuration is intended for the final configuration of the hybrid vehicle, in which the two DC-DC converters and the inverter are to share the same clock. The objective is to improve synchronisation between the three microcontrollers. This configuration is illustrated in Figure 1.2. The clocking system was then subjected to further analysis, with the oscillator circuit configured to acquire an external clock in place of the crystal signal. This was followed by the configuration of the System Phase-Locked Loop (SYSPLL) Module, which is responsible for generating the system clock from the clock provided by the oscillator circuit. During the configuration of the SYSPLL, a limitation was identified for the input clock frequency, which must be at least 10 MHz. Consequently, the search was conducted at the frequency of 10MHz. Two applications were developed to validate the system clock generated by the SYSPLL configuration, the first using the external clock output mode of the microcontroller and the second using the GTM to generate a PWM signal.

4.3 Power Structure Control Task

Subsequently, the control of the power structure, as illustrated in Figure 4.4, was implemented.

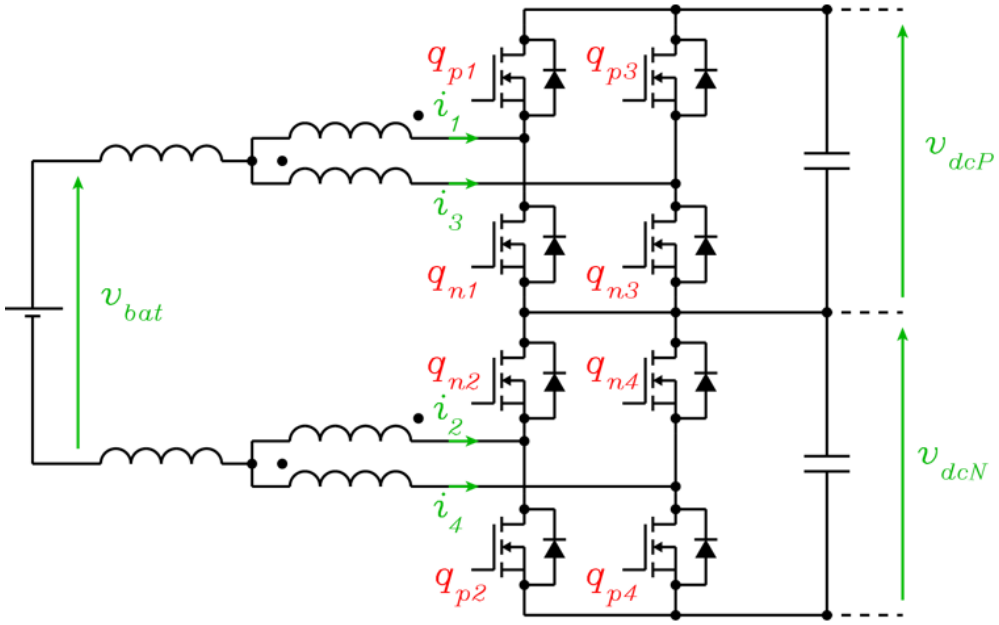

Figure 4.4: DC-DC Converter Electrical Circuit

The implementation of the aforementioned control necessitates the fulfilment of the following time requirements:

- A reference duty cycle and triangular carrier are associated with each switching pole.

- It is necessary to define a phase shift between the triangular carriers of the four switching poles, as illustrated in Figure 4.5

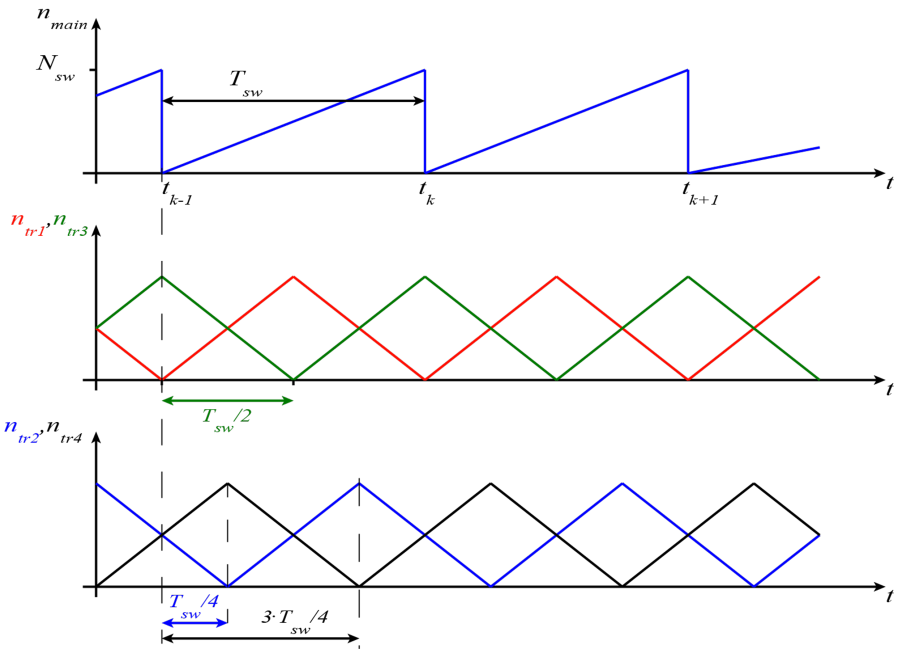

Figure 4.5: Triangular Carriers for PWM generation and Sampling instants determination

- All triangular carriers are synchronised to a main carrier, which is set by the inverter.
- The calculation of the duty cycle is controlled by an Interrupt Service Routine (ISR), which is synchronised with the main carrier.
- Each physical quantity to be sampled is associated with a specific carrier, which is synchronised with a triangular carrier controlling a switching pole. This is done in consideration of the sampling delays introduced by the sensor via the *Td_AD* factor. This is exemplified in Figure 4.6

SC_{AD_13d}	SC_{AD_24d}
i_1, i_3, v_{dcP}	i_2, i_4, v_{dcN}
v_{bat} (not request synchronous sampling)	

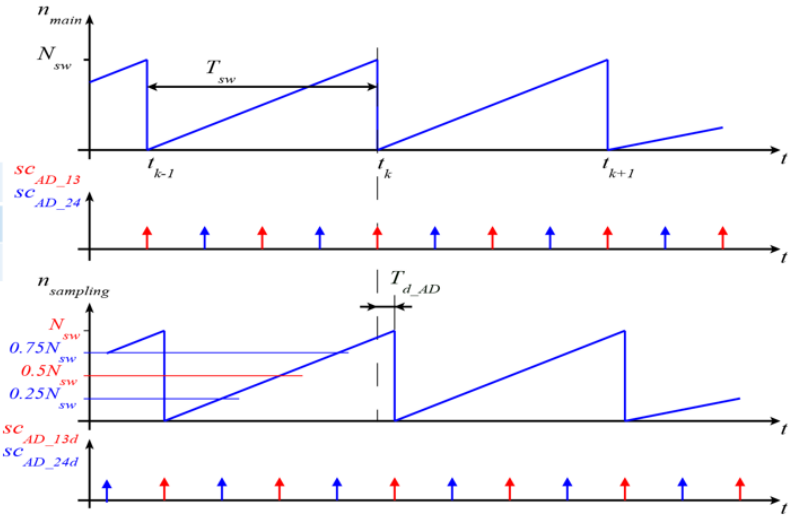

Figure 4.6: Sampling instants for A/D conversions. Parameter Td_AD takes into account the delay introduced by the sensors

- The update of the compare registers of the PWM-generating counters following the execution of the control must occur at precise instants of time, as illustrated in Figure 4.7

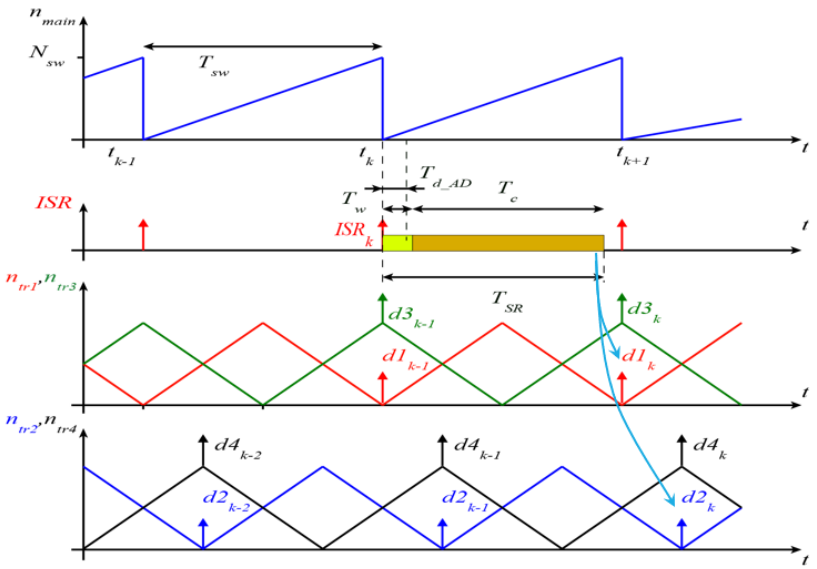

Figure 4.7: Synchronous Duty Cycle update

Finally, the duty cycle update control was implemented in the microcontroller, adapting it to the specific registers of the AURIX TC39, in particular those required to perform the synchronous update of the compare registers of the various counters involved. The call time to the ISR has been configured so that it occurs after all the samples required for the control execution are available. This is illustrated in Figure 4.8. In conclusion, it was verified that the execution time was within the set limit of 8 µs.

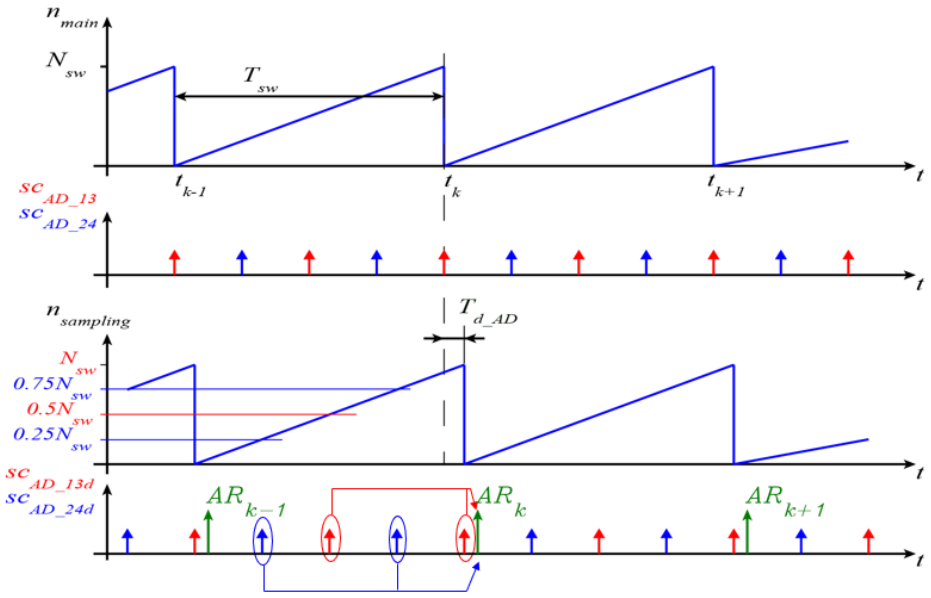

Figure 4.8: The blue samples represent the quantities i2, i4, vdcN and vbat. The red samples represent the quantities i1, i3 and vdcP. The samples circled will be used to determine the duty cycles in the ARk control task.

Chapter 5

Development Methodologies

5.1 Communication System Implementation

5.1.1 Implementation of Controller Area Network (CAN)

In order to implement the CAN protocol in the AURIX TC39, the MCMCAN module was employed, configured in the Classical CAN variant, which permits a maximum data rate of 1Mb/s.

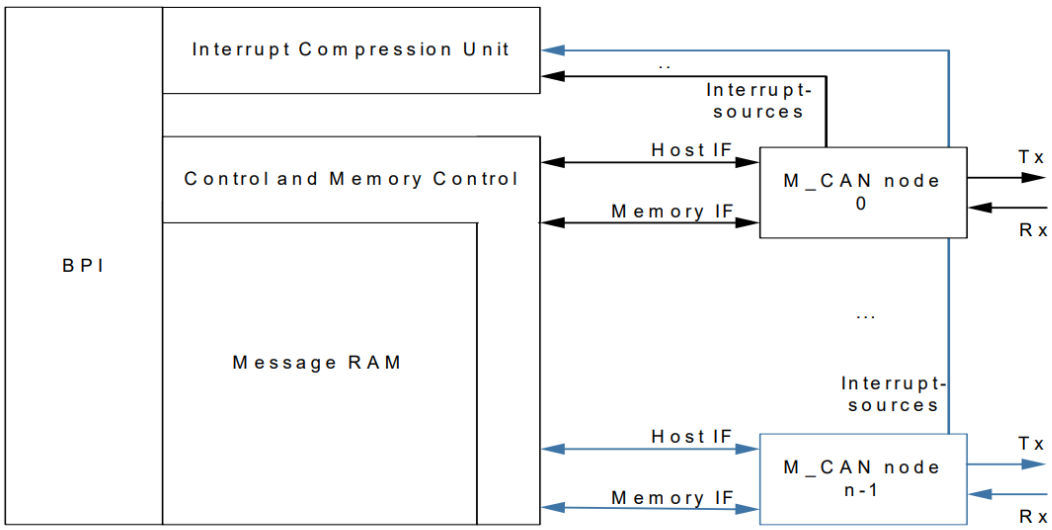

Figure 5.1: MCMCAN Module internal architecture[11]

The internal architecture of the module is depicted in Figure 5.1. Each MCM-CAN cluster comprises four CAN nodes, which share a common Message RAM. This is a portion of memory used to store messages to be transmitted or received signals. As illustrated in Figure 5.2 , three memories are available for reception: The **RX FIFO 0**, **RX FIFO 1** and **RX Buffer** are available for transmission purposes, with two additional memories being made available for this purpose: The **TX FIFO** and **TX Buffer**.

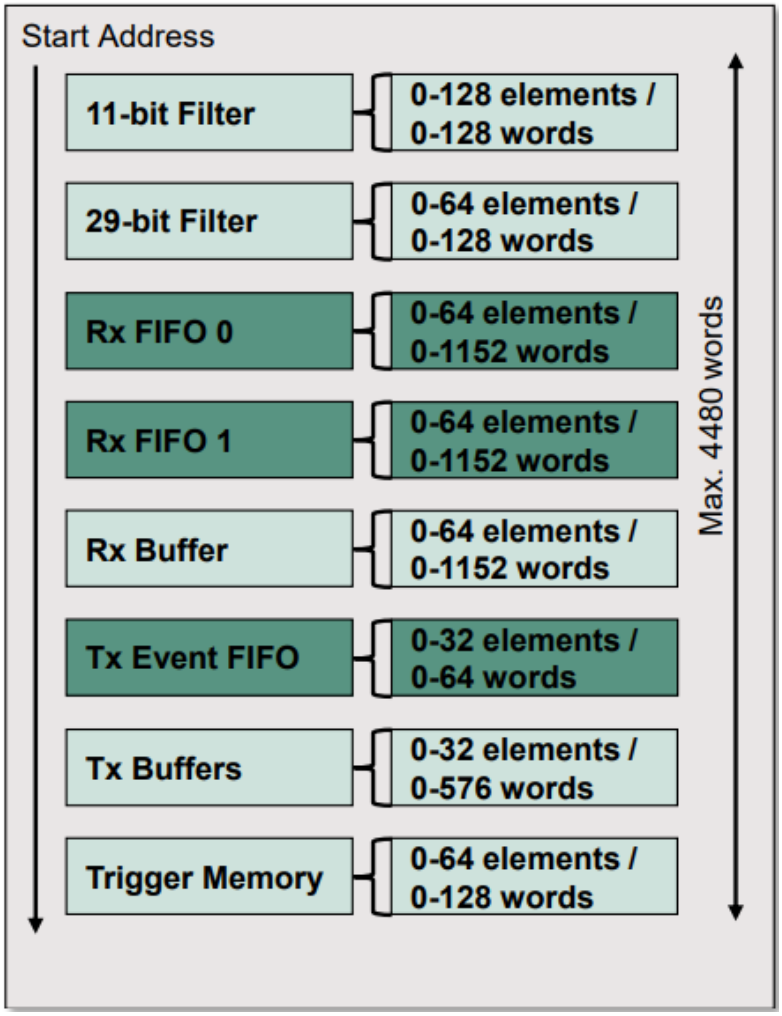

Figure 5.2: Message RAM internal subdivision[14]

In order to test CAN communication in the AURIX TC39, a CAN Database (CAN DBC) was created. This was employed to assess the efficacy of the message filtering and transmission mechanism, which was configured at 1 Mbps. Table 5.1 illustrates the CAN DBC configuration employed. The following messages are exchanged between the modules of the two communicating microcontrollers, MODULE_CAN0 and MODULE_CAN1, respectively.

CAN Message	**Message ID**	**Message Data Length Code**
Message 1	0x792	8
Message 2	0x016	3
Message 3	0x020	4

Table 5.1: definition of the DBC containing ID and DLC of the three CAN messages

As evidenced by Pseudo Code 5.1, the initial phase of the process entails configuring the two CAN modules. This involves the following steps:

- Configuring a node from the 4 available nodes.
- Setting the BaudRate of the communication.
- Configuration of the node type between Transmitter, Receiver or Transmitter and Receiver.
- Configuration of the transmit interrupt completed.
- Configuration of the receive interrupt for a new message received.
- Assignment of pins connected internally to the transceiver.

```
void initMcmcan(void)
{

    MCMCAN_initConfig(); // CAN Module initialization with default
    parameter

    MCMCAN_initNode();  // CAN Node initialization with default
   parameter

    MCMCAN.nodeId = CAN_node; // Select the CAN node

    MCMCAN.baudRate = CAN_baudrate; // Select the CAN baudrate

    MCMCAN.nodeType = CAN_type; // Configure the CAN node as
   Transmitter, Receiver or Transmitter and Receiver

    MCMCAN.interruptTransmissionComplete = TRUE; // Enable the
   interrupt generated at the end of the Transmission of a message

    MCMCAN.interruptNewMessageInRxBuffer = TRUE; // Enable the
   interrupt generated in receipt of a new message

    MCMCAN.pinConfig = CAN_pins; // Configuration of the Tx and Rx
    pins connected to the transceiver

    MCMCAN_nodeConfiguration();


    . . .
```

Pseudo Code 5.1: MCMCAN configuration function

The second part of the configuration, depicted in the Pseudo Code 5.2 , illustrates the filter configuration of the two modules, with the possibility of setting the following parameters:

- Memory in which the message will be saved.
- ID to define which messages will be accepted.
- If RxBuffer is designated as the destination memory, it is possible to select one of the available 64 receive buffers.

```
    . . .


    MCMCAN.filterNumber = filter(i); // i-th filter configuration

    MCMCAN.filterId = filterId; // configuration of the ID
   accepted by the filter

    MCMCAN.filterRxBuffer = filterBuffer(i); // Configure the i-th
    buffer as the target buffer

    MCMCAN_filter_configuration();

}
```

Pseudo Code 5.2: MCMCAN acceptance filter configuration

The transmission mechanism is configured within the function **transmitCanMessageX()**, with X representing a symbol belonging to one of the following values: **1, 2, or 3**. Each of these symbols corresponds to a specific CAN message, defined in Table 5.1.
A detailed definition of Message X is provided in the Pseudo Code 5.3. Within the aforementioned function, the **txMsg** object is configured with the ID and Data Length Code (DLC) that are to be assigned to the transmitter CAN data array **txData**[]. The transmission is initiated via the **MCMCAN_sendMessage()** function.

```
void transmitCanMessageX(void)
{

    MCMCAN_initMessage(); // CAN Message initialization with
   default parameter

    txData[] = TransmitMessage; // Definition of the array to be
   transmitted

    MCMCAN_txMsg.Id = MessageID; // Message ID definition

    MCMCAN_txMsg.DataLengthCode = MessageDataLengthCode // Message
    Data Length Code definition

    MCMCAN_sendMessage(MCMCAN_txMsg, txData[]); // Function that
   transmit the CAN message

}
```

Pseudo Code 5.3: Definition of the function that is responsible for configuring IDs and DLCs and for performing the transmission..

The reception takes place according to the configuration of the event that triggers the interrupt service routine, as shown in the Pseudo Code 5.4. This interrupt is initiated whenever the node that has received a new message successfully passes the acceptance filtering phase.

```
void canIsrRxHandler(void)
{

    MCMCAN_clearInterrupt(); // Clear New Message In Rx Buffer
   flag

    if(MCMCAN.NewMessageInRxBufferX == TRUE) // Checking which
   RxBuffer triggered the interrupt
    {
    MCMCAN_readMessage(MCMCAN_rxMsg, rxData[]); // Save the
   received message in the corresponding array
    }

}
```

Pseudo Code 5.4: ISR triggered by receiving a message that passes the acceptance filter

The receiving mechanism within the ISR involves identifying the ID of the received message within the **rxMsg** object, and then checking which of the three **RxBuffers** (one for each message) has triggered the interrupt. This is done in order to save the message in the array dedicated to it via the **MCMCAN_readMessage()** function. Table 5.2 contains the assignment of messages between **MODULE_CAN0** and **MODULE_CAN1**.

CAN Message	Transmitter Module	Receiver Module
Message 1	MODULE_CAN0	MODULE_CAN1
Message 2	MODULE_CAN0	MODULE_CAN1
Message 3	MODULE_CAN1	MODULE_CAN0

Table 5.2: CAN message assignment table

Finally, in order to implement the CAN BUS, the IDC10 connectors must be employed, with their pinout illustrated in Figure 5.3. These are internally connected to the **TLE9251VSJ** transceiver.

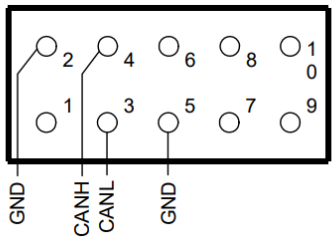

Figure 5.3: IDC10 Pinout

5.1.2 Implementation of Universal Asynchronous Receiver-Transmitter (UART)

In order to implement the 8Mb/s UART communication mechanism that allows the exchange of information between the inverter and the DC/DC converter, including reference currents and the status register, the Asynchronous/Synchronous Interface (ASCLIN) module was used, which is located in the AURIX TC39 microcontroller, whose internal structure was introduced in Chapter 3.
Direct Memory Access (DMA) was used to implement this mechanism, which allows the software behaviour to be adapted to a real-time system. DMA manages the transmit and receive data transfers between the FIFO memories and the arrays used in the software, thus reducing the load on the CPU. In the final stage of the UART protocol implementation, the External Request Unit (ERU) was configured to generate receive interrupts in response to external events, and the synchronisation mechanism was introduced to make communication more robust.

ASCLIN Configuration for Transmission

The Pseudo Code 5.5 shows the configuration of the ASCLIN module for transmission, carried out by the **initASCLIN()** function, divided into the following steps:

- Initialisation of the ASCLIN module.
- Baud Rate setting.
- Activation of single move mode for the TXFIFO.
- Configuration of the TXFIFO and setting the transmission pin

```
void initASCLIN_transmitter (void)
{

    ASCLIN_initConfig;  // UART Module initialization with default
    parameters

    ASCLIN.baudRate = UART_baudrate;    // Select the UART
   baudrate

    ASCLIN.txFifoInterruptMode = Single_Move_Mode;  // Set TxFIFO
   interrupt in Single Mode Mode

    ASCLIN.txFifo = txFifoBuffer;   // Configure the TxFIFO

    ASCLIN.pinConfig = ASCLIN_pins; // Configure the Transmission
   pin

}
```

Pseudo Code 5.5: Configuration of the Transmitter ASCLIN module

To implement the synchronisation mechanism, the transmitter node must toggle an output port. This signal is detected by the receiver node and interpreted as the start of the transmission of a 5 byte packet. The transmission mechanism is performed by the **transmitUARTMessage()** function, defined in Pseudo Code 5.6.

```
void transmitUARTMessage(void)
{
    PORT_togglePin(); // Toggle on synchronization signal

    ASCLIN_sendMessage(); // Function that initiates the
   transmission of the data packet
}
```

Pseudo Code 5.6: Configuration of the function to perform UART transmission

ASCLIN Configuration for Reception

The configuration of the UART node is shown in Pseudo Code 5.7 and includes the following steps:

- Initialisation of the ASCLIN module.
- Baud Rate setting.
- Activation of single move mode for the RXFIFO.
- Configuration of the RXFIFO and setting the receive pin.
- Configuration of the DMA as interrupt server.

```
void initASCLIN_receiver (void)
{

    ASCLIN_initConfig;  // UART Module initialization with default
    parameters

    ASCLIN.baudRate = UART_baudrate;    // Select the UART
   baudrate

    ASCLIN.rxFifoInterruptMode = Single_Move_Mode;  // Set RxFIFO
   interrupt in Single Mode Mode

    ASCLIN.rxFifo = rxFifoBuffer;   // Configure the RxFIFO

    ASCLIN.pinConfig = ASCLIN_pins; // Configure the Reception pin

    ASCLIN.interruptTypeOfService = DMA; // Configure DMA as
   server of interrupt

}
```

Pseudo Code 5.7: Configuration of the Receiver ASCLIN module

In the node configuration in single move mode, each byte received in the **RXFIFO** causes an interrupt to be generated. During this process, the CPU performs the byte extraction and storage operation from the **RXFIFO** to the destination buffer, repeating this process five times for each transmitted packet. In order to reduce the load on the CPU during these operations, DMA is used, which handles the transfer of data from the **RXFIFO** to the destination buffer in response to each interrupt generated, i.e. for each byte received.
In the implementation considered, the DMA has been configured to perform a single transaction consisting of five transfers, each corresponding to one of the five bytes of the data packet to be received, as shown in the Pseudo Code 5.8.

```
void initDMA (void)
{

    DMA_initConfig; // DMA Module initialization with default
   parameters

    DMA_initChannel;    // DMA Channel initialization with default
    parameters

    DMA.transferCount = 5;  // DMA Transfer per Transaction

    DMA.moveSize = 8;   // DMA Move of 8 bits

    DMA.requestMode = OneTransferPerRequest;    // DMA performs
   one Transfer per each Interrupt

    DMA.sourceAddress = &RxFIFO;    // DMA Takes data from the
   RxFIFO

    DMA.destinationAddress = &RxBuffer[0];  // DMA initial
   destination address

    DMA.destinationAddressIncrementStep = 1;    // DMA destination
    pointer increment of 1 bytes after a transfer
}
```

Pseudo Code 5.8: DMA configuration for UART receiver

This configuration aims to optimise the DMA transfer time, as shown in the following study [15]. At the end of the five transfer of the DMA transactions, an end-of-transaction interrupt is generated. During the ISR associated with this event, the content of the RxBuffer receive buffer is copied to the RxControlBuffer used for control. This ensures that, in the event of a communication interruption due to a malfunction, the RxControlBuffer will not be updated with the content of the last interrupted transmission, but with that of the previous one.

The Pseudo Code 5.9 shows the ISR of the end of the DMA transaction.

```
void DMA_int_Handler (void)
{

    int i=0;
    for(i=0; i<DATA_LEN; i++)
    {
        RxControlBuffer[i]=RxBuffer[i]; // At the end of the DMA
   transaction, RxBuffer is copied to RxControlBuffer
    }

}
```

Pseudo Code 5.9: ISR of the end-of-transaction DMA performed following the acquisition of 5 bytes belonging to the same data packet

To complete the communication mechanism, the ERU module was configured to generate an interrupt at the start of transmission following the toggling of the synchronisation signal generated by the transmitter. The interface pin was first configured to detect the external event, followed by the configuration of the interrupt to respond to both the rising and falling edge of the input pin. This configuration is implemented within the initERU() function as described in Pseudo Code 5.10. During the ISR generated by the input event, the DMA pointer is moved from the end of the RxBuffer array to its beginning. This process is described in Pseudo Code 5.11.

```
void initERU(void)
{

    ERU.pin = pin;  // Configure the input pin.

    ERU.inputChannel = ERU_channel; // Configure ERU channel

    ERU.enableRisingEdgeDetection();    // Enable rising edge
   detection
```

```
    ERU.enableFallingEdgeDetection();   // Enable falling edge
   detection

    ERU.enableInterrupt();  // Enable interrupt related to Rising
   and Falling detection

}
```

Pseudo Code 5.10: ERU configuration function to identify the start of UART transmission

```
void ERU_int_Handler (void)
{

    ERU_setDestinationAddress(&RxBuffer[0]);    // Resets the DMA
   pointer after the synchronization event

}
```

Pseudo Code 5.11: Interrupt service routine generated by the ERU when it detects the edge indicating the start of packet transmission.

5.2 External Clock Task

In order to implement the shared clock mechanism between the three microcontrollers in the hybrid vehicle architecture, with the objective of improving synchronisation and preventing clock shifting, the behaviour of the AURIX TC39 microcontroller mounted on the Triboard TC3x9 evaluation board was subjected to analysis. In this analysis, the clock is no longer sourced from the crystal mounted on the board, as in the default configuration, but from an external clock source. The external source must provide a 10 MHz periodic signal.

In order to develop this task, it was first necessary to examine the oscillator circuit inside the micro-controller. This circuit acquires the signal from either the crystal or the external clock source, and then propagates it to the System PLL. The System PLL is responsible for increasing the frequency of the input signal to a level that is compatible with the operational requirements of the micro-controller under standard conditions, with a maximum frequency of 300 MHz. The clock generation mechanism in the AURIX TC39 microcontroller is depicted in Figure 5.4. Two applications were developed to validate the configuration of the System PLL, in which the clock source is derived from an external signal. The first application serves to validate the behaviour using the External Clock Output mode, while the second application generates a PWM signal via the Generic Timer Module (GTM).

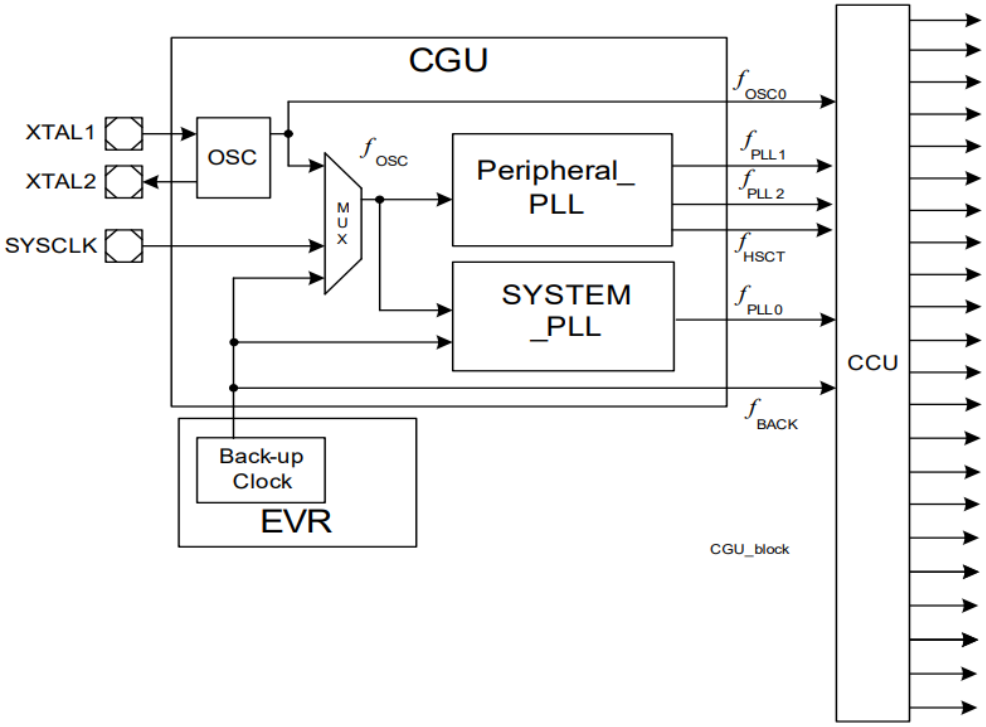

Figure 5.4: Clock Generation Unit Block Diagram[16]

5.2.1 Oscillator configuration

The oscillator circuit mounted on the AURIX TC39 has the capacity to operate in two distinct modes:

- **External Input Clock Mode**: The configuration illustrated in Figure 5.5 is employed when the input clock is a signal originating from an external source. In this mode, only the XTAL1 input pin is utilised, while the XTAL2 input pin must remain open[16]. In this configuration, the oscillator input frequency must be between 4 MHz and 40 MH[17].

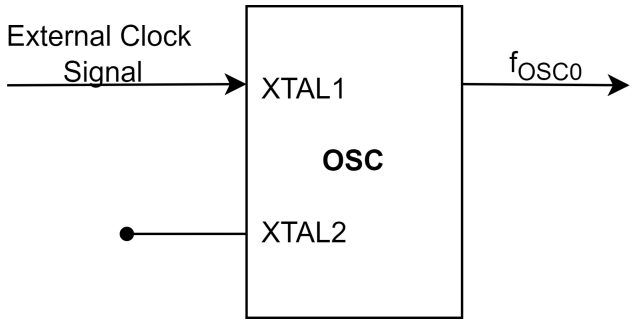

Figure 5.5: Oscillator configured in External Input Clock Mode

- **External Crystal Mode**: The configuration illustrated in Figure 5.6 involves a crystal or resonator for clock generation. In this mode, the XTAL2 pin serves to excite the crystal, while the XTAL1 pin is responsible for acquiring the generated signal. In this configuration, the crystal frequency must be between 16 MHz and 40 MHz[17].

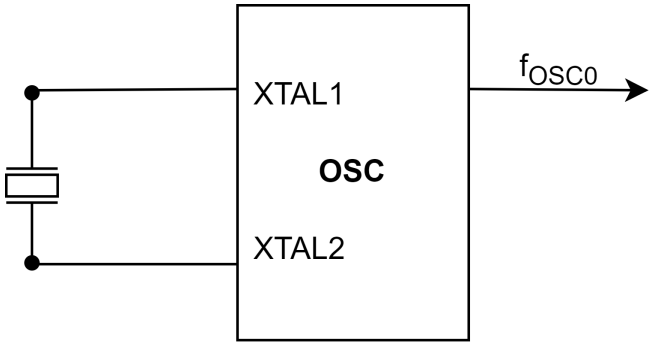

Figure 5.6: Oscillator configured in External Crystal Mode

Furthermore, the oscillator circuit also provides watchdog functionality for the input clock, monitoring that the input clock conforms, within a certain tolerance, to the clock set in the system register. This functionality is automatically enabled when the input clock to the PLL is the oscillator signal. In the event that the watchdog detects a discrepancy, the clock generated by the SYSPLL is replaced with the backup clock, a 100 MHz clock that is less accurate than the one generated by the crystal. However, this allows the system to continue operating even under fault conditions[16].

The following application, described in Pseudo Code 5.12, configures the oscillator in External Input Clock Mode to utilise the external clock. In accordance with the instructions provided in the user manual, the oscillator hysteresis was enabled for this configuration[16].

```
void initOsc(void)
{

    OSC_clearSafetyAccess();    // Enable write access to
   registers

    OSC.mode = ExternalInputClock;  // Configure the oscillator in
    External Input Clock

    OSC.hysteresis = Enabled;   // Enable Hysteresis
```

```
    OSC_setSafetyAccess();  // Disable write access to registers

}
```

Pseudo Code 5.12: Oscillator configuration for External Input Clock Mode

5.2.2 System Phase-Locked Loop Module configuration

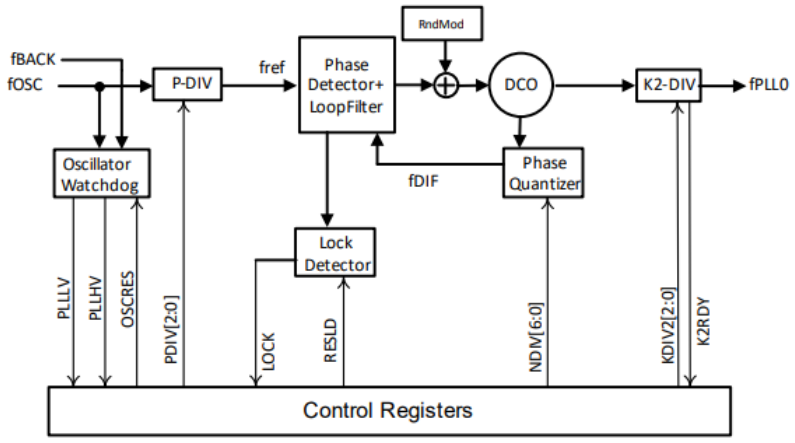

Figure 5.7: System PLL Block Diagram[16]

The SYSPLL is a frequency up-scaler for the oscillator module output, raising the frequency of the input signal to achieve the CPU operating frequency limit of 300MHz. The up-scaling mechanism is illustrated in Figure 5.7, which allows the following blocks to be identified:

- **Oscillator Watchdog**: Functionality provided by the oscillator which monitors the input clock frequency to the system to verify that it is within the operating range

- **Divider**: the input frequency $\mathbf{f_{osc}}$ is conditioned by the following divider blocks in order to obtain the output frequency $\mathbf{f_{pll0}}$, defined by the following formula:

$$f_{pll0} = \frac{N * f_{osc}}{P * K_2} \tag{5.1}$$

  For the SYSPLL to operate correctly, the frequency f_{osc} must fall within the limit defined in the datasheet. This limit is between 10MHz and 40MHz. The divider parameters N and P must be dimensioned so that the frequency within the control loop is between 400MHz and 800MHz [17].

- **Lock Detector**: Monitors that the frequency of the DCO is within the range [400MHz, 800MHz].

The objective of the SYSPLL configuration is to increase the input frequency from 10 MHz to 300 MHz, in order to fully utilise the capabilities of the AURIX TC39. Table 5.3 illustrates the configuration of parameters **P**, **N** and **K2** to achieve the desired final frequency. The configuration of the SYSPLL parameters is illustrated in Pseudo Code 5.13.

N Parameter	P Parameter	$\mathbf{K_2}$ Parameter
60	1	2

Table 5.3: configuration of SYSPLL parameters N, P and K2

```
void initSYSPLL(void)
{

    SYSPLL_initConfig();    // SYSPLL Module initialization with
   default parameters

    SYSPLL.nDivider = 60;   // Configuration of N divider

    SYSPLL.k2Divider = 2;   // Configuration of K2 divider

    SYSPLL.pDivider = 1;    // Configuration of P divider
```

```
}
```

Pseudo Code 5.13: System PLL configuration to raise the frequency from 10 MHz to 300 MHz

Two applications were developed with the objective of verifying the correctness of the oscillator and system PLL configuration.

Application 1: Validation using External Clock Output

The initial application is to validate the behaviour of the micro-controller when it receives a 10MHz clock signal as input, utilising the External Clock Output mode. To operate in the following mode, the AURIX TC39 incorporates a Fractional Divider module, as illustrated in Figure 5.8. This module accepts the input frequency f_{spb} and divides it by a factor of 1/n in **Normal Divider Mode** or by a factor of n/1024 in **Fractional Divider mode**.

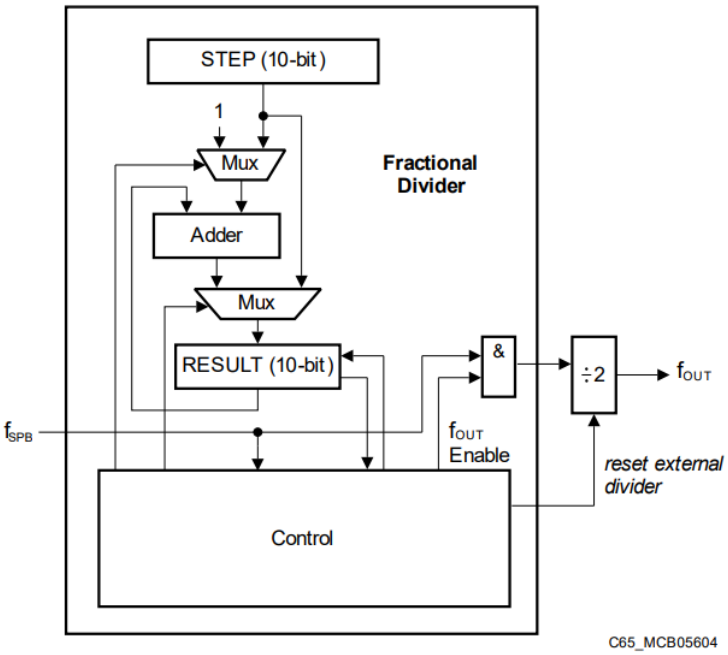

Figure 5.8: Oscillator configured in External Crystal Mode[16]

In the following application, **Normal Divider Mode** was employed, resulting in the generation of the following signal as an output:

$$f_{out} = \frac{f_{spb} * 1}{n * 2} \tag{5.2}$$

with $\mathbf{n} = 1024 - \mathbf{STEP}$, where **STEP** can have a range from $\mathbf{000_H}$ to $\mathbf{3FF_H}$. The clock output signal $\mathbf{f_{out}}$ is divided by 2 in the above formula to produce a square-wave signal with 50% duty cycle at the output.
The frequency $\mathbf{f_{spb}}$ is obtained via the Clock Distribution (CCU) module shown in Figure 5.9. The following module has the role of distributing the system clock to the various modules of the AURIX TC39. Each module is associated with a divider that adapts the input frequency $\mathbf{f_{source0}}$ to the operating frequency of the individual module.

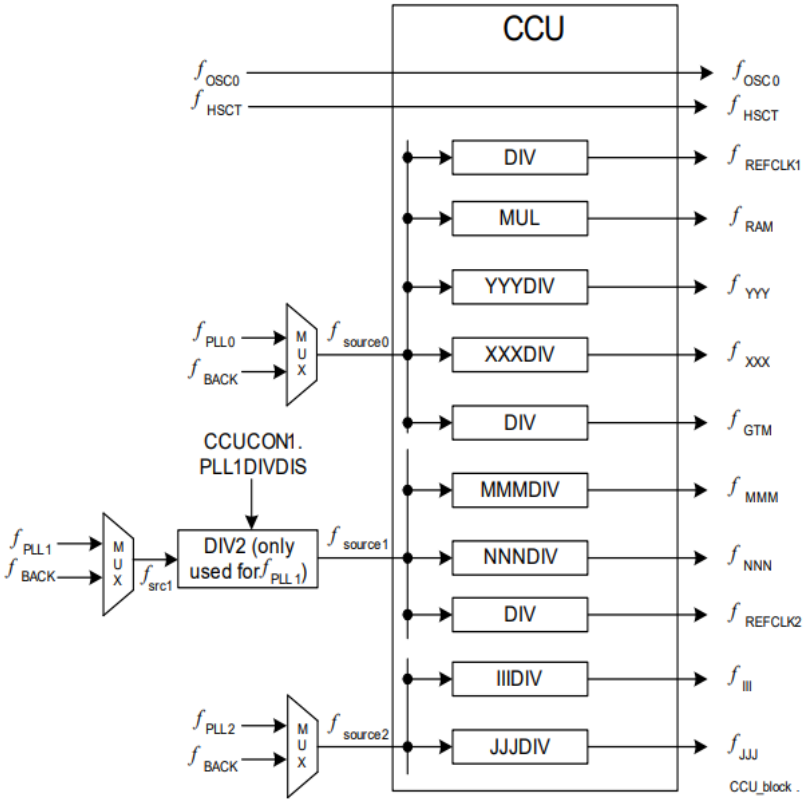

Figure 5.9: Clock Distribution Unit Block Diagram[16]

In the following application, the desired frequency for f_{spb} is 100 MHz. In order to obtain this frequency, it is necessary to divide the frequency $f_{source0}$, which corresponds to f_{pll0}, by a factor of 3. This is because the configuration of the SYSPLL has been set up in such a way that parameters N, P and K2 have been configured to obtain $f_{pll0} = 300$ MHz.

The objective of the subsequent application is to generate a signal with a frequency of 1 MHz and analyse it using an oscilloscope to determine frequency of the signal. This approach enables the determination of the frequency $f_{source0}$, which corresponds to f_{pll0}, via the following formula:

$$f_{source0} = f_{out} * SPB_{DIV} * n * 2 \tag{5.3}$$

The parameters employed in the equation are enumerated in the following list:

- $\mathbf{f_{out}}$: frequency produced by External Clock Output Mode, measured by oscilloscope
- $\mathbf{SPB_{DIV}}$: divisor by a factor of 3 to obtain frequency f_{spb} from $f_{source0}$
- $\mathbf{n = 1024 - STEP}$, where $STEP = 974$ to obtain $f_{out} = 1$ MHz.

This process allows for the verification of the correct configuration of the SYSPLL. The following Pseudo Code 5.14 illustrates the configuration of the CCU and External Clock Output mode.

```
void initCCU(void)
{

    CCU_clearSafetyAccess();    // Enable write access to
   registers

    CCU_initConfig();   // CCU Module initialization with default
   parameters

    CCU.ClockSel= 1;    // Configure f_source0 = f_pll0
```

```
    CCU.SPBDivider = 3; // Configure SPB divider = 3 -> f_spb=
   f_sourc0/3

    CCU_setSafetyAccess();  // Disable write access to registers

}

void initExtClk(void)
{

    EXTCLK_clearSafetyAccess(); // Enable write access to
   registers

    EXTCLK.enable = TRUE;   // Enable External Clock Output Mode

    EXTCLK.outputSignal = f_out;    // Select Fractional Divider
   as output signal

    EXTCLK.mode = NormalMode;   // Configure Fractional Divider in
    Normal Mode

    EXTCLK.step = 0x3CE;    // Select step = 974 to obtain n=50

    EXTCLK_setSafetyAccess();   // Disable write access to
   registers

}
```

Pseudo Code 5.14: Configuration of CCU and External Clock Output Mode for Application 1

Application 2: Validation using GTM

In the second application, the correct operation of the AURIX in External Input Clock mode was verified using the GTM. The input clock frequency of the GTM is defined by the CCU in the following ways:

- $f_{GTM} = \frac{f_{sourceGTM}}{GTM_{DIV}}$
- $f_{sourceGTM} = f_{spb} * 2$ if $\mathrm{GTM_{DIV}} = 1$, otherwise $f_{sourceGTM} = f_{source0}$

In this instance, $\mathrm{GTM_{DIV}} = 1$ was configured, resulting in $f_{GTM} = f_{spb} * 2 = 200MHz$. This is consistent with the approach taken in Application 1, where $f_{spb} = \frac{f_{source0}}{3}$.
The Clock Management Unit (CMU), depicted in Figure 5.10, is responsible for generating the clocks required by the GTM's various sub-modules, including the Timing Module (TIM), the Timing and Output Module (TOM), and the ATOM. With regard to the TOM, this module is capable of utilising one of five fixed clocks, each with a division factor with respect to the input clock.
In the following application, the clock $CMU\_FXCLK0 = \frac{CMU\_CLK0}{2}$ has been enabled. In the theoretical case, this should be $CMU\_FXCLK0 = \frac{200MHz}{2} = 100MHz$. In this context, the TOM is employed to generate a PWM with a frequency of 2 kHz and a duty cycle of 50%. Pseudo Code 5.15 shows the configuration of the CCU and TOM to realise Application 2.

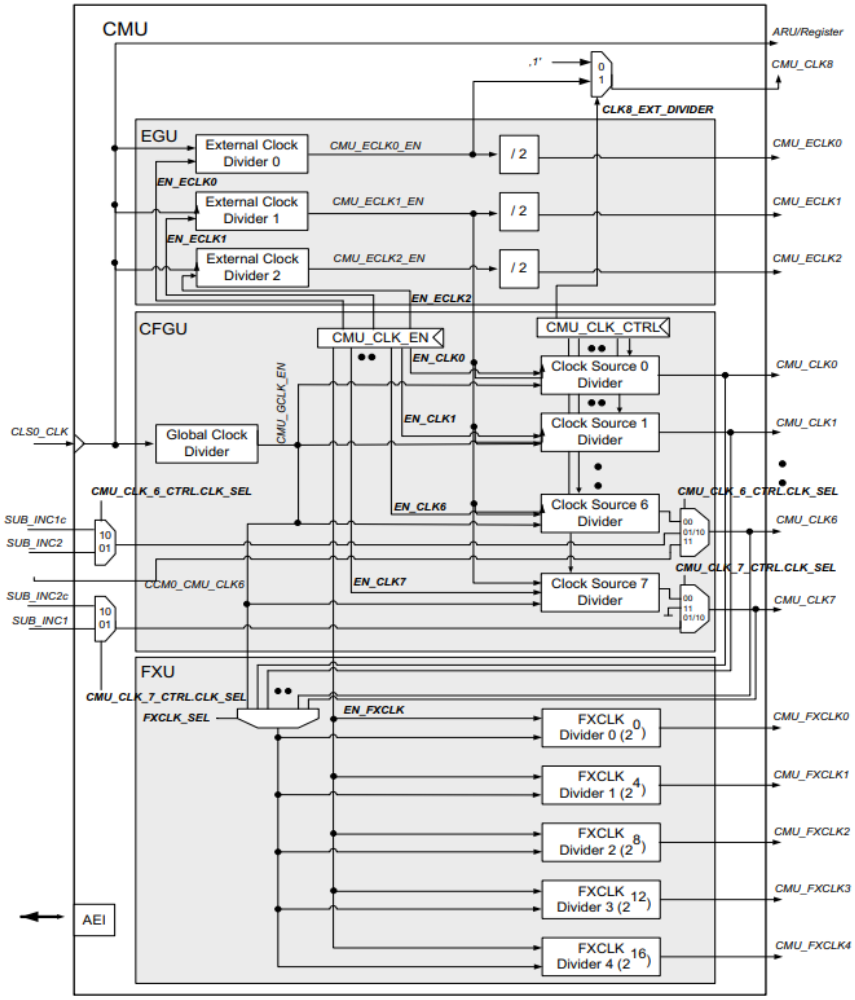

Figure 5.10: Clock Management Unit Block Diagram[11]

```
void initCCU(void)
{

    CCU_clearSafetyAccess();     // Enable write access to
   registers

    CCU_initConfig();    // CCU Module initialization with default
   parameters
```

```
    CCU.ClockSel= 1;    // Configure f_source0 = f_pll0

    CCU.SPBDivider = 3; // Configure SPB divider = 3 -> f_spb=
   f_sourc0/3

    CCU.GTMDivider = 1; // Configure f_sourceGTM=2*f_spb and f_GTM
   =f_sourceGTM

    CCU_setSafetyAccess();  // Disable write access to registers

}


void initGTMTom(void)
{

    GTMTom_initConfig();    // GTM Module initialization with
   default parameters

    GTMTom_enablefxclk0();  // Function to enable the Fixed Clock
   0 frequency to the TOM

    GTMTom.channel = 0; // Configure TOM Channel 0 to generate the
    PWM

    GTMTom.period = 49999;  // Configure the period in ticks which
    correspond to 2kHz PWM if the system clock is correct

    GTMTom.dutyCycle = 24999;   // Configure the duty cycle in
   ticks which correspond to 50% if the system clock is correct

    GTMTom.outputPin = outPin;  // Configure the output pin of the
    PWM signal

    GTMTom_start(); // Function to start the PWM generation

}
```

Pseudo Code 5.15: Configuration of CCU and TOM for Application 2

5.3 Power Structure Control Task

In order to implement the Power Structure Control to manage the power flows of the DC/DC converter, the TOM was used in the configuration shown in Figure 5.11, where the eight channels of the TOM are connected in cascade and all synchronised with respect to the EXT_CLK signal from the AURIX TC39, which is managed by the inverter.

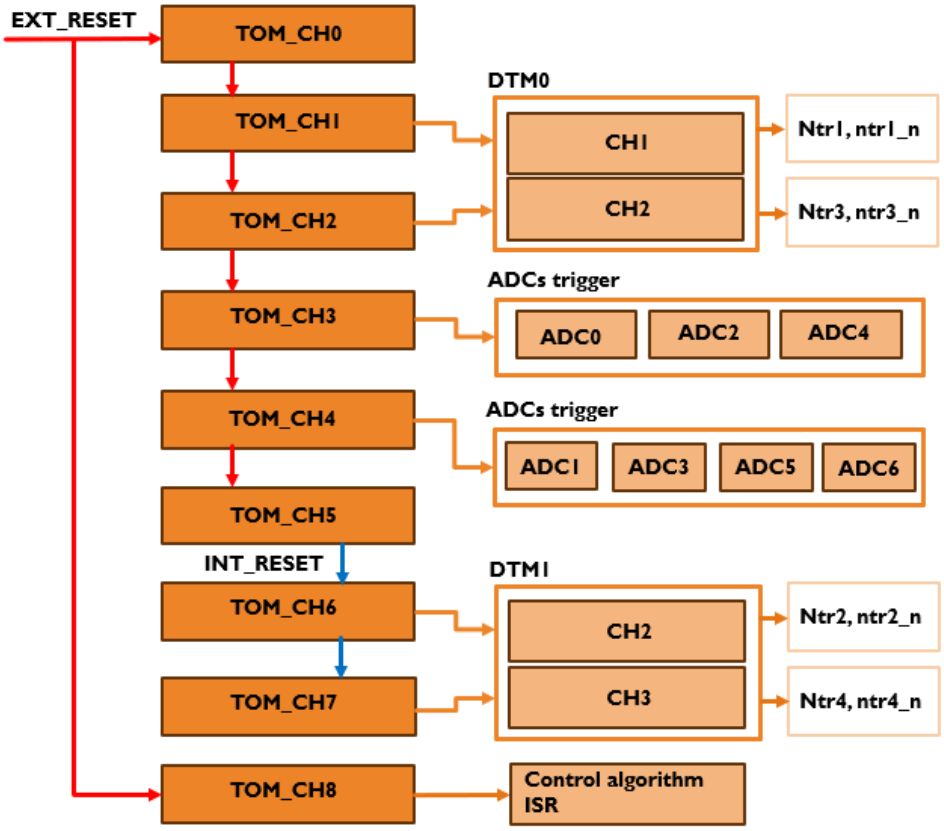

Figure 5.11: TOM Channels configuration

During the initial phases, immediately following the system's activation, the reset signal from the inverter is not transmitted. At this stage, the DC/DC converter is in an idle state in which all the micro-controller modules are active, but the counters that manage all the DC/DC control have not yet been activated. Upon system startup, the reset signal from the drive is transmitted. The TGC is triggered from the moment the reset signal is detected by the TIM module, resulting in the synchronous activation of all counters depicted in Figure 5.11, although with a slight delay.

The small delay introduced by the TGC in enabling all the counters in synchronous mode precludes the possibility of perfectly replicating the modulating triangular signals defined in Figure 4.5. This is because the TOM channel counters must be used in Up-Down mode, and if an error is present at the outset of counting, it will persist throughout the system's operational phase.

The solution to this problem is to replicate the modulating triangle waves by means of counters in Up mode, so that even if there is a small phase shift in the first period when the channels start counting, this error is immediately cleared by the arrival of the next reset signal, i.e. after the first $10\mu s$ after start-up.

Figures 5.12 and 5.13 show the difference between the two counting modes. In Up-Down mode, on arrival of the external reset (**TIM_EXT_CAPTURE)** the direction of counting is reversed. In Up mode when the reset (**TIM_EXT_CAPTURE)** arrives, the counter value is reset to zero in the following case when $CN0 = CM0$, but in external reset mode it is reset when the **TIM_EXT_CAPTURE** signal arrives.

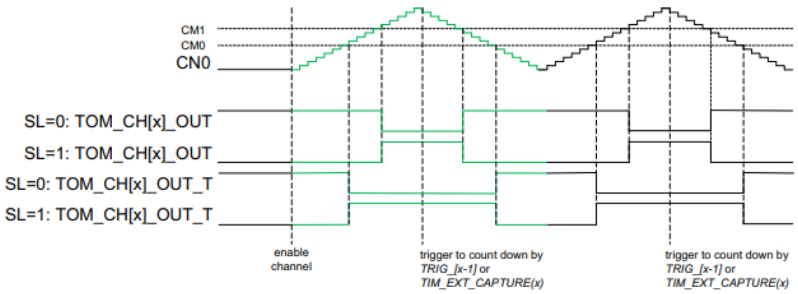

Figure 5.12: UP-DOWN counting mode[11]

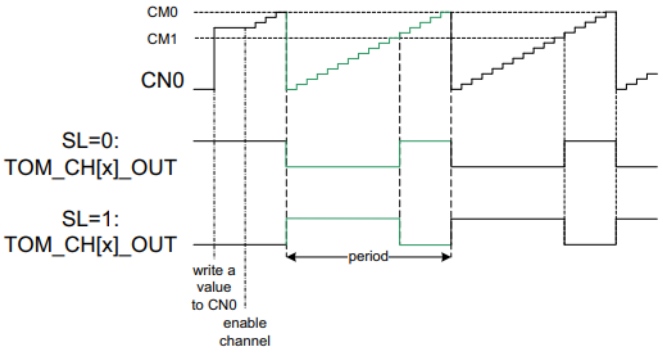

Figure 5.13: UP counting mode[11]

The following section describes the functions associated with the counters within the structure represented by Figure 5.11:

- **CH0**: Watchdog timer. The function of this channel is to monitor the reset signal generated by the inverter, verifying that it arrives with a frequency of 100 kHz within a specified tolerance range of 100 ns. In the event that the watchdog window is exceeded, the following timer generates an interrupt. This allows for the internal configuration of CH0 to be modified so that the reset no longer originates from the inverter,but it is CH0 itself that generates the reset signal for the other counters of the TOM. This ensures that DC-DC operation can continue even in the absence of the reset signal from the inverter.

- **CH1, CH2, CH6, CH7**: The following channels have the task of reproducing the four triangular signals shown in Figure 4.5. These are **ntr1, ntr3, ntr2 and ntr4**, respectively. As illustrated in Figure 5.11, channels CH1 and CH2 are reset by the signal **EXT_RESET** and are connected to channels CH1 and CH2 of DTM0, which generates two complementary PWMs with dead time for each input PWM. In contrast, channels CH6 and CH7 are reset by the signal **INT_RESET**, generated by CH5, which implements the phase shift between the four carriers. The latter are also connected to the DTM module, specifically to channels CH2 and CH3 of DTM1.

- **CH3, CH4**:The following channels are responsible for generating the rising and falling edges that will be acquired by the EVADC module and will act as triggers for the conversions depicted in Figure 4.6.

- **CH8**: The following channel is responsible of triggering the control task at a frequency of 100 kHz.

5.3.1 Timer Input Module (TIM) Configuration for External Reset Mechanism

The Timer Input Module (TIM) is responsible for acquiring the reset signal from the inverter in order to ensure synchronisation between the counters on the TOM. Figure 5.14 illustrates the block diagram of the TIM module, which depicts a series of channels connected in cascade.
Each channel receives a series of signals, including **TIM_IN(x)**, and generates the **TIM_EXT_CAPTURE(x)** signal, which propagates up to the TOM module to provide resets to the counters. In the subsequent configuration, the **TIM_CHANNEL 0** is employed, which will generate the **TIM_EXT_CAPTURE(0)** signal.

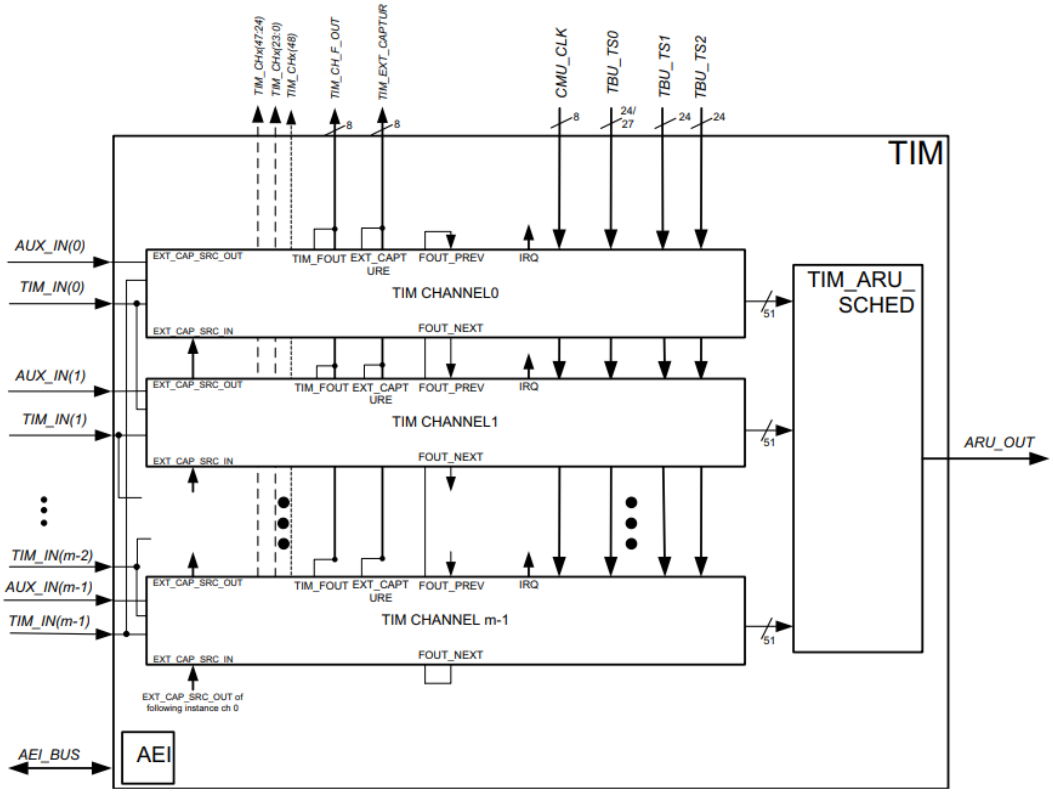

Figure 5.14: Timer Input Module Block Diagram[11]

The mechanism for generating the **TIM_EXT_CAPTURE** reset signal is based on the acquisition and identification of a rising edge, which is monitored by the **EXT_CAP_SRCx** sub-block shown in Figure 5.15 . This sub-block is responsible for generating the pulse to reset the counters of the TOM. The Pseudo Code 5.16 illustrates the configuration of the TIM in order to achieve the desired behaviour.

```
void initTim(void)
{

    GTM_enable();   // Function to enable the GTM

    TIM_initConfig();   // TIM initialization with default
   parameters
```

```
    TIM.clusterIndex = 0;    // Configure TIM cluster 0

    TIM.channelIndex = 0;    // Configure TIM channel 0

    TIM.inputPin = InputPin;     // Configure the peripheral pin
   from which the reset signal is acquired

    TIM.activeEdge = RaisingEdge;    // Configure Raising Edge
   detection

    TIM.mode = ExternalCapture; // Configure TIM mode in External
   Capture Mode

    TIM_init();

}
```

Pseudo Code 5.16: Configuration of TIM to acquire the external reset signal and generate the internal reset TIM_EXT_CAPTURE

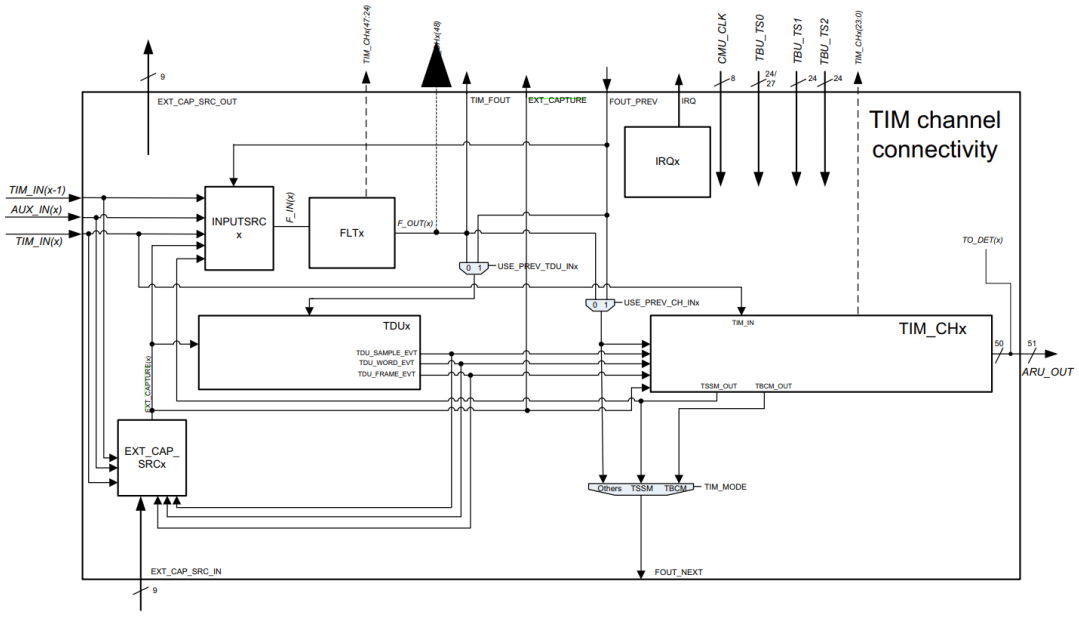

Figure 5.15: TIM Channel Block Diagram[11]

5.3.2 Watchdog Timer Configuration

The implementation of the Watchdog Timer utilised TOM Channel 0, thus changing the source of the reset for all other counters from the reset imposed by the inverter to the reset generated by the Watchdog Timer. In order to guarantee an error tolerance in the transmission of the reset signal, the watchdog window has been configured to 10.1 µs. This ensures that if the reset signal has a periodicity greater than the watchdog window, the system will enter a safe state in which the reset source will be changed. Figure 5.11 illustrates that the TOM channel comprises two compare registers, which can be utilised for the generation of rising and falling edges of a PWM signal or for interrupt generation. The latter function was employed to implement the watchdog mechanism, with the CM0 register defining the reset time window.

Register	No Time-Out Configuration	Time-Out Configuration
RST_CCU0	1	0
TRIGOUT	0	1

Table 5.4: TOM Channel configuration when the time-out event is triggered

In the event of $CN0 = CM0$, the TOM channel generates an interrupt service routine, modifying the internal configuration of Channel 0 as illustrated in Table 5.4. The Pseudo Code 5.17 provides a comprehensive illustration of the configuration of TOM Channel 0 and the interrupt service routine that is executed following a timeout.

```
void initTom_Channel0(void)
{

    TOM_CH0.UpDownMode = UpMode;     // Set TOM Channel 0 counter
   in Up Mode

    TOM_CH0.RSTCCU0 = 1;     // Set TOM Channel 0 to be reset from
   TIM_EXT_CAPTURE

    TOM_CH0.EXT_TRIG = 1;    // Set TOM Channel 0 to be reset from
   TIM_EXT_CAPTURE
```

```
    TOM_CH0.EXTTRIGOUT = 1; // Set TOM Channel 0 to propagate the
   TIM_EXT_CAPTURE reset signal to subsequent counters

    TOM_CH0.TRIGOUT = 0;     // Set TOM Channel 0 to propagate the
    TIM_EXT_CAPTURE reset signal to subsequent counters

    TOM_CH0.CM0 = TimeWindow;   // Configure the Time window to
   10.1 micro seconds

    TOM_CH0.Interrupt = InterruptOnCM0; // Enable interrupt
   generated on CN0=CM0

}

void WATCHDOG_Handler(void)
{

    TOM_CH0.RSR_CCU0 = 0;   // Set TOM Channel 0 to be reset from
   CN0=CM0

    TOM_CH0.TRIGOUT = 1;    // Set TOM Channel 0 to propagate the
   internal reset signal to subsequent counters

    TOM_CH0.CM0 = PWM_Period;   // Set TOM Channel 0 to reset
   subsequent signals to a frequency of 100kHz

}
```

Pseudo Code 5.17: Configuration of TOM Channel 0 for watchdog timer function

5.3.3 PWM For Switching Pole Control

In order to implement the generation of the eight PWMs for controlling the MOSFET in the DC-DC circuit diagram, it is necessary to manage the phase shift between the **ntr1, ntr2, ntr3** and **ntr4** carriers. Furthermore, the centred aligned PWMs must be implemented using counters in Up mode. Finally, the dead time must be introduced between the PWMs and the complementary ones, with the configuration of this being carried out via DTM.

Phase Shift and Duty Cycle Configuration

In order to reproduce the triangular carriers depicted in Figure 4.5, the cascaded configuration of the TOM channels illustrated in Figure 5.11 was employed. In this configuration, **TOM_CH1** and **TOM_CH2** channels implement the **ntr1** and **ntr3** carriers, respectively, while **TOM_CH6** and **TOM_CH7** channels represent the **ntr2** and **ntr4** carriers. Given that the theoretical carriers of the signals are triangular, it is possible to assign the positive vertex of each signal to the centre of the PWM signal part with a high value.
From the carriers ntr1 and ntr3, it can be observed that the distance between the high vertices of the two triangulars corresponds to half a period. Furthermore, from the moment the reset arrives, and therefore the two channels TOM_CH1 and TOM_CH2 are reset, for TOM_CH1 the high vertex of the triangular corresponds to a time of 50 µs from the arrival of the reset, while for TOM_CH2 the point corresponding to the high vertex of the carrier ntr3 corresponds to the instant of time of the arrival of the reset. With regard to the aforementioned points, which identify the apex of the triangular shapes generated by the counters in Up mode, registers CM0 and CM1 will be configured to achieve the desired duty cycle, as illustrated in the following equations:

$$CM_0 = ResetPoint + DutyCycle * 500 \tag{5.4}$$

$$CM_1 = ResetPoint - DutyCycle * 500 \tag{5.5}$$

The definitions of the parameters in equations 5.4 and 5.5 are provided below:

- **ResetPoint** indicates the equivalent arrival time of the reset in ticks.
- **DutyCycle** represents the computed duty cycle in percentage.

With regard to the carriers ntr2 and ntr4, it can be observed from Figure 4.5 that these are out of phase with each other by half a period. However, they are out of phase with ntr1 by a quarter period and three-quarters period, respectively. In order to implement the aforementioned phase shifts correctly, the counter TOM_CH5 must be employed in addition. The function of this counter is to reset the channels TOM_CH6 and TOM_CH7 when $CN0 = 249$, which is equivalent to one quarter period. At this subsequent moment in time, both counters TOM_CH6 and TOM_CH7 will be reset and thus commence counting from $CN0 = 0$. With regard to TOM_CH6, the centre of the PWM generated by it is 50 µs from the arrival of the reset from TOM_CH5, while for TOM_CH7 it is 0 seconds. The same logic is employed to configure the duty cycles for ntr2 and ntr4 as for ntr1 and ntr3.

Dead Time Implementation

In order to effectively regulate the power flows within each switching pole, thus preventing the occurrence of the shoot-through phenomenon, it is essential to implement two distinct PWMs, with a defined interval of dead time between them of 200 ns. In order to perform the aforementioned task, the Dead Time Module was employed, whereby a PWM was taken as an input and generated the complementary one with the desired dead time. In the following application, two clusters of the Dead Time Module were employed, DTM0 and DTM1, respectively.
Cluster DTM0 is internally connected to channels TOM_CH1 and TOM_CH2, and generates the signals **ntr1_n** and **ntr3_n**. These, in conjunction with the two signals generated by the two TOM channels, provide the four PWM signals required for switching poles 1 and 3. Similarly, DTM1 is connected internally to the TOM_CH6 and TOM_CH7 channels, which generate the two PWM signals **ntr2_n** and **ntr4_n**, thereby completing the generation of the signals required to

control switching poles 2 and 4. The Pseudo Code 5.18 illustrates the configuration procedure of the DTM module.

```
void initDtm(void)
{

    GTM_enable();   // Function to enable the GTM

    DTM_initConfig();   // DTM initialization with default
   parameters

    DTM_RisingDelay = 200ns // DTM rising delay of 200ns
   configuration

    DTM_FallingDelay = 200ns // DTM falling delay of 200ns
   configuration

    TIM_init();

}
```

Pseudo Code 5.18: Configuration of DTM

Enhanced Versatile Analog-to-Digital Converter (EVADC) Trigger Configuration

In order to configure the EVADC module channels to be triggered via the TOM channels, it is first necessary to refer to the internal interconnections between the TOM and EVADC, which are detailed in the micro-controller's user manual [18]. From the data extracted from the internal configuration of the AURIX TC39, it can be discerned that the channels of the TOM that are suitable for hardware triggering of the EVADC conversion are listed below:

- **TOM0 Channel 3**
- **TOM0 Channel 4**

- **TOM0 Channel 6**
- **TOM0 Channel 7**

In the configuration of the TOM channels as depicted in Figure 5.11, it can be observed that beyond the channel TOM_CH5, the reset for the channels is provided in a shifted manner with respect to the one provided by the inverter, as elucidated in Section "Phase Shift and Duty Cycle Configuration". In order to utilise the TOM channels for triggering the EVADC in phase with the EXT_RESET signal, it was determined that the only two available channels, TOM_CH3 and TOM_CH4, immediately preceding TOM_CH5, should be utilised. The two TOM channels used to trigger the ADC conversions have been assigned to the quantities shown in Table 5.5.

TOMO Channel	**Physical quantities**
Channel 3	i_1, i_3, v_{dcP}
Channel 4	$i_2, i_4, v_{dcN}, v_{bat}$

Table 5.5: Assignment of triggers for the physical quantities to be sampled.

In order to ensure that the acquisitions are triggered according to the timing provided in Figure 4.6, it is necessary to check that the first sample is taken by TOM_CH4 at 2.5 µs after the start of the period.
Following the setting of the two TOM channels for triggering, the first edge from the moment the channels are first activated is executed by TOM_CH3, which would not allow the sequence described above. For this purpose, the TGC is used to enable TOM_CH3 only after TOM_CH4 has generated the first edge for sampling. Specifically, TOM_CH4 generates an interrupt each time it generates the first edge of the period, and this is iterated until the EVADC module is configured. From the moment EVADC has been configured in the software start-up and TOM_CH4 generates the interrupt, the signal produced by TOM_CH4 will be enabled within the corresponding ISR in a synchronous manner, i.e. at the beginning of the next period. In the following period, TOM_CH4 generates an interrupt again when this time it performs the first actual trigger on the EVADC, in the ISR the output of TOM_CH3 will be enabled, this time asynchronously, so that when TOM_CH3

generates the next edge, it will propagate to the EVADC generating the trigger. This configuration mechanism is explained in detail in Pseudo Code 5.19.

```
void Tom0_Ch4IsrHandler (void)
{

    // Verifies that EVADC has been initialised
    if(Evadc_is_configured == 0)
    {
        TOM0_CH4.clearFlag();   // Reset of Interrupt flag
    }

    // Verifies that EVADC has been initialized and TOM0_CH4
   output is not enabled
    else if(Evadc_is_configured == 1 && TOM0_TGC0.
   OutEnableSynchCH4 == 0)
    {
        TOM0_TGC0.OutEnableSynchCH4 = 2; // Synchronously enables
   TOM0_CH4 output

        TOM0_CH4.clearFlag(); // Reset of Interrupt flag
    }

    // Verifies that EVADC has been initialized and TOM0_CH4
   output is enabled
    else if(Evadc_is_configured == 1 && TOM0_TGC0.OutEnableSynch
   == 2)
    {
        TOM0_TGC0.OutEnableAsynchCH3 = 2;   // Asynchronously
   enables TOM0_CH3 output
    }
}
```

Pseudo Code 5.19: Definition of the ISR for the TOM_CH3 and TOM_CH4 channel inizilisation mechanism to trigger on sampling

Subsequently, the TOM channels were configured to provide the requisite rising and falling edges at the precise points where sampling was to be performed. As reported in Chapter 4, it is necessary to consider the delay introduced by the current sensors, as it does not provide voltage values corresponding to the same instant of time in which the current sampling is carried out. In order to achieve this, the rising and falling edges that determine the triggers for sampling have been shifted by an amount **Td_AD**, corresponding to 500 ns. This allows the voltage measurements corresponding to the sampling of currents at the vertices of the triangular carriers to be acquired. The configuration for the subsequent task is presented in Pseudo Code 5.20, which illustrates the procedure for configuring channels TOM_CH3 and TOM_CH4 to generate the sampling triggers for the EVADC.

```
void initTom_SamplingChannels(void)
{

    // TOM0 Channel 3 configuration

    TOM_CH3.UpDownMode = UpMode;    // Set TOM Channel 3 counter
   in Up Mode

    TOM_CH3.RSTCCU0 = 1;    // Set TOM Channel 3 to be reset from
   Trig[2]

    TOM_CH3.EXT_TRIG = 0;   // Set TOM Channel 3 to be reset from
   Trig[2]

    TOM_CH3.EXTTRIGOUT = 0; // Set TOM Channel 3 to propagate the
   Trig[3] reset signal to subsequent counters

    TOM_CH3.TRIGOUT = 0;     // Set TOM Channel 3 to propagate the
    Trig[3] reset signal to subsequent counters

    TOM_CH3.CM0 = 5_micro_sec + Td_AD;  // Configure the first
   trigger point
```

```
    TOM_CH3.CM1 = 0_micro_sec + Td_AD;  // Configure the second
   trigger point

    // TOM0 Channel 4 configuration

    TOM_CH4.UpDownMode = UpMode;    // Set TOM Channel 4 counter
   in Up Mode

    TOM_CH4.RSTCCU0 = 1;    // Set TOM Channel 4 to be reset from
   Trig[3]

    TOM_CH4.EXT_TRIG = 0;   // Set TOM Channel 4 to be reset from
   Trig[3]

    TOM_CH4.EXTTRIGOUT = 0; // Set TOM Channel 4 to propagate the
   Trig[4] reset signal to subsequent counters

    TOM_CH4.TRIGOUT = 0;     // Set TOM Channel 4 to propagate the
    Trig[4] reset signal to subsequent counters

    TOM_CH4.CM0 = 2.5_micro_sec + Td_AD;    // Configure the first
    trigger point

    TOM_CH4.CM1 = 7.5_micro_sec + Td_AD;    // Configure the
   second trigger point

    TOM_CH4.Interrupt = InterruptOnCM0; // Enable interrupt
   generated on CN0=CM0

}
```

Pseudo Code 5.20: Configuration of TOM_CH3 and TOM_CH4

Control Software for Duty Cycle Update configuration

The final stage of the Power Structure Control Task entails the adaptation of the previously developed control function to the AURIX TC39. In order to complete this task, a series of points relating to the behaviour of the microcontroller were analysed. These included the triggering of the control task, measuring the worst-case execution time, as the deadline of 8 µs must be observed, and checking the duty cycle updates synchronously on the control PWMs.

In order to achieve this objective, the optimal point at which to execute the control task was identified. In accordance with the specifications of the application, the control task is to be initiated once the four sampling points indicated in Figure 4.8 have been acquired.
To this end, the time taken by the EVADC to carry out two consecutive samplings at 5 µs intervals was initially measured, utilising the configuration of the EVADC that employs the DMA mechanism to execute a transaction comprising two transfers: the initial transfer corresponding to the initial sampling and the subsequent transfer corresponding to the subsequent sampling, i.e. the final sampling.
To measure this time interval, the AURIX TC39's internal system timer was employed. Following the initial sampling, the system timer reading was obtained, while the subsequent reading was taken within the interrupt service routine at the conclusion of the DMA transaction. In order to generate the trigger for the conversions, the TOM_CH3 was employed in one-shot mode, with the intention of creating two fronts 5 µs apart following the trigger provided via software. Once the optimal point for executing the control task had been determined, the TOM_CH8 was configured to trigger the task in accordance with the condition $CN0 = CM0$

Once the optimal point at which to call the control task was determined, the task was executed on the microcontroller to ascertain whether the execution time exceeded the deadline of 8 µs. In order to obtain the most critical measurement, the time was recorded in the system's **normal operational mode**, which is the most computationally demanding.

Finally, a simulation was conducted to verify the duty cycle updates in synchronous mode. The control task was invoked every 10 µs at the previously established optimum point, and the duty cycle of the four PWMs associated with the four triangular carriers was increased by 10% with each call of the task. This method enabled the determination that the duty cycles update correctly with a periodicity of 10 µs, in accordance with the requirement.

Chapter 6

Experimental Results

6.1 Communication System Validation

6.1.1 Controller Area Network (CAN) Validation

As explained in chapter 5.1, a **DBC** was defined containing the characteristics of the CAN messages exchanged between the two modules CAN0 and CAN1. In order to verify the communication system, several CAN messages were sent from among those in the DBC; in particular, messages were sent in the order and with the contents shown in Table 6.1.

CAN Message Tx Order	**Message Content**
Message 1	0x55555555 0xABCDEFAB
Message 2	0x24ACDA
Message 3	0x34AB38BC
Message 1	0x55555555 0xABCDEFAB
Message 2	0x24ACDA
Message 1	0x55555555 0xABCDEFAB

Table 6.1: CAN Message Transmission Order and Content

As a result of the following test it is expected that, after transmission, the array **RxData1** will contain 3 CAN messages of type **CAN_Message_1**, while the array **RxData2** will contain 2 CAN messages of type **CAN_Message_2** and finally the buffer **RxData3** will contain only one CAN message of type **CAN_Message_3**. First, the accuracy of the actual Baud rate was checked against the theoretical one. Figure 6.1 shows the Baud rate measurement using an oscilloscope.

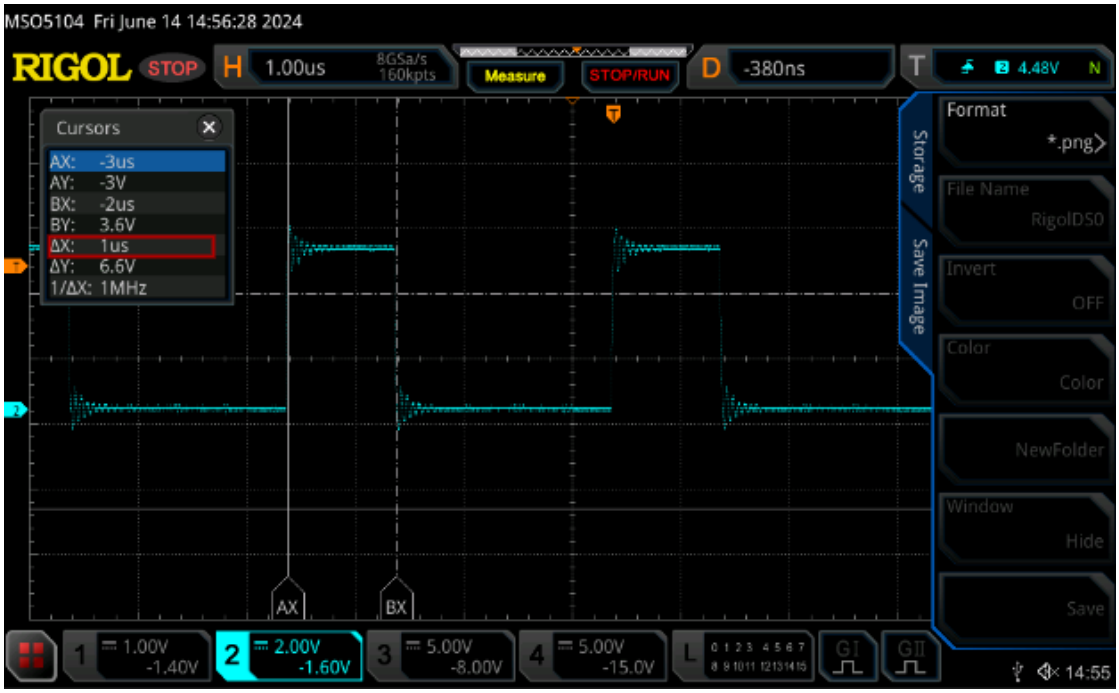

Figure 6.1: Measurement of the accuracy of teh CAN transmission

The result of the test carried out on the micro-controllers in communication, configured in the way illustrated in Figure 6.2 is reported in Table 6.2, from which it is possible to note the correct acquisition of the messages on the different buffers, demonstrating that the transmission and reception mechanism, and in particular the acceptance filtering, works correctly.

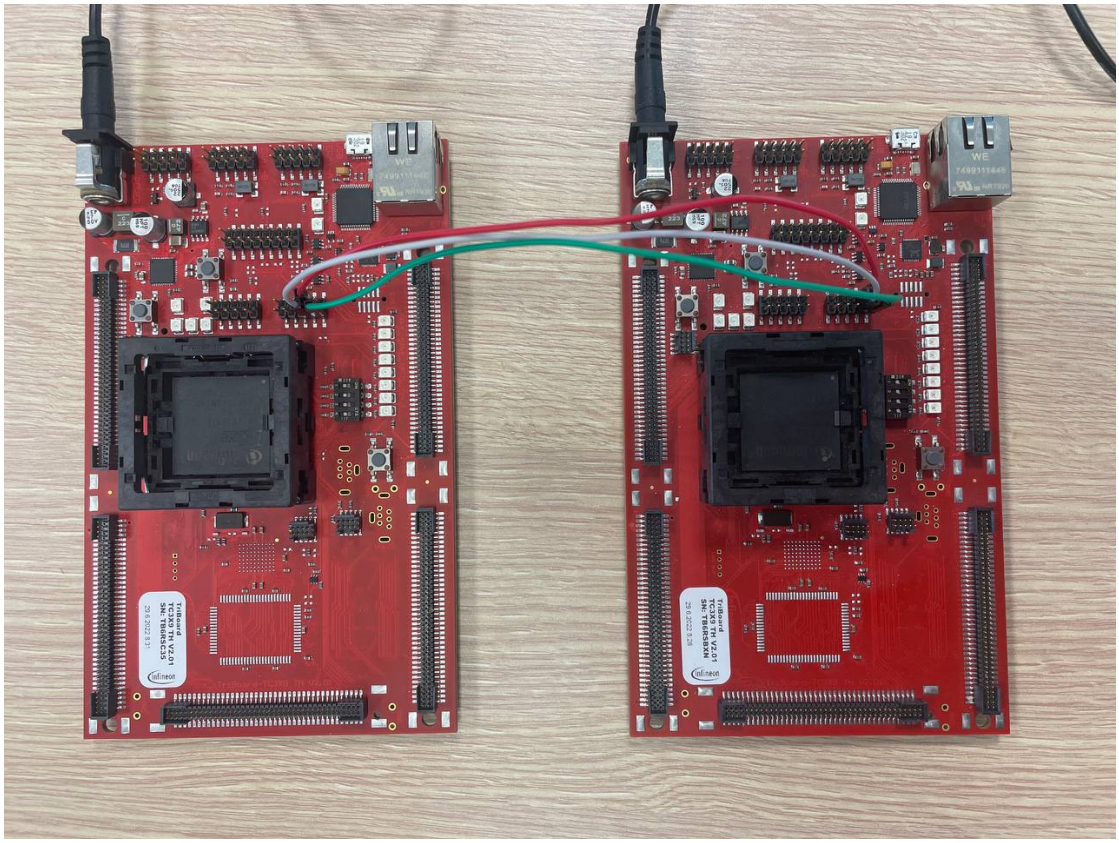

Figure 6.2: Configuration of two Triboard TC3x9 to test CAN communication

RxData	First 32 bit of the buffer	Last 32 bit of the buffer
RxData1[0]	0x55555555	0xABCDEFAB
RxData1[1]	0x55555555	0xABCDEFAB
RxData1[2]	0x55555555	0xABCDEFAB
RxData2[0]	0x24ACDA	//
RxData2[1]	0x24ACDA	//
RxData2[2]	//	//
RxData3[0]	0x34AB38BC	//
RxData3[1]	//	//
RxData3[2]	//	//

Table 6.2: contents of registers RxData1, RxData2 and RxData3 following CAN communication

6.1.2 Universal Asynchronous Receiver Transmitter (UART) Validation

To validate the UART communication mechanism between the two micro-controllers, the correct transmission of a single data packet was first verified to measure the accuracy of the transmission baud rate. The next step was to validate the synchronisation mechanism implemented to make the communication more robust in the event of a transmission interruption. For both tests, the two TC3x9 Triboards were configured as shown in Figure 6.3.

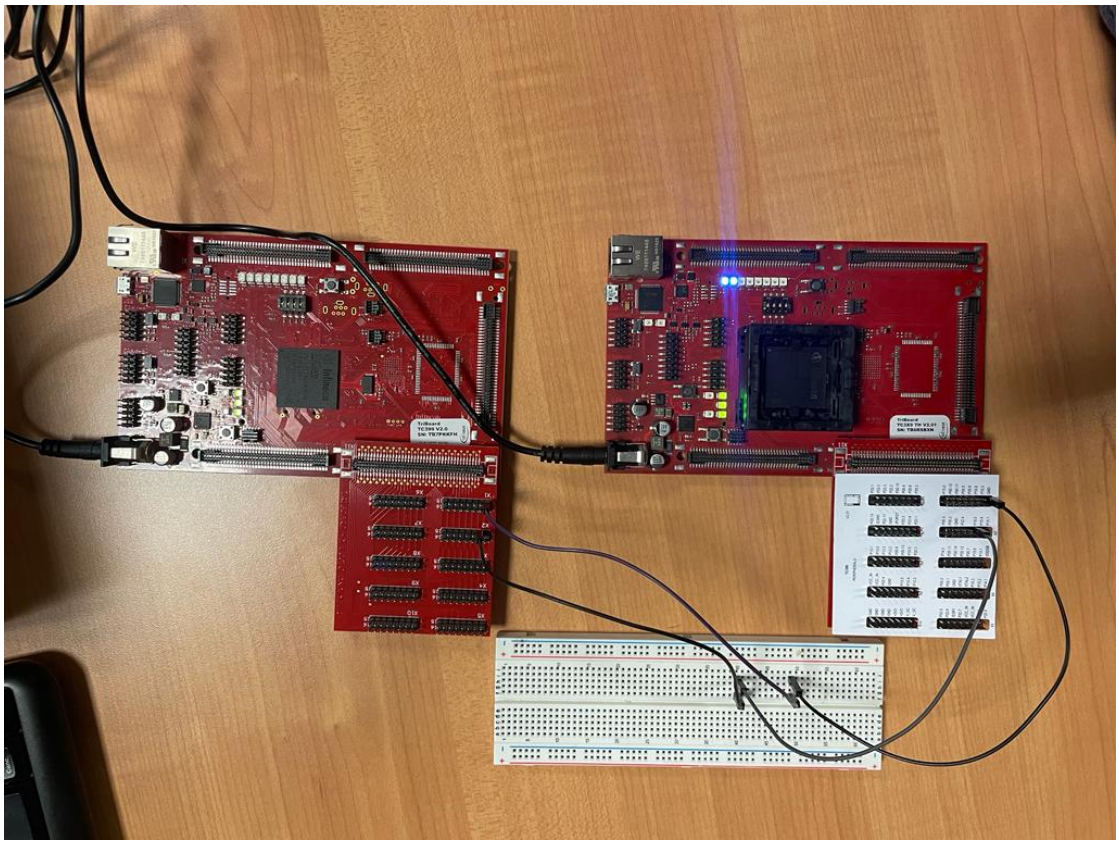

Figure 6.3: Configuration of two Triboard TC3x9 to test UART communication

Verification of transmission accuracy

To measure the accuracy of the transmission baud rate, a data packet was sent and the transmission was recorded on an oscilloscope to make the following measurement. Figure 6.4 shows the measurement of the duration of a single bit, which was found to be 125 ns, corresponding to the theoretical transmission time of 8Mbps.

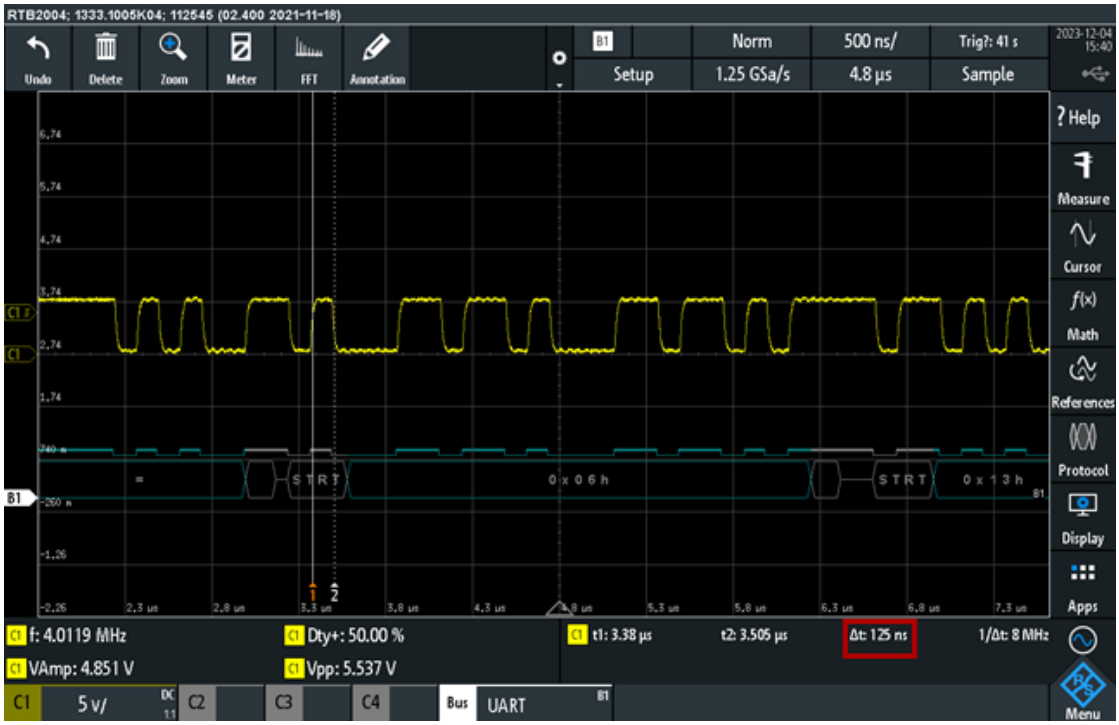

Figure 6.4: Measurement of UART transmission time accuracy

Synchronization Mechanism Validation

Two AURIX TC39s were used to validate the communication mechanism in order to make the system more robust in the event of communication line failures, one to simulate the inverter, which acts as the transmitter, and the other for the DC-DC, which acts as the receiver. For validation, the micro-controller managing the DC-DC converter was used in debug mode, while a logic analyser was used to verify transmission by the micro-controller managing the inverter.

The following test identifies two receive buffers for the DC/DC converter:

- **RxBuffer**: destination buffer of the DMA transfer, which takes data from the RxFIFO.
- **RxControlBuffer**: buffer used in the control task containing the last complete UART transmission.

In the following scenario, two data packets were sent in the order shown in Table 6.3

UART Message Tx order	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Data Package 1	0x23	0x43	0x12	0x28	0x01
Data Package 2	0x11	0x12	0x13	0x14	//

Table 6.3: UART Message Transmission Order and Content

As can be seen, packet 2 consists of 4 bytes, so that a communication interruption due to a malfunction is simulated. Figure 6.5 shows the communication acquired via the Logical Analyser.

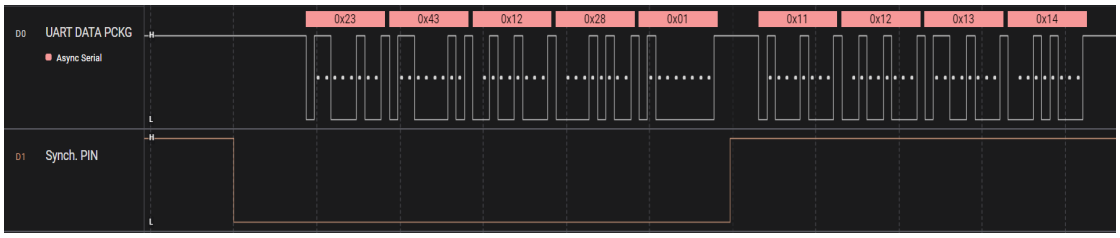

Figure 6.5: Acquisition of UART communication via logical analyser. The top shows the two transmitted data packets, while the bottom shows the status of the synchronisation signal toggling before the start of a packet transmission

As a result of the communication, it is expected that the RxControlBuffer is equal to the content of the first packet sent, since, as 5 bytes have not arrived since the last communication, the content of packet 1 in the RxControlBuffer has not been replaced with the partially sent packet 2. Figure 6.6 shows the last 4 transfers of the DMA in RxBuffer, the last content of the buffer is the last element of packet 1 as it was not overwritten.

```
Name : [0]
    Default:'□'
    Decimal:17
    Hex:0x11
    Octal:0c0021
    Binary:0b00010001

Name : [1]
    Default:'□'
    Decimal:18
    Hex:0x12
    Octal:0c0022
    Binary:0b00010010

Name : [2]
    Default:'□'
    Decimal:19
    Hex:0x13
    Octal:0c0023
    Binary:0b00010011

Name : [3]
    Default:'□'
    Decimal:20
    Hex:0x14
    Octal:0c0024
    Binary:0b00010100

Name : [4]
    Default:'□'
    Decimal:1
    Hex:0x01
    Octal:0c0001
    Binary:0b00000001
```

Figure 6.6: RxBuffer content after the transmission of the two Data Packages

Figure 6.7 shows the content in RxControlBuffer, which as can be seen respects the expected behaviour in that it contains the first transmitted data packet, as the latter is complete.

```
Name : [0]
    Default:'#'
    Decimal:35
    Hex:0x23
    Octal:0c0043
    Binary:0b00100011

Name : [1]
    Default:'C'
    Decimal:67
    Hex:0x43
    Octal:0c0103
    Binary:0b01000011

Name : [2]
    Default:'□'
    Decimal:18
    Hex:0x12
    Octal:0c0022
    Binary:0b00010010

Name : [3]
    Default:'('
    Decimal:40
    Hex:0x28
    Octal:0c0050
    Binary:0b00101000

Name : [4]
    Default:'□'
    Decimal:1
    Hex:0x01
    Octal:0c0001
    Binary:0b00000001
```

Figure 6.7: RxControlBuffer content after the transmission of the two Data Packages

6.2 External Clock Task Verification

In order to verify the correct configuration of the **System PLL**, two applications were developed, Application 1 and Application 2, respectively, as defined in 5.2. For both applications, a 10 MHz input clock signal to the microcontroller with a voltage between 0V and 2V was used, as illustrated in Figure 6.8. The two results obtained from the two validation methods are presented below.

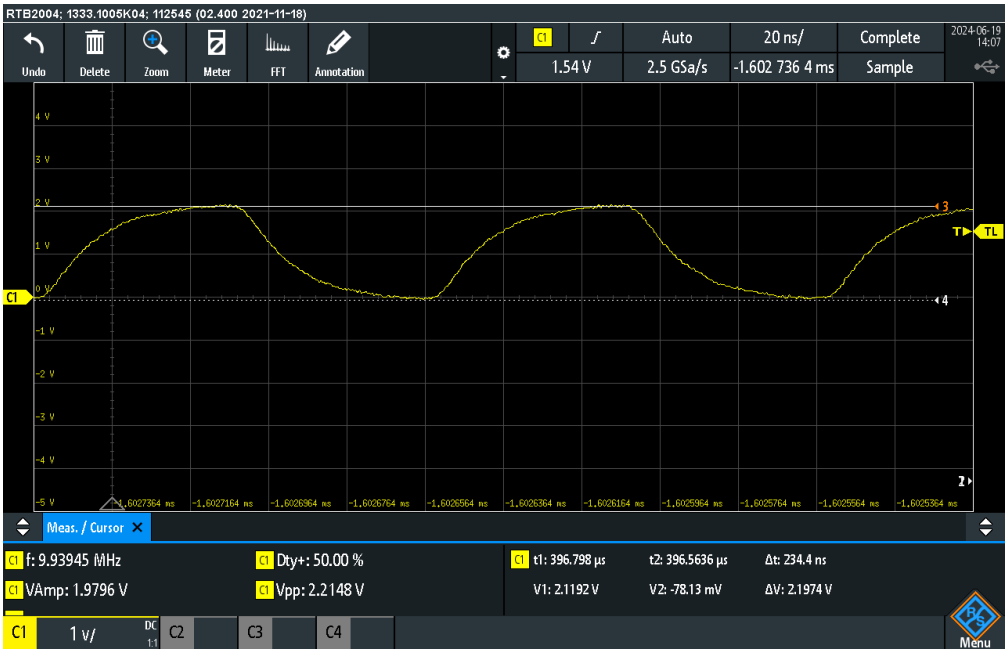

Figure 6.8: External clock signal at 10 MHz and amplitude between 0V and 2V

6.2.1 Results of Application 1

The objective of the subsequent application is to validate the configuration of the SYSTEM PLL utilising the micro-controller in **External Clock Output Mode**. By measuring the generated output signal f_out, it is possible to determine the clock frequency generated by the SYSPLL. To ensure an accurate measurement of the frequency in question, the oscilloscope was employed. The resulting measurement of the output signal is illustrated in Figure 6.9, which depicts a frequency of approximately 1 MHz on the f_out signal. From the measured frequency, it is possible to determine the frequency f_source0 using the formula defined in section 5.2.3:

$$f_{source0} = f_{out} * SPB_{DIV} * n * 2 = 995.149 * 10^3 * 3 * 50 * 2 = 298.5MHz \quad (6.1)$$

Figure 6.9: Output signal produced in External Clock Output Mode

6.2.2 Application 2 Results

In order to generate a PWM signal of 2 kHz, it was necessary to configure the compare register to guarantee the theoretical clock frequency of the TOM CMU_FXCLK0, which is 100 MHz. The value used by the compare register to determine the period is therefore $\frac{100MHz}{2kHz} = 50{,}000$ ticks. From the measurement of the PWM signal frequency shown in Figure 6.10, the frequency was found to be 1.99 kHz, from which is possible to compute the clock frequency of the TOM:

$$CMU\_FXCLK0 = 1.99 * 10^3 * 50000 = 99.5MHz \tag{6.2}$$

From the following frequency, it is possible to derive the frequency f_GTM:

$$f_{GTM} = 2 * CMU\_FXCLK0 = 2 * 99.5MHz = 199MHz \tag{6.3}$$

The same f_GTM is derived from the frequency f_spb, according to the following relationship:

$$f_{GTM} = 2 * f_{spb} = \frac{2 * f_{source0}}{3} \tag{6.4}$$

The inverse of the equation 6.4 yields the following result:

$$f_{source0} = \frac{f_{GTM} * 3}{2} = \frac{199MHz * 3}{2} = 298.5MHz \tag{6.5}$$

, which corresponds to the frequency also obtained in Application 1

Figure 6.10: Output signal produced by GTM in External Input Clock Mode

From the results provided by **Application 1** and **Application 2**, it is possible to trace the frequency of the input signal to the microcontroller. Exploiting the configuration of the SYSPLL, the following result was obtained:

$$f_{osc} = \frac{f_{source0} * 2}{60} = 9.95MHz \tag{6.6}$$

. The following result shows that from the input signal to the micro-controller, although there is a deviation in the signal between the desired frequency and the actual frequency of 50kHz, the micro-controller is still able to function correctly, guaranteeing the correct operation of the modules inside it.

6.2.3 Verification of Backup Clock Triggering

In order to verify the correct intervention of the backup clock in the event of a fault in the input clock signal to the microcontroller, the clock signal was voluntarily interrupted in the case of Application 1. As the backup clock is at a frequency of 100MHz, it is expected that the frequency of the signal f_out will be as follows:

$$f_{out} = \frac{f_{back}}{SPB_DIV * n * 2} = \frac{100 * 10^6}{3 * 50 * 2} = 333.3kHz \tag{6.7}$$

Figure 6.11 shows the output signal f_out in the fault situation where the external clock fails. As can be seen from the measurement taken, the frequency of the f_out signal is 335.8 kHz, resulting in a slight deviation from the theoretical value of approximately 3kHz.

Figure 6.11: Output signal Produced in External Clock Output Mode when Backup Clock is active

To confirm that the backup clock is effectively active, the CLK_SEL bit is monitored, which determines which of the two clocks **f_pll0** and **f_back** will be distributed across all modules in the form of **f_source0**. Table 6.4 shows the definition of the CLK_SEL bitfield. First, the CLK_SEL bitfield was observed during correct system operation, i.e. in the situation where the input clock is present and that it meets the system requirements, obtaining from debugging the result shown in Figure 6.12.

	BIT	Description
CLK_SEL	00	f_{BACK} is used as clock source $f_{source0}$, f_{src1}, and $f_{source2}$
	01	f_{PLL0} is used as clock source $f_{source0}$; f_{PLL1} is used as clock source f_{src1}; f_{PLL2} is used as clock source $f_{source2}$

Table 6.4: Definition of CLK_SEL bitfield

Name	Type	Value
'extclk.c'::clk_sel	unsigned long	1

Figure 6.12: CLK_SEL bitfield status when input clock frequency meets requirements

Once the input clock is removed from the microcontroller, the following bit is automatically changed, making the switch from frequency f_pll0 to f_back, as shown in Figure 6.13.

Name	Type	Value
'extclk.c'::clk_sel	unsigned long	0

Figure 6.13: CLK_SEL bitfield status when f_back clock is activated

6.3 Power Structure Control Task Validation

6.3.1 Phase shift between the four carriers

The initial validation of the power structure control task was conducted on the configuration of TOM channels 1, 2, 6 and 7. This was done to verify that the phase shifts between the counters complied with the specifications shown in Figure 4.5. In order to validate the correctness of the phase shift, groups of two signals were acquired by oscilloscope using different duty cycles for each measurement. This was done in order to verify that the phase shifts are always made with respect to the centre of the time interval in which the signals have a high value.
Figure 6.14 illustrates the signals ntr1 (yellow) and ntr2 (green), which demonstrate a phase shift corresponding to the theoretical value of 2.5 µs. The duty cycles of ntr1 and ntr2 are 60% and 20%, respectively.

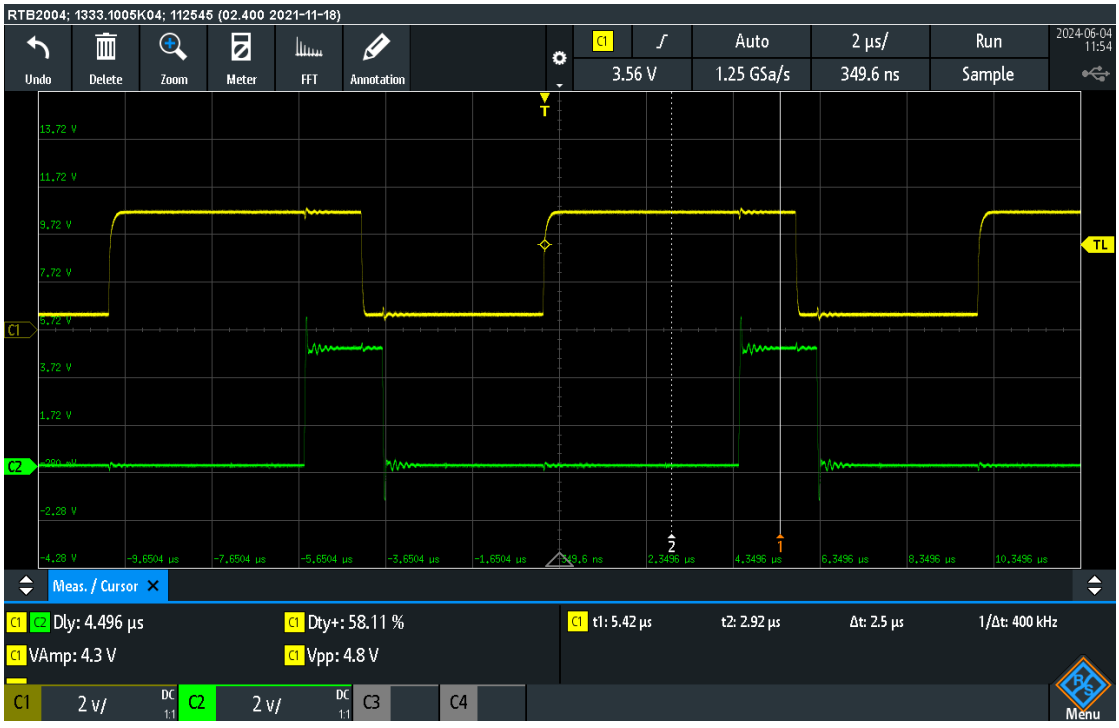

Figure 6.14: phase shift between **ntr1** and **ntr2** acquired by oscilloscope. From the measurements, $\Delta t = 2.5\,\mu s$

Figure 6.15 illustrates the measurement conducted on signals ntr1(yellow) and ntr3(green), with duty cycles of 26% and 36% respectively, and with a phase shift of half a period, i.e. 5 µs.

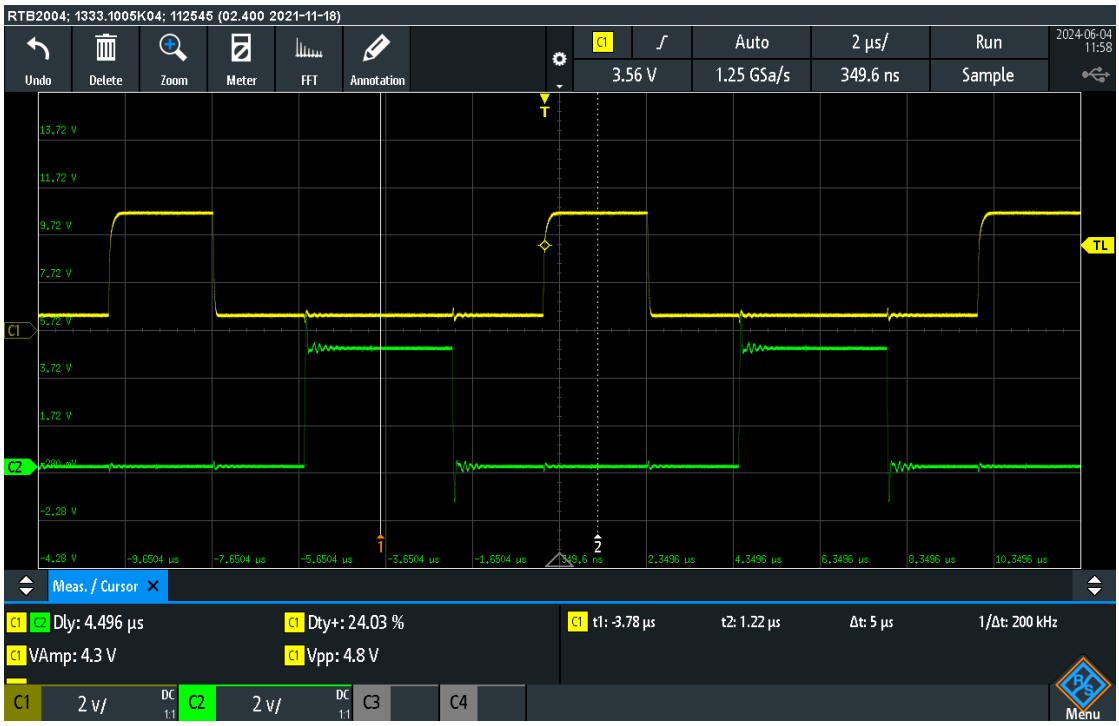

Figure 6.15: phase shift between **ntr1** and **ntr3** acquired by oscilloscope. From the measurements, $\Delta t = 5\,\mu s$

Finally, the signals ntr1 (yellow) and ntr4 (green) were measured with duty cycles of 72% and 60%, respectively. The acquisition of these signals is shown in Figure 6.16. From this figure, it can be verified that the phase shift is 2.5 µs, which is in the opposite direction to the phase shift between ntr1 and ntr2.

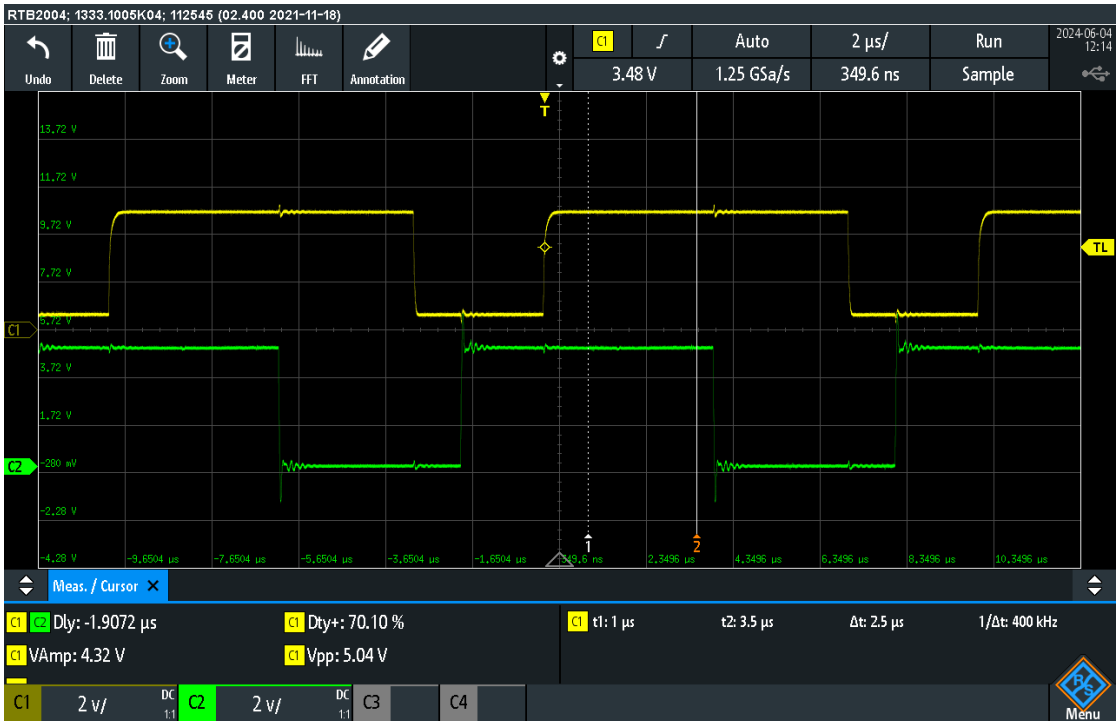

Figure 6.16: phase shift between **ntr1** and **ntr4** acquired by oscilloscope. From the measurements, $\Delta t = 2.5\,\mu s$

6.3.2 Dead Time measurement

Once the configuration of the channels for the phase shift had been validated, the DTM was enabled in order to verify that, in addition to the dead time, the total eight PWMs for controlling the MOSFETs met the specified requirements. Firstly, the dead time between the **ntr1** and **ntr1_n** signals was measured using an oscilloscope. The acquisition of this is shown in Figure 6.17, and it is precisely equal to the theoretical value, namely 200 ns.

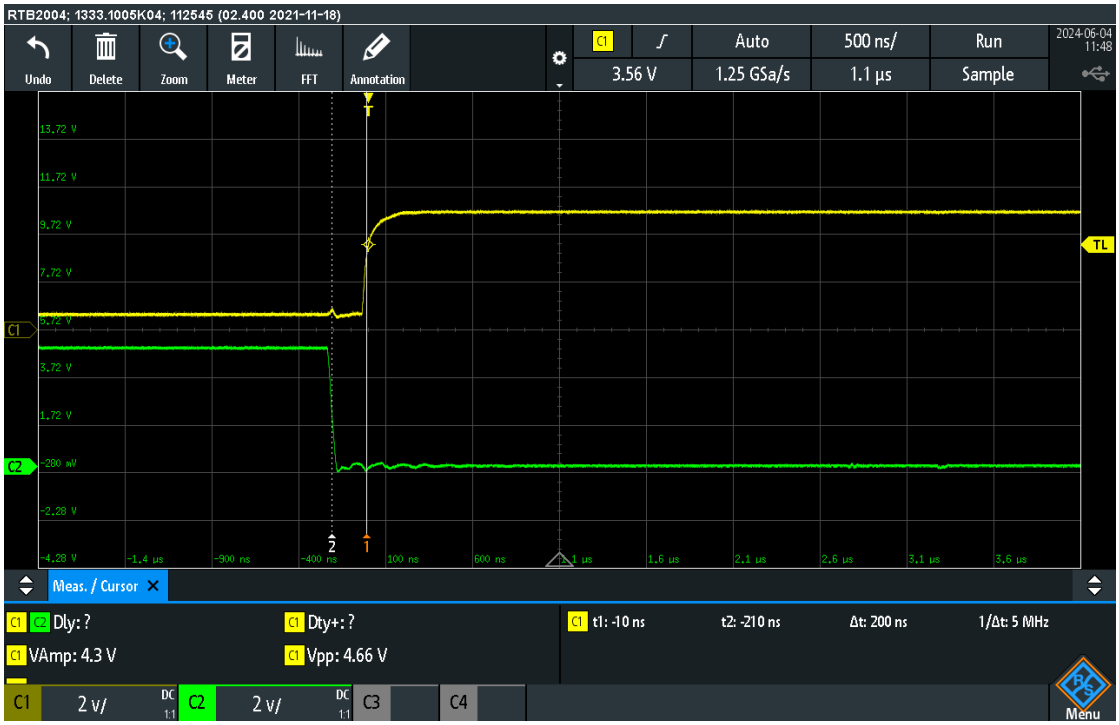

Figure 6.17: Dead Time Measurement

The 8 PWMs for controlling the 8 MOSFETs were then acquired by means of a logical analyser, which are shown in Figure 6.18. One aspect to be taken into consideration is the error on the duty cycle introduced by the dead time; in fact, following the dead time, the time interval in which the signal has a high value is less than 100 ns on each edge, for a total of 200 ns. This reduction corresponds to 2% less duty cycle than the theoretical one, as shown in Figure 6.19, in which it can be seen that the actual duty cycle is 58% while the theoretical duty cycle is 60%.

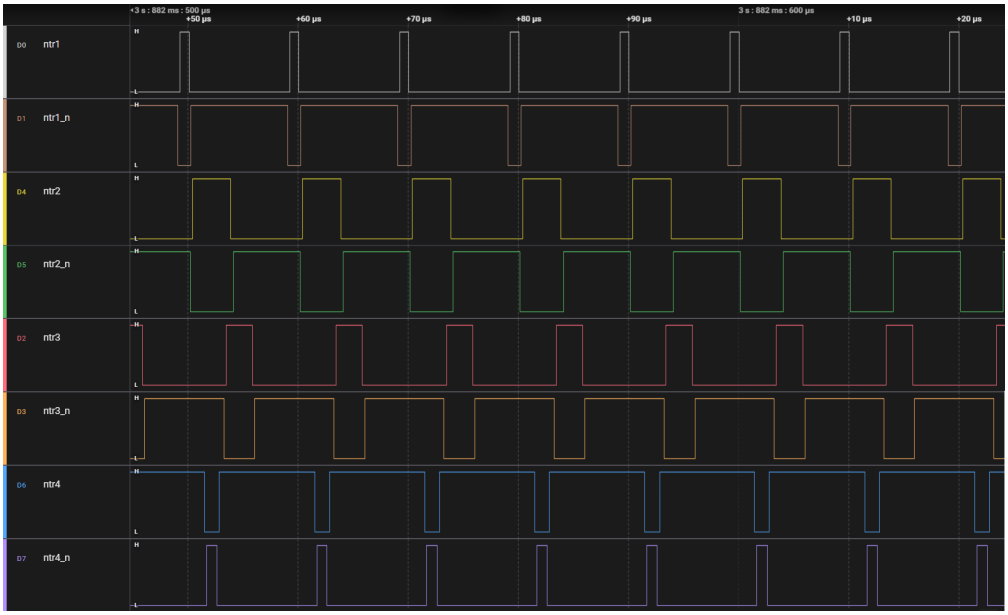

Figure 6.18: Acquisition of the 8 PWMs for controlling the DC-DC converter

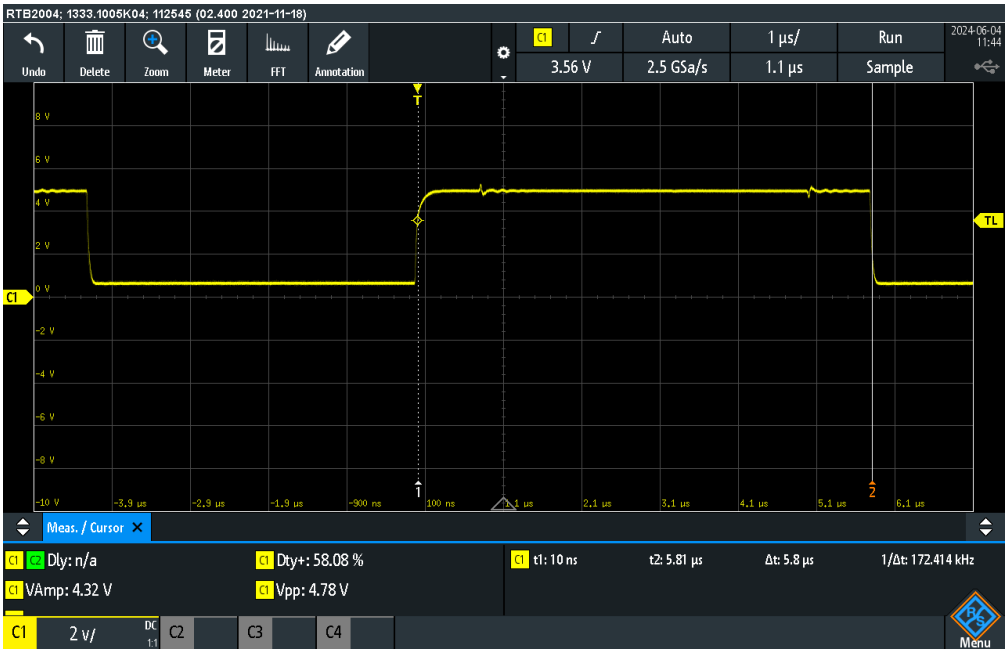

Figure 6.19: Measurement of the 2% duty cycle variation due to the presence of dead time

6.3.3 EVADC Trigger Validation

To verify the initialisation mechanism of the TOM CH3 and CH4 channels for triggering the EVADC converter in order to guarantee the correct sampling sequence, the logical analyser was used to acquire the signals produced by TOM_CH3 and TOM_CH4 following the arrival of the reset signal from the inverter. Figure 6.20 shows the initialisation sequence of the triggers for sampling, from which it can be seen that the first edge is generated by TOM_CH4, thus guaranteeing the correct acquisition sequence. The same figure shows that the second trigger, the one performed by TOM_CH3, occurs 2.5 µs after the first trigger, thus meeting the requirements.

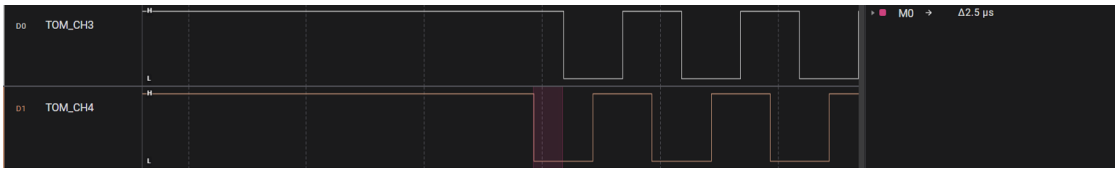

Figure 6.20: Representation of the trigger sequence for EVADCs from channels TOM_CH3 and TOM_CH4 following the first arrival of the external reset signal.

6.3.4 Control Task Verification

The first validation of the control task was carried out at the optimal point of execution of the control ISR. Using the technique described in Chapter 5.5, the time between the first conversion carried out by TOM_CH3 and the end of the DMA transaction corresponding to the samplings also associated with TOM_CH3 was measured. Figure 6.21 shows the two acquisitions made on the System Timer, respectively tick0 corresponding to the time of the first sampling and tick2 corresponding to the end of the DMA transaction.

'Evadc_config.c'::tick2	long long	1660315326
'Tom_config.c'::tick0	long long	1660314738

Figure 6.21: The time interval between the initial trigger and the conclusion of the DMA transaction is to be quantified utilising the methodology delineated in Method 1.

Performing the difference between tick2 and tick0 yields a tick interval of 588, which corresponds to 5.88 µs since the System Timer has a clock of 100 MHz. The same measurement is shown in Figure 6.22, performed with logical analyser, which is slightly different due to the lower accuracy of the device.

Figure 6.22: The time interval between the initial trigger and the conclusion of the DMA transaction is to be quantified utilising the methodology delineated in Method 2.

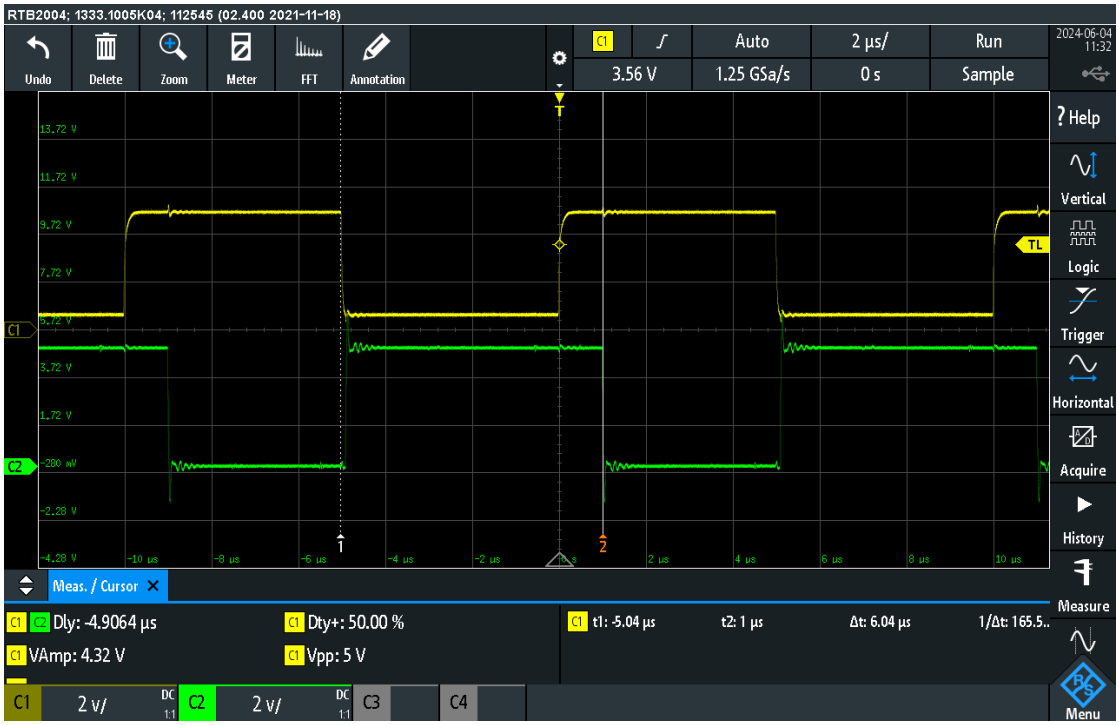

Figure 6.23: Oscilloscope measurement of the time between the first sampling by TOM_CH3 and the call to the control task. As can be seen from the measurement, the value Δt corresponds to the theoretical value, i.e. 6 µs

Following the measurement of the time elapsed between the initial sampling and the conclusion of the DMA transaction, the start time of the control task was set at 6 µs from the initial sampling. This was done in order to ensure that, upon the invocation of the task, all four acquisitions are available on the buffers utilized within the task. Figure 6.23 illustrates the temporal interval between the initial sampling event and the initiation of the control task, which is represented by the falling edge generated by **TOM_CH8**. Once the call point of the control algorithm was determined, it was analysed the worst case execution time among the various possible states of the DC-DC converter.

The measured execution times associated with each state were entered in Table 6.5, from which it is possible to identify the state that provides the worst case execution time, i.e. the GO state. From the following analysis, it can be concluded that the AURIX TC39 micro controller is capable of executing the control task within the imposed dead time of 8 µs.

STATE	**Average Execution Time**
RESET	4.95 µs
START UP	4.75 µs
START UP CC	4.35 µs
GO	5.09 µs

Table 6.5: Measurement of the execution times for each state of the DC-DC converter

Finally, the control task was analysed in order to verify that the synchronous update of the duty cycles functions correctly. In order to achieve this, the control task was configured in such a way that the duty cycles within it are increased by 10% each time the task is executed. Figure 6.24 illustrates the sequence of updates for the four PWM carriers, ntr1, ntr2, ntr3 and ntr4, respectively.

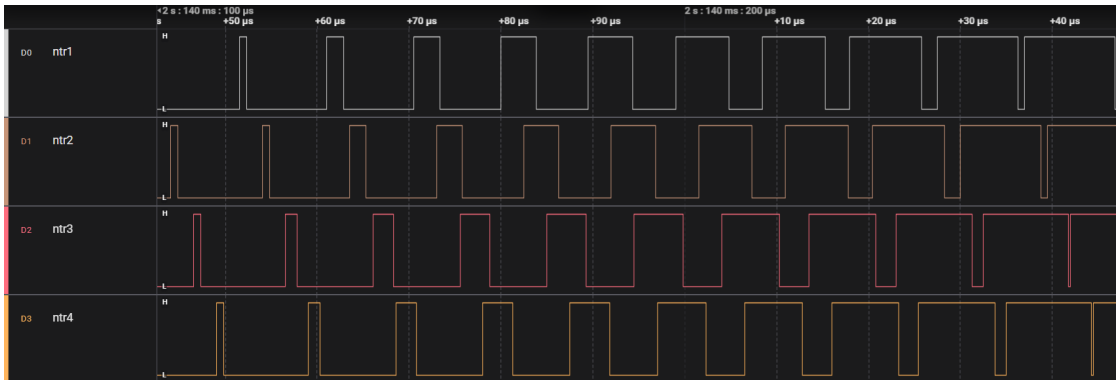

Figure 6.24: Synchronous duty cycle update of the 4 main PWMs carriers for controlling the MOSFETs of the DC-DC converter

Chapter 7

Conclusion and future works

The AURIX TC39 micro-controller, due to its internal architecture and the sub-modules it contains, represents an optimal solution for the type of application developed. In terms of the implementation of the communication network with the other micro-controllers in the electrical architecture, it was possible to implement the UART and CAN protocols, thereby ensuring precise and fast transmission times, particularly for the UART, for which it was possible to configure a baud rate of 8MBaud, thus meeting the requirement imposed in order to guarantee adequate DC-DC control. Furthermore, the ERU module enabled the implementation of a synchronisation mechanism, thereby enhancing the robustness of the UART protocol.

To analyse the behaviour of the microcontroller in External Clock Input mode, the System PLL was examined. It was found that the System PLL was capable of configuring the system clock to 300MHz, derived from an external clock of 10MHz. A further module that requires in-depth study is the Peripheral PLL, which is responsible for generating the clock for the interface sub-modules with the external environment. These include the MCMCAN and ASCLIN modules, which are used for the CAN and UART protocols, respectively. A potential future area of investigation could be the comprehensive analysis of the collective behaviour of the three micro-controllers, with their respective PERPLLs configured in an appropriate manner, with the objective of validating the configuration for the final architectural design, which involves the three AURIX micro-controllers sharing the

same input clock and communicating with each other.
The utilisation of the TOM module for the generation of the eight complementary PWMs with dead time proved to be an optimal solution for the control of the power structure. This was achieved by the use of nine TOM channels, which were all part of the same cluster, thus guaranteeing the propagation of the reset signal common to all counters. The implementation of the dead time was facilitated by the DTM within the GTM, as it was possible to implement the dead time via hardware, despite the three instants of clock delay caused by the DTM between input and output. This did not prove to be a problem, as the delay represents a constant value, thus guaranteeing deterministic behaviour.
From the experimental results, the AURIX TC39 micro controller was able to execute the periodic control task with a worst-case execution time of 5.09 micro seconds, thus falling within the imposed dead line of 8 µs, within which the reading from the sampling registers, the functions for calculating duty cycles and updating the shadow registers for the compare registers are executed.
To further optimise the robustness of the control, guaranteeing higher margins than the current dead line, a multi-core architecture analysis could be carried out in order to distribute the execution of the tasks in the application over the 6 cores of the AURIX TC39 architecture.
To conclude, it would be interesting to validate the software by performing an integration test, thus merging all the software portions developed and verifying the behaviour of the software and of the micro-controller in a simulated environment.

Bibliography

[1] Meera Khalid, Anjaly Mohan, and Binojkumar A. C. «Performance Analysis of Vector controlled PMSM Drive modulated with Sinusoidal PWM and Space Vector PWM». In: *2020 IEEE International Power and Renewable Energy Conference*. 2020, pp. 1–6. DOI: 10.1109/IPRECON49514.2020.9315210 (cit. on p. 8).

[2] Lifan Xiao, Jian Li, Junhua Chen, Ronghai Qu, and Yongqian Xiong. «Synchronous SVPWM for field-oriented control of PMSM using phase-lock loop». In: *2017 IEEE Energy Conversion Congress and Exposition (ECCE)*. 2017, pp. 4324–4331. DOI: 10.1109/ECCE.2017.8096745 (cit. on p. 8).

[3] Hasan Komurcugil and Sertac Bayhan. «Chapter 2 - Neutral-point-clamped and T-type multilevel inverters». In: *Multilevel Inverters*. Ed. by Ersan Kabalcı. Academic Press, 2021, pp. 29–56. ISBN: 978-0-12-821668-2. DOI: https://doi.org/10.1016/B978-0-12-821668-2.00003-9 (cit. on p. 9).

[4] Chung-Wei Tseng, Jain-Hong Liu, Ching-Tsai Pan, and Chia-Chi Chu. «Dual PWM Control for High Step-Up DC-DC Converters». In: *2020 IEEE Industry Applications Society Annual Meeting*. 2020, pp. 1–7. DOI: 10.1109/IAS44978.2020.9334730 (cit. on p. 10).

[5] Joon-Hee Lee and Seung-Ki Sul. «Compensation of Nonlinearity of Inverter through Estimation of Dead Time Effect». In: *2020 IEEE 9th International Power Electronics and Motion Control Conference (IPEMC2020-ECCE Asia)*. 2020, pp. 572–577. DOI: 10.1109/IPEMC-ECCEAsia48364.2020.9367949 (cit. on p. 12).

[6] Roberta Casablanca. «Study and development of an Electronic Power Converter for Electrical Vehicle based on General Time Multiplexing (GTM).[Master's thesis]». MA thesis. Politecnico Di Torino, 2023 (cit. on p. 13).

[7] Infineon. *AURIX SoC*. URL: https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc3xx/aurix-family-tc39xxx/?_gl=1*1plvuql*_up*MQ..&gclid=CjwKCAjwgpCzBhBhEiwAOSQWQWQlP1sGIaI1x2vKUZvcHQEZJ9T0a7VaVeC65JHmXhCvoQhIENwoiRoCKfEQAvD_BwE&gclsrc=aw.ds#interactivegraphic-64474314-0a94-11e9-bd35-40f2e90e8d04/overview/k/k (cit. on p. 14).

[8] Infineon. $KIT_A2G_TC399_5V_TRB$. URL: https://www.infineon.com/cms/en/product/evaluation-boards/kit_a2g_tc399_5v_trb/ (cit. on p. 16).

[9] Yi-yuan Fang and Xue-jun Chen. «Design and Simulation of UART Serial Communication Module Based on VHDL». In: *2011 3rd International Workshop on Intelligent Systems and Applications*. 2011, pp. 1–4. DOI: 10.1109/ISA.2011.5873448 (cit. on p. 17).

[10] Ashok Kumar Gupta, Ashish Raman, Naveen Kumar, and Ravi Ranjan. «Design and Implementation of High-Speed Universal Asynchronous Receiver and Transmitter (UART)». In: *2020 7th International Conference on Signal Processing and Integrated Networks (SPIN)*. 2020, pp. 295–300. DOI: 10.1109/SPIN48934.2020.9070856 (cit. on p. 19).

[11] Infineon. *AURIX™ TC3xx User Manual Part-2*. Version 2.0 (cit. on pp. 20–24, 29, 31, 32, 42, 66, 70, 72, 73).

[12] Tiphoon-HIL. *CAN Bus protocol*. URL: https://www.typhoon-hil.com/documentation/typhoon-hil-software-manual/References/can_bus_protocol.html#:~:text=The%20Controller%20Area%20Network%20protocol,and%20interact%20in%20a%20network (cit. on p. 25).

[13] Infineon. *AURIX Training Generic Timer Module*. URL: https://www.infineon.com/dgdl/Infineon-AURIX_Generic_Timer_Module-Training-v01_00-EN.pdf?fileId=5546d46269bda8df0169ca6e2c652546 (cit. on pp. 28, 31).

[14] Infineon. *AURIX Training CAN Interface*. URL: https://www.infineon.com/dgdl/Infineon-AURIX_TC3xx_CAN_Interface-Training-v01_00-EN.pdf?fileId=5546d46274cf54d50174da4ee8cb226c (cit. on p. 43).

[15] Giuseppe Di Carolo. «Study and Development of Advanced Core Communication and Memory Management Techniques for Real-Time Battery Electric Vehicle (BEV) Applications.[Master's thesis]». MA thesis. Politecnico Di Torino, 2023 (cit. on p. 53).

[16] Infineon. *AURIX™ TC3xx User Manual Part-1*. Version 2.0 (cit. on pp. 56–59, 61, 62).

[17] Infineon. *AURIX™ TC39x Data Sheet*. Version 1.2 (cit. on pp. 57, 60).

[18] Infineon. *AURIX™ TC39x User Manual*. Version 2.0 (cit. on p. 78).

[19] Sarah Azimi Luca Sterpone Boyang Du. «Radiation-induced single event transient modeling and testing on nanometric flash-based technologies». In: *Microelectronics Reliability* 55 (2015).

[20] Luca Sterpone Sarah Azimi Corrado De Sio Daniele Rizzieri. «Analysis of single event effects on embedded processor». In: *Electronics* 10 (2021).

[21] Andrea Portaluri Luca Sterpone Sarah Azimi Corrado De Sio Daniele Rizzieri. «A comparative radiation analysis of reconfigurable memory technologies: FinFET versus bulk CMOS». In: *Microelectronics Reliability* 138 (2022).