Politecnico di Torino

College of Computer Engineering, Cinema and Mechatronics

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

Study and development of an Electronic Power Converter for Electrical Vehicle based on General Time Multiplexing (GTM)

Supervisor:
Prof. Luca STERPONE

Candidate:
Roberta CASABLANCA

Co-supervisor:
Ph.D. Sarah AZIMI

Academic year 2022-2023

Abstract

In the last few years, technological goals for vehicles span across various fields, including researching driving comfort, reducing emissions, reusing energy, improving functional safety, and automating car movements. Traction control is a crucial area of study in hybrid cars, where the role of electronics plays an important part in connecting and controlling the mechanical components of the vehicle. This allows for autonomous control and the use of algorithms that enhance the car's performance, as is the case with all engineering applications. Focusing on a hybrid car, multiple parts can be controlled and three main interfaces are present: fuel cell, motor and battery, which must interface.

This thesis concerns on a specific aspect of traction control, specifically how signals are managed and transferred between interfaces to regulate acceleration and deceleration. When signals pass between different interfaces like the engine and the battery, they need to be remodeled. This can be achieved using structures such as DC/AC inverters or DC/DC converters. In hybrid cars, electronic controls can be used to manage the DC/AC or DC/DC interfaces with the engine, battery, or fuel cell. The electronics are integrated into the car via SoC (system-on-chip) and, the one used in this case, is an Infineon Tricore Board (AURIX™ TriBoard TC3x9), equipped with GTM from Bosch, a generic timer platform useful in different application domains, in particular in the automotive field and control of motor signals.

The presented work focuses on the DC/DC converter and, in detail, on the handling of DC/DC switch movements needed to manage the direction (i.e. acceleration or deceleration) and amount of current and voltage, through the use of PWM signal. The study explored different approaches for the creation of complementary PWM signals, going deeper in problems such as the short circuit between switches and solving them with the insertion of dead time in control signals. For the case of multiple cells of

switches and phase-shift control, also the shifting between signals it has been realized, with different methods.
These multiple methods were created through the development of C-code programs using the "Aurix Development Studio" platform. Simulations results are shown and used methods are compared, in terms of speed, complexity and synchronization.

Contents

List of Figures

Chapter 1

Introduction of Traction Control Handling

Nowadays, electronics play a crucial role in cars as they provide additional value in terms of comfort, safety, precision, and extra features.
Many driver assistance systems, like cruise control, automatic parking, advanced front lighting, and automatic windshield wipers, have been developed to improve driver comfort. Electronic controls, such as engine and battery management, have also been implemented to reduce fuel consumption and increase efficiency. Lowering exhaust emissions is a goal to protect the environment, and automatic control of fuel or air concentration can help achieve this. Recently, engineers have focused on improving vehicle safety measures, such as traction control, anti-lock braking systems, hill descent control, obstacle and collision avoidance, lane keeping, lane change assistance, and driver drowsiness detection.
In recent years, the growing focus on clean energy and electronic controls has sparked a specific interest in electric and hybrid cars. These vehicles can have their traction controlled through electronics, allowing for the regulation of acceleration and deceleration, as well as the generation of energy through braking (regenerative power).
A generic hybrid car has multiple signals that flow through its main three interfaces: engine, battery and fuel cell, as shown in Figure Fig. 1.1.

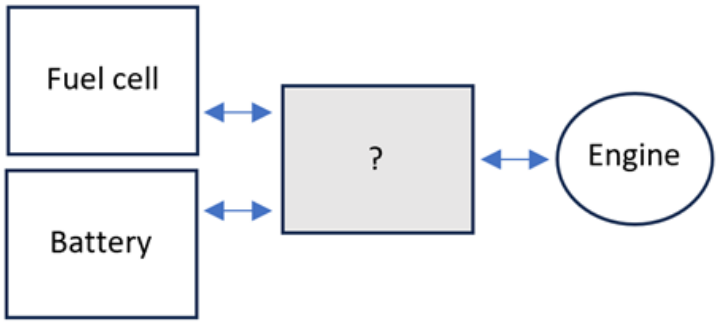

Figure 1.1: Hybrid vehicle interfaces block diagram

Fuel cells convert the chemical energy of fuel and an oxidizing agent into electricity through redox reactions, while the battery can be sometimes used to supply the electric motor: their usage mode depends on the hybrid structure parallel or series. These parts of the vehicle are in communication because, for example, if the battery is discharged the engine is used, or, if there is a regenerative braking system, the braking energy of the car wheels can charge the battery.
Multiple electronic controls so are needed to manage signals between these interfaces and one of the main aspects is the problem of modulating signals because, for example, the battery uses DC power and the motor needs AC power.
For this problem, the DC/AC inverter and DC/DC converter are introduced (they are developed in the "?" block of Figure 1.1) and so the power is able to flow (in the right shape and order of size) when the vehicle accelerates or decelerates, when the battery discharges or charges by regenerative braking ext.
The electronics used to control signals are integrated into the car via SoC (system-on-chip) which is an integrated circuit that in a single chip contains an entire system and they are usually made of silicon.
One example of a SoC is the microcontrollers offered by STM or Infineon, as Fig. 1.2.

Figure 1.2: Infineon SoC example [1]

Microcontrollers are equipped with all the needed modules for signal acquisition, signal generation, memory allocation, calculations, clock generation, interrupt generation ext. Going forward with the years, microcontrollers have become more specialized in automotive applications, with the introduction of specific submodules that solve tasks and problems that occur in vehicle control.
In particular, the AURIX TriCore microcontroller of Infineon TC3xx is useful for a wide range of automotive and industrial applications, such as the control of combustion engines, electrical and hybrid vehicles, transmission control units and braking systems.
This thesis, with the use of this microcontroller, focuses on the exploration of DC/DC converter control signals. It is made of multiple switches (FET technology), that can be opened or closed with a digital signal, in order to manage the flow and direction of current.
The digital signal used to control DC/DC switches is called PWM (Pulse Width Modulation) and its generation techniques are explored, comparing them and dealing with the problems that can be encountered in this type of application.
The goal is to regulate current and power flow direction to enable effective traction

control for the vehicle by utilizing state-of-the-art FET technology and the Infineon Tricore microcontroller in the best possible way.

Chapter 2

State of art

2.1 SoC used for traction control

Vehicle controls are possible thanks to the integration of electronics into the car via SoC (system-on-chip) which is a single chip that contains an entire system in it, so, in addition to the on-chip central processing unit (CPU), there are memory interfaces, input/output interfaces, and secondary storage interfaces, often alongside other components such as radio modems and a graphics processing unit (GPU). System-on-Chips (SoCs) are typically constructed from silicon, using various technologies that are selected based on technical needs[2].
Examples of microcontrollers are those of STMicroelectronics or Infineon, and, in recent years, they have become specific for automotive or industrial applications, such as control of hybrid vehicles.
STMicroelectronics provides a diverse range of microcontrollers specifically tailored for use in automobiles. This includes the 32-bit SPC5 family that utilizes Power Architecture technology, the 8-bit STM8A family, and the 16-bit ST10 legacy MCUs. In addition, STMicroelectronics has recently unveiled its newest line of high-performance 32-bit microcontrollers, known as Stellar. These microcontrollers are built on the ARM R52 multi-core platform and are designed specifically for use in the automotive industry.[1]

Infineon provides the AURIX tricore microcontroller, that is a board with some modules specialized for automotive purposes and, in particular, the TriBoard TC3X9

Th evaluation board has been used for this thesis work.
TriCore is a unified, 32-bit Microcontroller DSP optimized for real-time embedded systems and it is shown in Fig. 2.1. “Tricore” name is Infineon trademark for 32 bit MCU.
It is a high-performance microcontroller with multiple (up to six) TriCore CPUs, program and data memories, buses, bus arbitration, interrupt system, DMA controller and a powerful set of on-chip peripherals such as serial controllers timer units, and analog-to-digital converters.[3]
The board contains a Bosch GTM IP, a timer module with specialized submodules for acquiring and generating automotive signals and many other applications. A schematic representation of the board is shown in Fig. 2.2.

Figure 2.1: 32 bit AURIX Tricore Microcontroller [4]

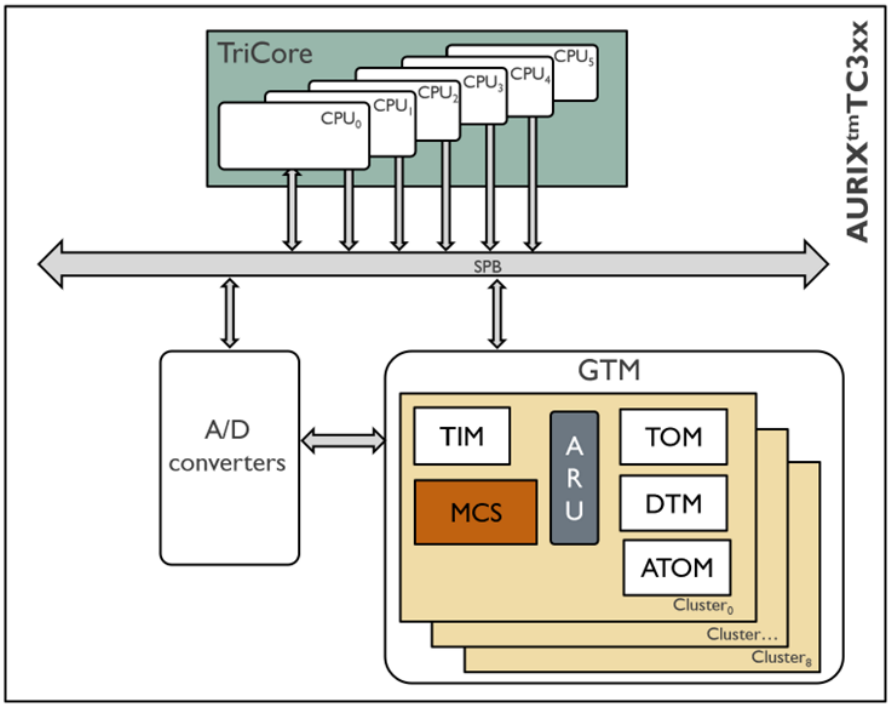

Figure 2.2: AURIX TC3xx schematic block

2.2 PWM definition

PWM (Pulse Width Modulation) is a technique for controlling the average power delivered by an electric signal. The average voltage and current supplied to the load are adjusted by switching the supply between 0% and 100%.
The longer the switch is on, the higher the total power delivered to the load. Because of their on/off nature, PWM works very well on digital controls, and it is very useful for those applications in which power must be handled.
In digital control, switches are essential to recreate the 0/1 signal. These switches are typically made using field-effect transistors (FETs), which can be in various types as depicted in Fig. 2.3.

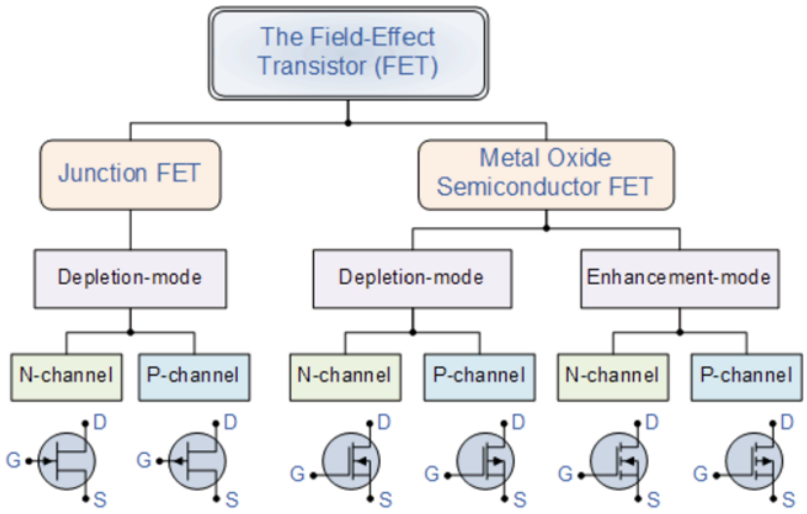

Figure 2.3: Types of FET

The mathematical concept of duty cycle plays a crucial role in PWM. It describes the amount of 'on' time during a period interval. A low duty cycle indicates low power as the power is mostly off. The duty cycle is expressed in percentage, 100% being fully on; an example of different duty cycle values is shown in Fig. 2.4.

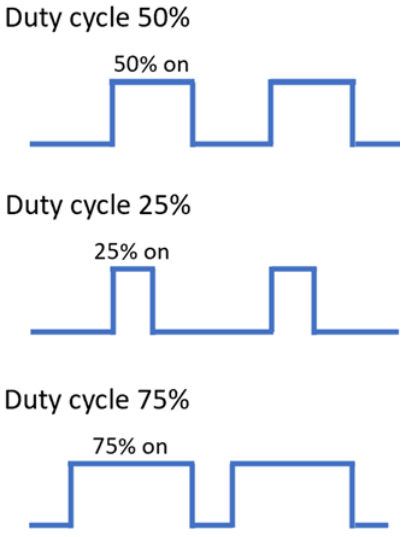

Figure 2.4: Examples of PWM with different duty cycle values

2.3 PWM through history

The development of PWM switching techniques can be traced back historically and linked to advancements in technology. This progress began with analogue-based systems, then moved on to discrete digital, ROM-based, and eventually microprocessor-implemented control schemes. Most analogue-based PWM control schemes rely on "natural" sampled switching strategies or "regular" sampling, which can be implemented using microprocessor technology and are expected to serve as the foundation for most microprocessor PWM control [5].

One of the first PWM application was that of Corliss steam engine, that was patented in 1849 (Fig. 2.5). It used pulse-width modulation to control the intake valve of a steam engine cylinder.

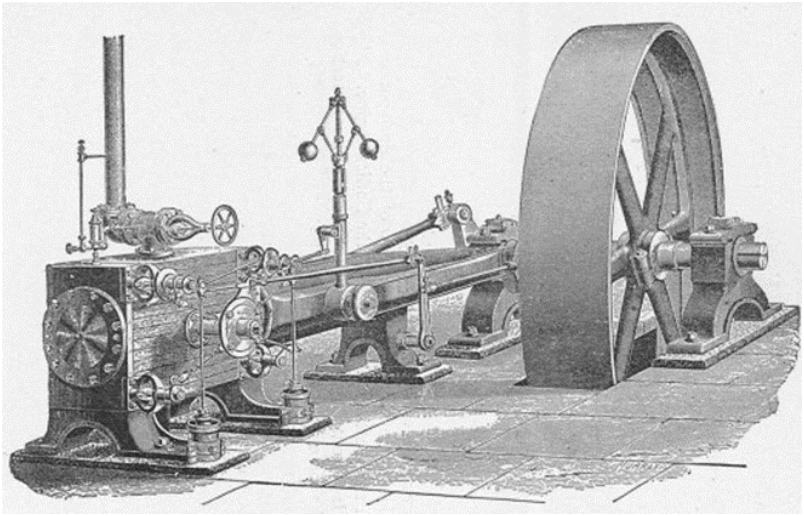

Figure 2.5: Corliss steam engine: the valve gear is on the right of the cylinder block, on the left of the picture

2.4 PWM concept in relation to electric vehicles

Analyzing the concept of PWM, it is possible to make an example of how a motor of an electric vehicle can be managed by a PWM signal.
Taking into account a digital signal, it can have only two values: "0", corresponding

to 0 V and engine off and vehicle off, and "1", related to, for instance, 5V and engine on at the maximum of its speed, for example, 100 km/h.
Using only a digital signal, the vehicle can only be still or go at full speed: this is also not possible for physics reasons, because it implies an instantaneous acceleration or deceleration exists. This means that a basic digital signal is insufficient to control a motor in a real-world vehicle. The use of PWM so becomes the solution to this motor control problem.
The digital signal can be made periodic, for example with the frequency of 50 Hz so a period of 20 ms, and with 50% of duty cycle:

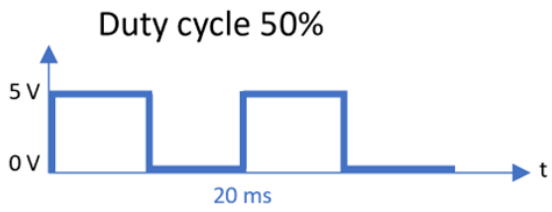

Figure 2.6: Duty cycle of 50%

During half of the period, the motor receives maximum power while it receives zero power during the other half. This means that it is equivalent to receiving 2.5 V for the entire period, which is half of the maximum power because of the 50% duty cycle.

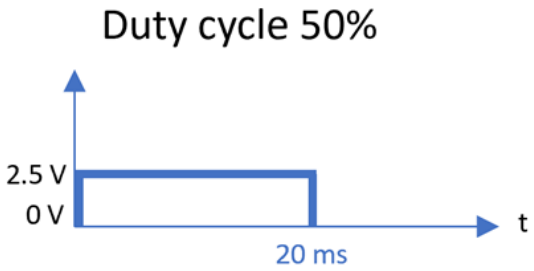

Figure 2.7: Equivalence to 2.5 V for the entire period

Imposing a duty cycle of 25%, the motor receives for the entire period 1.25 V (25%

of 5 V).

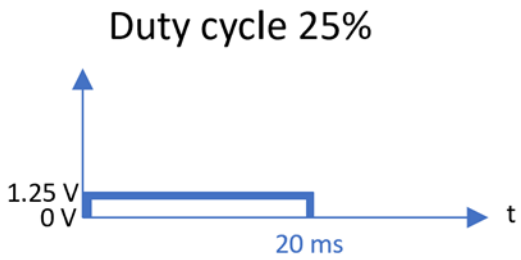

Figure 2.8: Equivalence to 1.25 for the entire period

So different motor velocities are related to different duty cycles; so for a gradual braking or acceleration a sequence of different duty cycles is needed, as shown in Fig. 2.9 and Fig. 2.10.

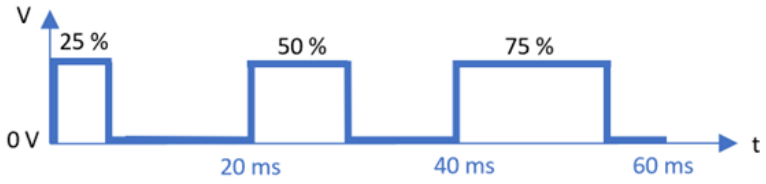

Figure 2.9: Signal with variable duty cycle values

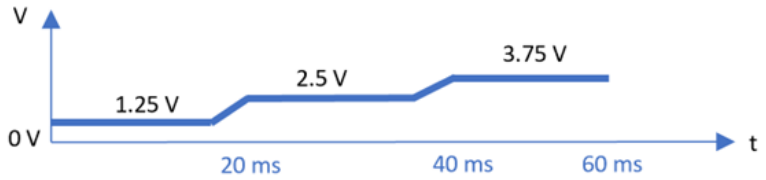

Figure 2.10: Gradual acceleration of vehicle due to variable duty cycle

2.5 PWM realization

Going into the details of creating the PWM, one common method for determining switching instants is to compare a triangular carrier wave sampling signal directly with a sinusoidal modulating wave. This technique was widely used in the past due to its simplicity and easy implementation with analogue techniques.

2.5.1 General PWM modulation

For the most common PWM modulation, as shown in Fig. 2.11, the carrier $V_p(t)$ is a triangular or sawtooth wave of fixed frequency f while the modulating wave Vi modifies the duration of the high state with respect to the entire period (the duty cycle value), where the period T is: $T = 1/f$. In this way, the average value of the modulated carrier is proportional to the amplitude of the modulating voltage [6].

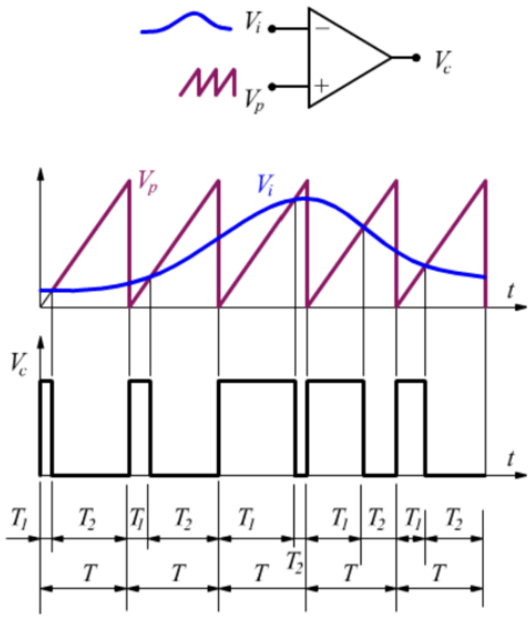

Figure 2.11: Comparator and signals scheme: vp is the carrier, vi the modulating wave

Sinusoidal waves can be saved in look-up tables or RAM memories. For micropro-

cessor application, the triangular carrier wave can be generated through software's up/down counter. The speed at which the counter is incremented (or decremented) determines the carrier frequency and accuracy of the sampling process.

2.5.2 Asymmetric PWM

Another example of modulation is the asymmetric PWM, which uses a sawtooth/ramp waveform as carrier and a constant value $v_c(t)$ as modulating signal (Fig. 2.12and Fig. 2.13).

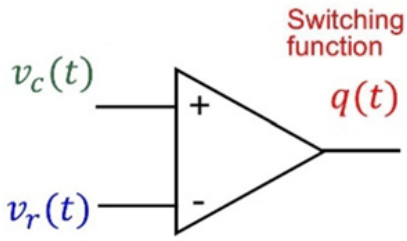

Figure 2.12: Comparator between carrier and modulating signal

The value of $v_c(t)$ is included in the domain: $0 \leq v_c(t) \leq \hat{V}_r$.

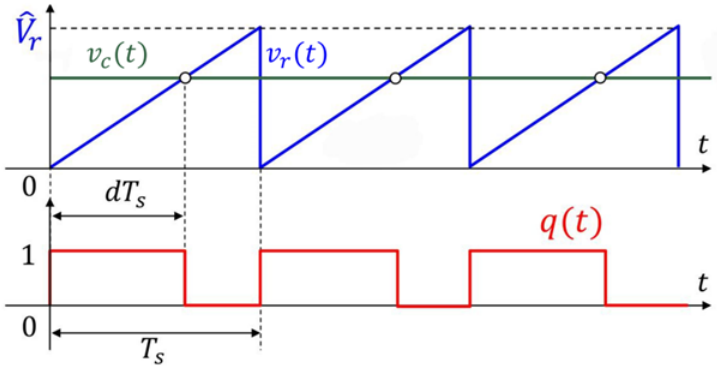

Figure 2.13: Waveforms for asymmetric PWM

The value of $q(t)$ is generated considering the rules for comparison of signals, so:

- $q(t) = 1$ if $v_c(t) > v_r(t)$;
- $q(t) = 0$ if $v_c(t) < v_r(t)$.

The value of $d(t)$, the duty cycle of the PWM waveform, is so related to the carrier and modulating waves as:

$$d(t) = \frac{1}{\hat{V}_r} * v_c(t)$$

2.5.3 Symmetrical PWM with unipolar triangular carrier

Another example of modulation is the symmetrical PWM, which uses a unipolar triangular waveform as carrier and a constant value v_c (t) as modulating signal (Fig. 2.14).

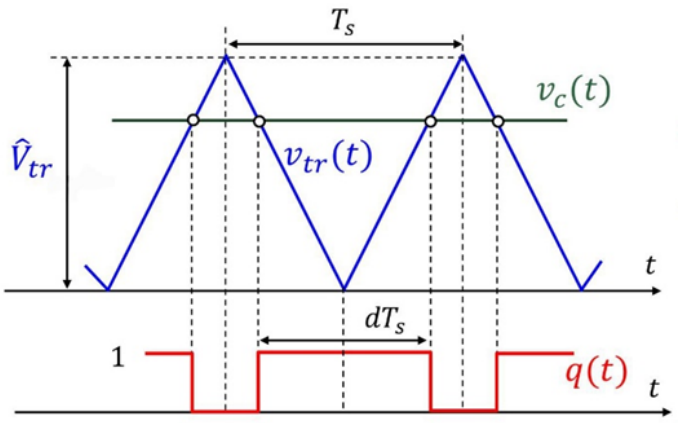

Figure 2.14: Waveforms for symmetrical unipolar PWM

The value of $v_c(t)$ is included in the domain: $0 \leq v_c(t) \leq \hat{V}_{tr}$.

The value of $q(t)$ is generated following the rules for comparison of signals, so:

- $q(t) = 1$ if $v_c(t) > v_{tr}(t)$;
- $q(t) = 0$ if $v_c(t) < v_{tr}(t)$.

The value of $d(t)$, the duty cycle of the PWM waveform, is so related to the carrier and modulating waves as:

$$d(t) = \frac{1}{\hat{V}_{tr}} * v_c(t)$$

2.5.4 Symmetrical PWM with bipolar triangular carrier

Another example of modulation is the symmetrical PWM, which uses a bipolar triangular waveform as carrier and a constant value $v_c(t)$ as modulating signal (Fig. 2.15).

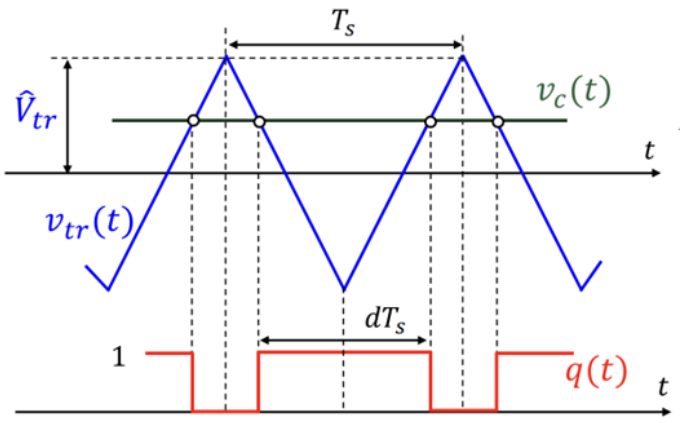

Figure 2.15: Waveforms for symmetrical bipolar PWM

The value of $v_c(t)$ is included in the domain: $-\hat{V}_{tr} \leq v_c(t) \leq \hat{V}_{tr}$.

The value of $q(t)$ is generated following the rules for comparison of signals, so:

- $q(t) = 1$ if $v_c(t) > v_{tr}(t)$;
- $q(t) = 0$ if $v_c(t) < v_{tr}(t)$.

The value of $d(t)$, the duty cycle of the PWM waveform, is so related to the carrier and modulating waves as:

$$d(t) = \frac{1}{2} + \frac{1}{\hat{V}_{tr}} * v_c(t)$$

2.6 Digital PWM implementation in microcontrollers

For power electronics control, the digital PWM modulator is implemented in microcontrollers and its main feature is the comparison between the carrier wave $n_{tr}(t)$, realized through an up/down counter, and a modulating signal that is an integer number $n_c(t)\epsilon[0, N]$(Fig. 2.16).

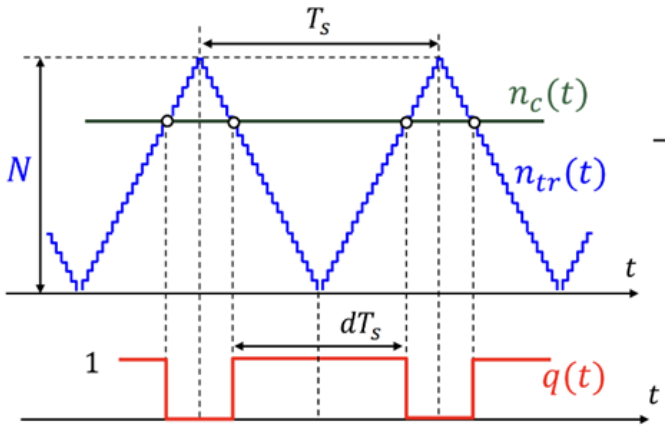

Figure 2.16: Digital PWM realization

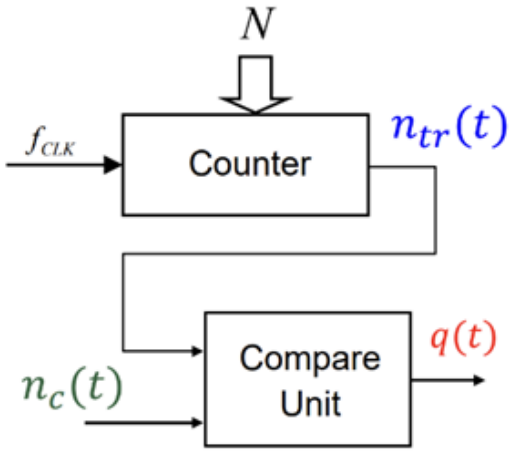

Figure 2.17: Digital implementation for PWM realization

When the counter is increased or decreased, it is compared to a sample of the

modulating wave to determine when to switch the PWM, thanks to the compare unit, as shown in figure 2.17. To create the width-modulated pulse, the microprocessor must load a value into the timer counter that corresponds to the desired pulse width. The timer's clock controls the decrement rate of the counter. Once the counter reaches a value of zero, it generates an interrupt signal to inform the microprocessor that the time period has ended[7].
The number $n_c(t)$ depends by the desired duty cycle:

$$n_c(t) = d(t) * N$$

The value N to count is imposed by the desired switching frequency and the counter clock frequency f_{clk}:

$$N = (T_s * f_{clk})/2 = f_{clk}/(2 * f_s)$$

Nowadays microprocessors have become oriented and highly specialized in automotive applications, so they have specialized modules for PWM realization for motor control to make PWM management easier and more precise.
STM and Infineon are examples of more sophisticated microprocessors that make easier the goal of traction control.

2.7 Dead time insertion in PWM waveforms

PWM can be used to manage the switches of a bidirectional switching cell, such as a DC/DC converter in automotive applications. For a real bidirectional switching cell, a dead time must be inserted between the individual switching functions, to avoid the short circuit on the input voltage source. (Fig. 2.18)

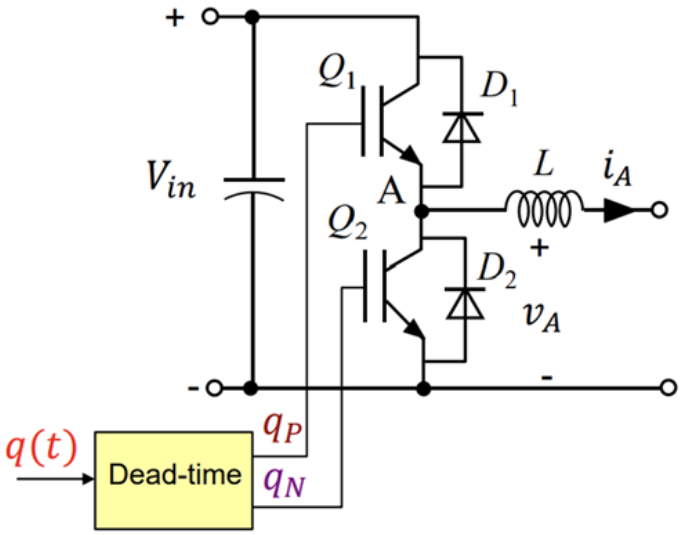

Figure 2.18: Bidirectional switching cell with dead time insertion

The situation of short circuit, called "shoot through", is highly risky as it has the potential to cause overheating and harm to both the transistors and the entire circuit. When designing FET-based circuits, it is important to take into account shoot-through caused by device delays and non-idealities like gate capacitance and diode reverse recovery effects. This is because a MOSFET cannot function as a perfect switch and there is a slight delay between the gate control signal being turned on/off and the MOSFET itself turning on/off.

A common way to address the issue of shoot-through resulting from the non-idealities of FETs is so by incorporating dead time into the PWM control.

Dead time refers to a brief period of time that is introduced between the switching edges (Fig. 2.19) of the PWM signals that control switches on the same leg, particularly in the context of motor control.

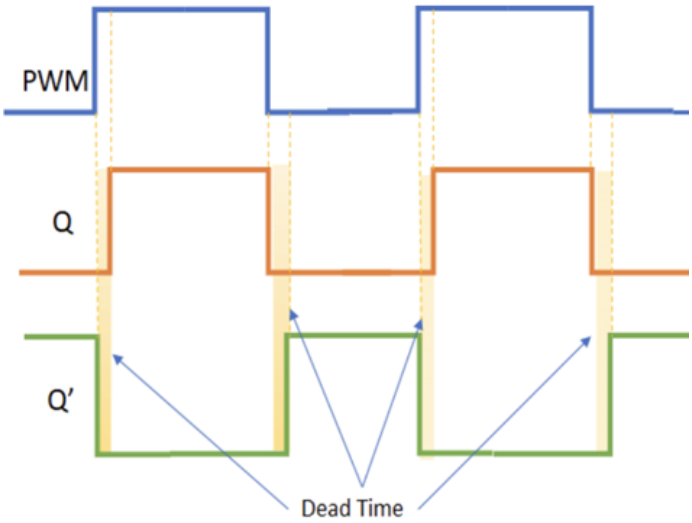

Figure 2.19: PWM waveforms with dead time

By leaving a buffer of time between when one FET turns off and when the other turns on, dead time prevents shoot-through by ensuring that no two transistors on the same leg will be on simultaneously.

The insertion of dead time creates however some problems in the circuit, indeed, the dead-time effect, at high switching frequencies can cause significant voltage and current losses and serious instability and low-frequency output harmonics.
Additionally, the voltage error and distortion are directly proportional to the PWM switching frequency due to the increasing dead time to sampling time ratio. The dead-time elimination or compensation can be a solution to the dead-time issue at high switching frequencies and there are open studies about this issue[7] [8].
Dead time insertion is always requested in automotive context, so the microcontrollers dedicated to motor control have dedicated dead time units that must be programmed to add the dead time to the generated PWM.

Chapter 3

Background

3.1 Generic Timer Module (GTM)

The Generic Timer Module (GTM) is a Bosch product with an architecture designed to offer a flexible platform for embedded control applications in various domains [9]. It is located inside the TC3xx architecture, that is the microcontroller of Infineon used [10]. It is realized using several sub-modules organized into channel clusters (Figure 3.1).

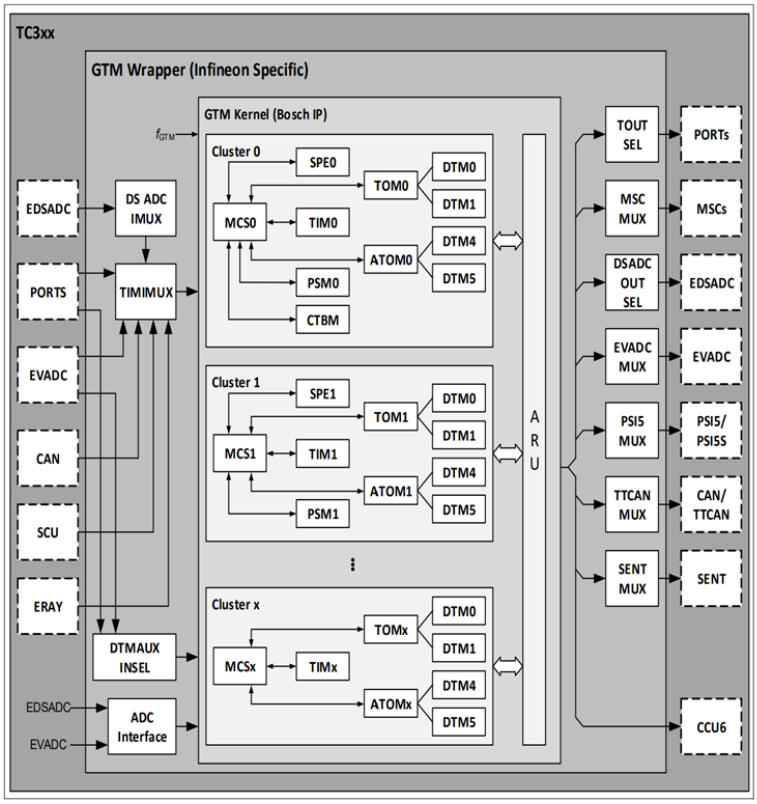

Figure 3.1: GTM architecture block diagram [11]

The main infrastructural submodules are:

- CMU (Clock Management Unit): offers clock prescalers for GTM modules and counters;
- TBU (Time Base Unit): common time bases provider;
- ICM (Interrupt Concentrator Module): interrupt concentrator of GTM;
- PSM (Parameter Storage Module);
- BRC (Broadcast Module);
- ARU (Advanced Routing Unit): for transferring data between GTM submodules;

Input/Output submodules:

- TIM (Timer Input Module): module for incoming signals;

- TOM (Timer Output Module): module for signal generation;
- ATOM (ARU-connected Timer Output Module): additional connection with ARU;
- DTM (Dead Time Module): useful to insert dead time between PWM and its inverse for half bridges controlling;

Application-specific submodules:

- DPLL: for engine positioning detection and angle clock;
- MAP: maps the signal sampled at TIM to DPLL;
- SPE: for input pattern detection and output generation;

Submodules for functional safety:

- CMP(Output Compare Unit): compares signals checking for errors;
- MON(Monitor Unit): to signal errors to CPU;

GTM also provides a programmable core: the MCS (Multi channel sequencer)[9]. For generating PWM, there are various available submodules, including the TOM (Timer Output Module) and ATOM (Aru-connected Timer Output Module) modules which have been explored.

3.1.1 Timer Output Module (TOM)

The Timer Output Module (TOM) is capable of producing PWM signals that can either be dependent or independent of one another.
Two TOM channels are shown in Figure 3.2.

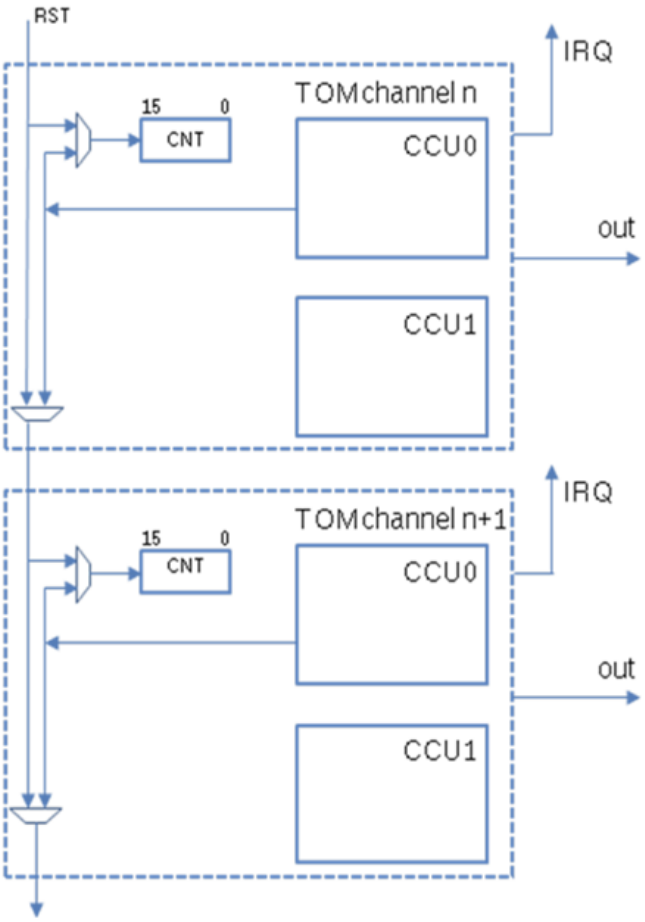

Figure 3.2: Architecture of TOM channels [9]

Each TOM channel has the following characteristics:

- The counter frequency can be chosen from one of five prescaler clocks provided by CMU;
- It has is own 16 bit counter, used to generate the edges of the PWM;
- It has two capture compare units (CCU0 and CCU1) which are used to compare the counter against a value that can be configured. CCU0 is responsible for determining the duration of the PWM period while CCU1 determines the duration of the duty cycle.

The TOM submodule, in the board used, is equipped with 6 clusters of 16 channels each and it is possible to communicate with many other submodules.

3.1.2 ARU-connected Timer Output Module (ATOM)

In contrast to the TOM, the ATOM internal registers have a 24 bit width. The channels' shadow registers or compare registers can be reloaded from a source linked to the ARU, depending on the channels' mode[9].
Also the ATOM submodule channels are comprised of two compare units with compare registers and associated shadow registers (Figure 3.3).

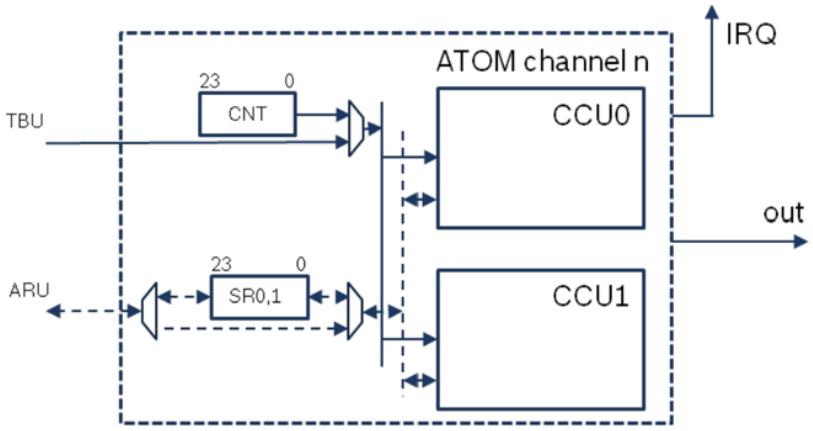

Figure 3.3: Architecture of ATOM channel [9]

The ATOM submodule, in the board used, is equipped with 12 clusters of 8 channels each, with a counter value: of 24 bit and 8 global programmable prescalers.

The operation modes of this submodules are 5:

- Signal Output Mode PWM (SOMP);
- Signal Output Mode Compare (SOMC);
- Signal Output Mode Immediate (SOMI);
- Signal Output Mode Serial (SOMS);
- Signal Output Mode Buffered Compared (SOMB).

ATOM, thanks to the ARU connection, can automatically reload new data values and control signals without the need of CPU interaction.

The ATOM/TOM channel is able to generate complex PWM signals with different duty cycles and periods which can be changed synchronously and asynchronously.

- Synchronous change means that the duty cycle or period duration changes after the end of the preceding period (used setting);
- An asynchronous change means that the duration changes during the actual running PWM period.

3.1.3 Dead Time Module (DTM)

This module is mainly used for the the dead time insertion. It is composed by (Figure 3.4):

- 6 clusters CDTM[i];
- In each CDTM[i]:
  - CDTM[i]_DTM 0...3 for TOM[i] channels;
  - CDTM[i]_DTM4-5 for ATOM[i] channels;
  - In each DTM[x] there are 4 inputs channels

The main function of the DTM is to derive for each input DTM_IN0 to DTM_IN3 the individual inverse signal (DTM[i]_OUT[x]_N) and to apply an edge specific delay between the edge of the original signal and the edge of the derived inverted signal (i.e., the dead time).

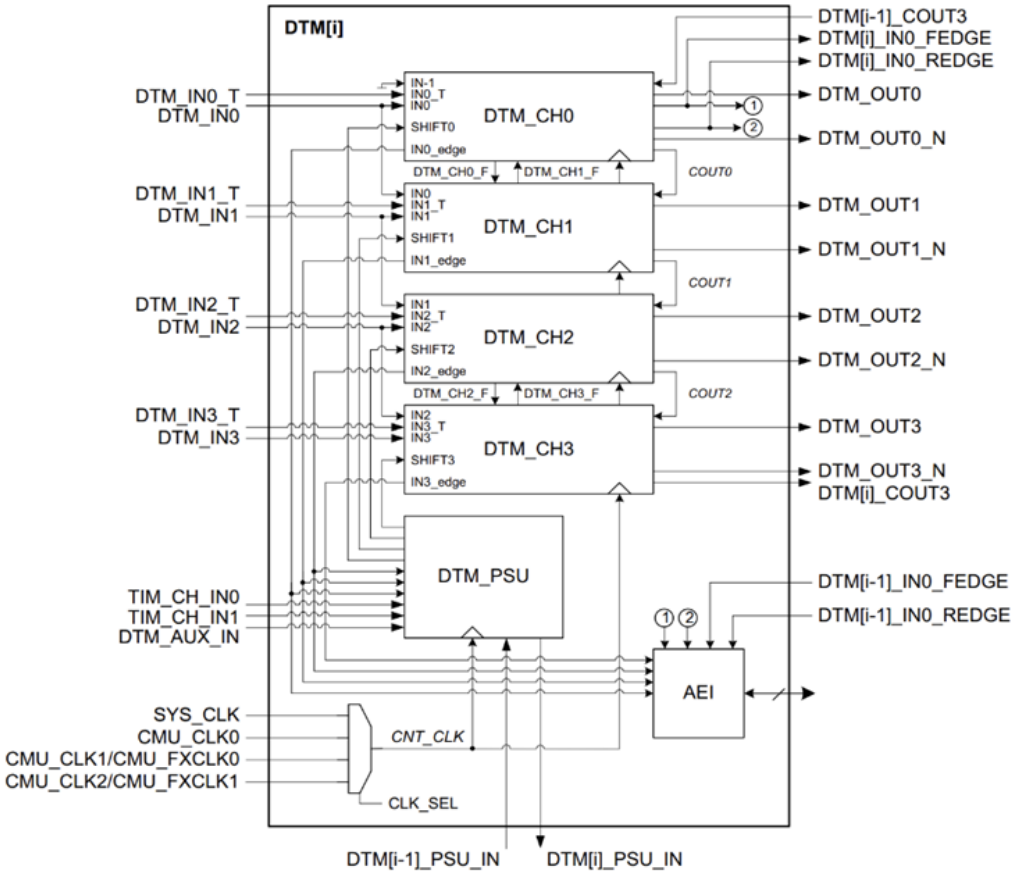

Figure 3.4: Architecture of DTM submodule [11]

The main problem of the use of DTM is the 3 clock periods delay between its input and outputs, so between the main triangular wave (DTM_IN[x]) and PWMs for switches (DTM_OUT[x] and DTM_OUT[x].N) (Figure 3.5).

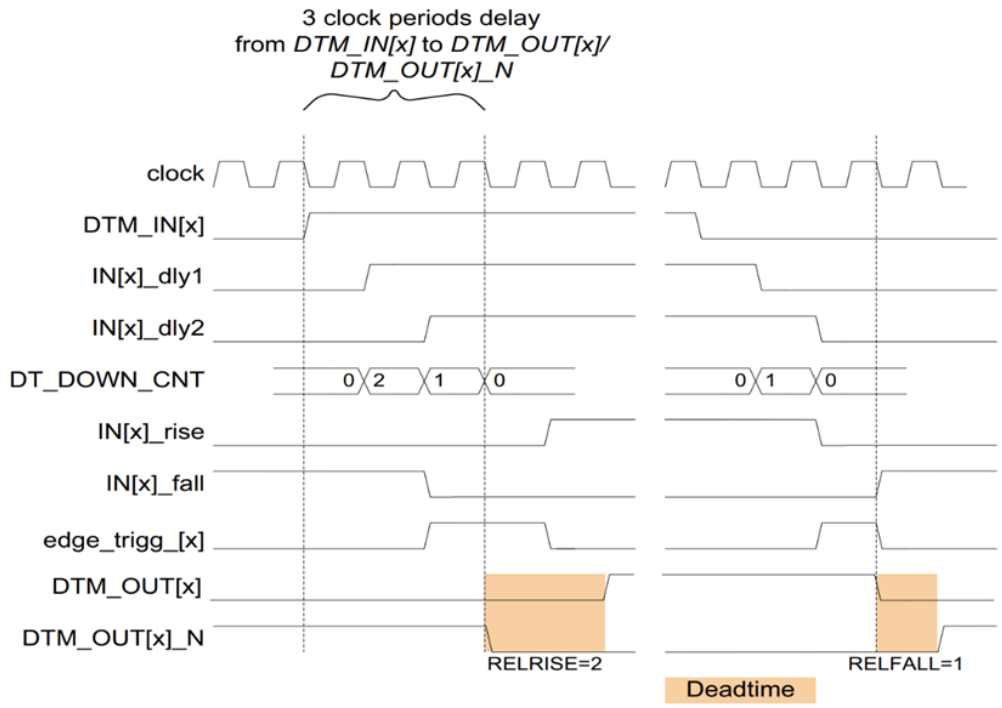

Figure 3.5: DTM signals overview [11]

3.1.4 Clock Management Unit (CMU)

This submodule (Figure 3.6) generates the clock prescalers for the counters and registers of GTM submodules.

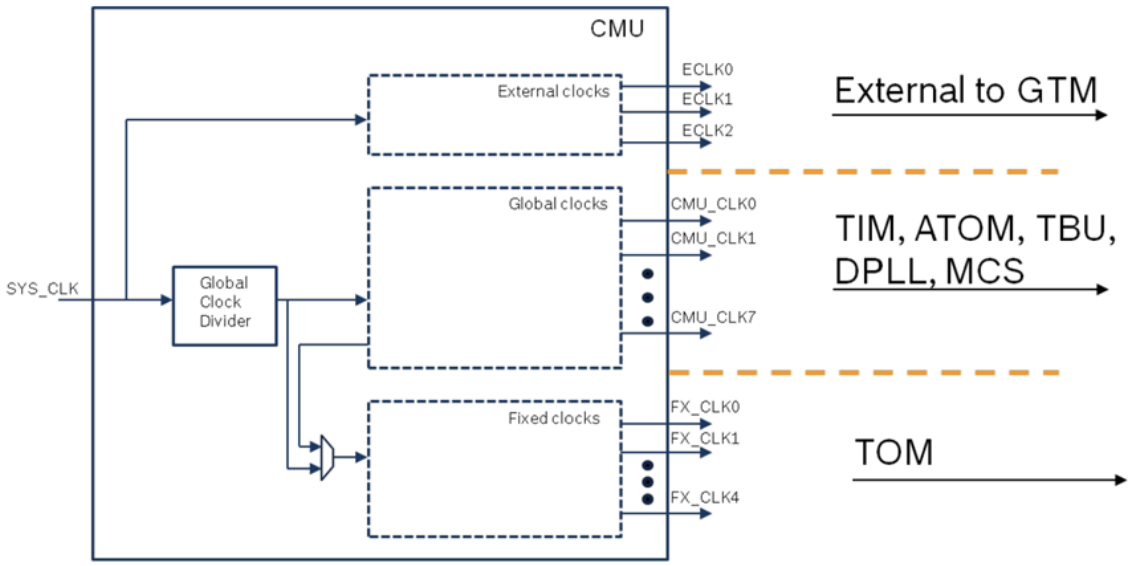

Figure 3.6: CMU architecture [9]

Different clocks are available: three clocks for external applications to GTM, eight global (programmable) clocks for applications in submodules such as TIM, ATOM, TBU, DPLL and MCS and five fixed clocks for TOM submodules.
To set the frequency of ATOM channels, the SYS_CLK signal and The Global Clock Divider must be set. The maximum value of frequency of GTM is 200 MHz and this value was used for the application of this thesis work.

Chapter 4

Proposed solution

4.1 Goals

When driving a motor, the key controls are its speed and direction. To accomplish this, it is common to use PWM to drive the MOSFET gates, so the switches for example of a DC/DC converter of an hybrid vehicle, as shown in Figure 4.1.

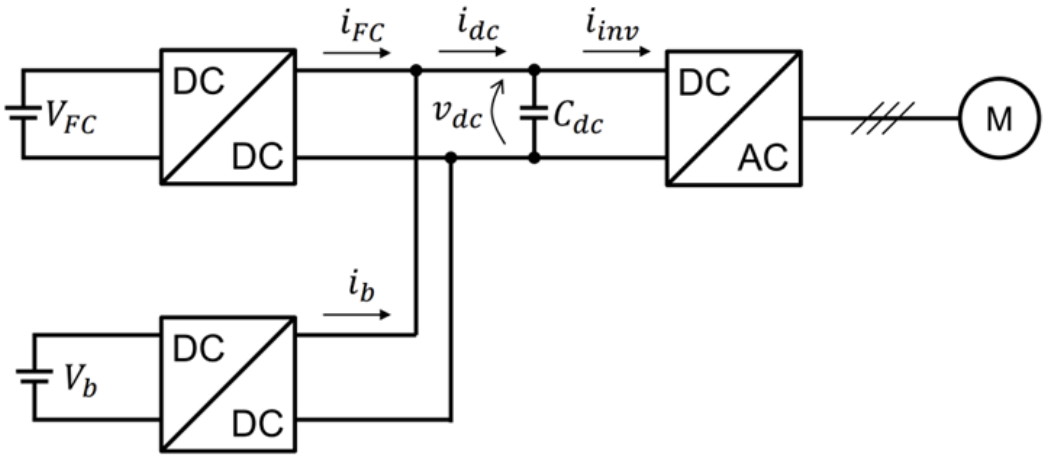

Figure 4.1: Electrical scheme of a hybrid vehicle

The vehicle being considered has three primary interfaces: the fuel cell, battery, and motor. The DC/AC inverter plays a crucial role in converting the quantities to alternate mode, allowing for control operation of the motor. Additionally, the DC/DC

interfaces are necessary for regulating the magnitude of DC quantities from/to the battery and fuel cell of the vehicle [12]. PWM allows for the motor's speed to be controlled by adjusting its duty cycle (the amount of time it is on), allowing for precise power supply to the motor.
The internal structure of a single DC/DC converter is displayed in Figure 4.1.
In order to regulate the switches, PWM signals labeled as Qp1, Qn1...Qp4, Qn4 are required.
This control scheme, where multiple gates are driven by PWM signals 180° out of phase with one another, is known as complementary PWM. When using complementary PWM, it's important to be aware of the risk of a short circuit, also called "shoot-through", which can occur in the same leg when both switches, for example Qp1 and Qn1, are turned on simultaneously. This happens because Qp1 and Qn1 are two switches in the same leg that work together. Typically, to resolve this issue, dead time is added to the edges of PWMs.

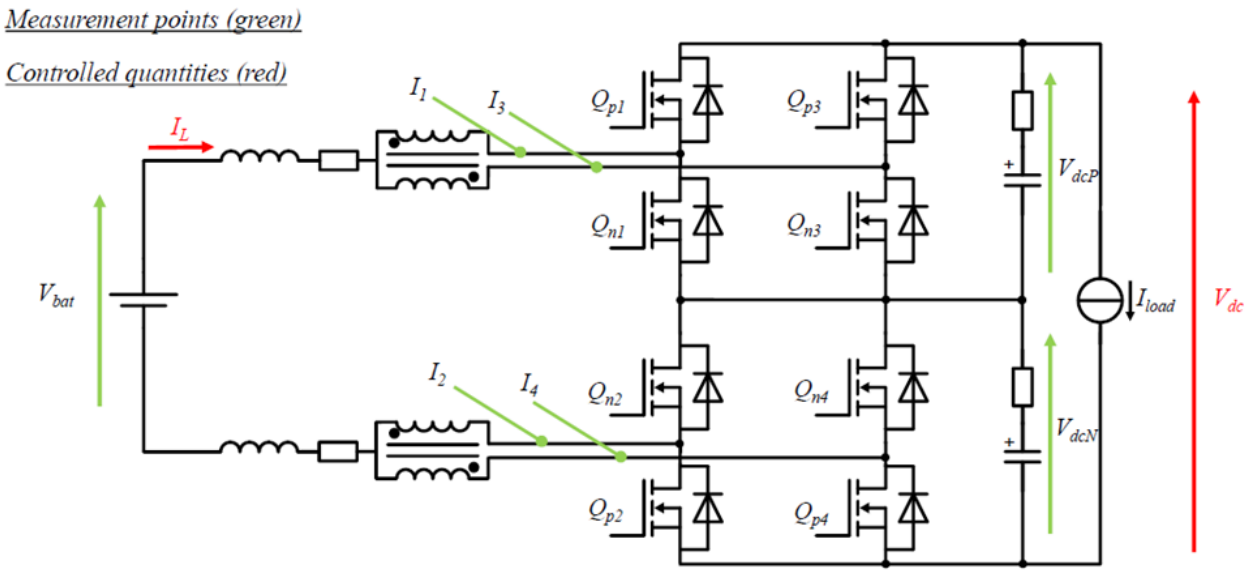

Figure 4.2: Electrical scheme of DC/DC converter

The PWM generation, for the current scenario, has multiple goals such as:

- Clock frequency of 200 MHz, maximum frequency available in Infineon board (this choice is motived later);

- 8 PWM with a frequency of 100 kHz (f_sw: frequency of switch) and upgradable duty cycle;
- 4 PWM of those mentioned above are inverse PWM;
- PWM signals and their respective complementary have a dead time of 100 ns;
- PWM signals must be shifted of some given value $(1/2 * T_{sw}, 1/4 * T_{sw}, 3/4 * T_{sw})$(Figure 4.3).

The following shown solutions is specifically designed for the ATOM submodule, but transitioning to the TOM submodule should only require minor changes to the code language.

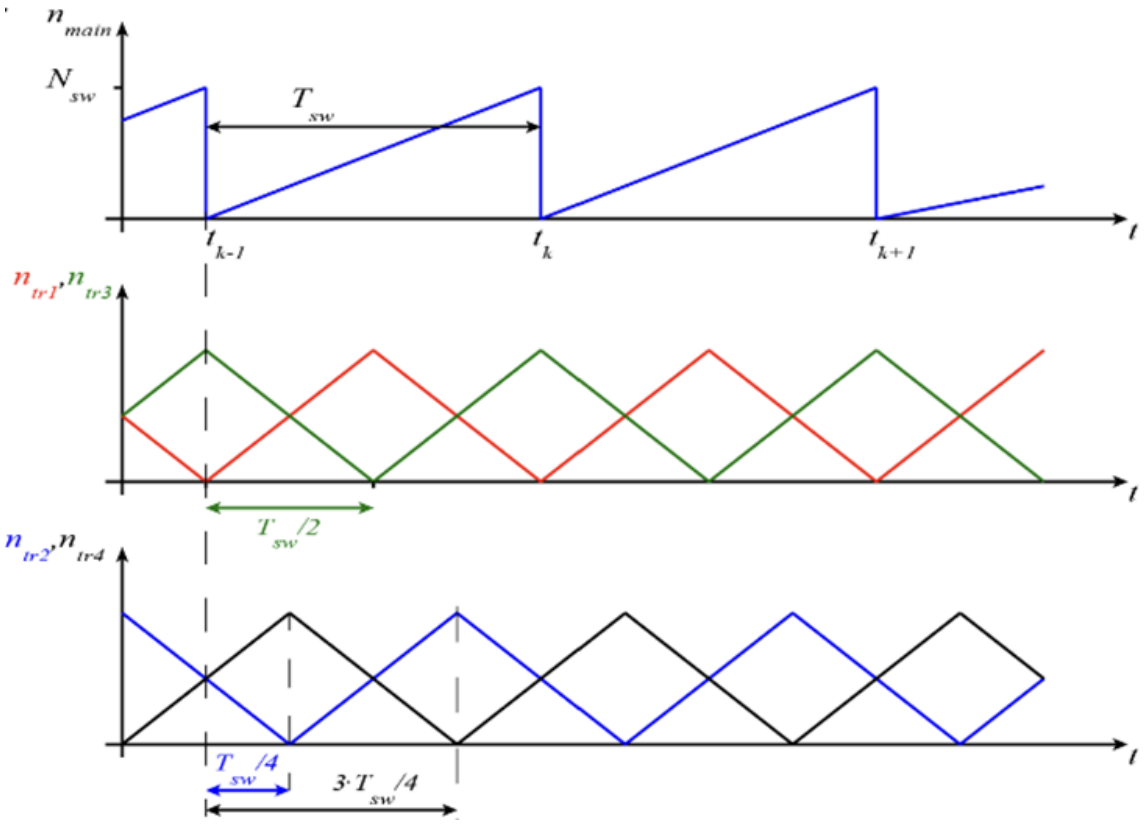

Figure 4.3: Shifted signal goal

4.2 Frequency setup

In order to accomplish the goal of a clock frequency of 200 MHz, is necessary to set the CMU (Clock Management Unit) of GTM. In the previous figure 3.6, various clocks can be observed that can be set based on the submodule used, which in this instance is the ATOM module. For ATOM module, the clocks available are eight global clocks: from CMU_CLK0 to CMU_CLK7. Choosing that the clock used will be CMU_CLK0, it is necessary to set the system clock to its maximum (200 MHz) and to program the Global Clock Divider in order to maintain this value until CMU_CLK0.

Observing a more precise imagine of CMU in Figure 4.4, the first step is to set CLS0_CLK to 200 MHz, that is GTM maximum frequency. To do that the "CLSc_CLK_DIV" bit (Cluster c Clock Divider) must be set to 01, in order to enable cluster c without clock divider, so keeping frequency to its maximum value. To modify this bit is necessary to set the "RF_PROT" bit to zero to unlock its protection and make the bit writable. To keep this value of frequency also for CMU_CLK0, the Global Clock Divider is set to 1 so to have a final clock for ATOM counters equal to 200 MHz, in order to guarantee a more precision and resolution in terms of PWM realization.
Another frequency requirement is about the PWM frequency, that has to be 100 kHz. This value is related to the clock frequency of ATOM counters through the relation:

$$\text{Period ticks of counter} = \frac{Clock frequency}{PWM desired frequency} = \frac{200MHz}{100kHz} = 2000 \text{ ticks}$$

To have the desired PWM frequency, the ATOM counter must be set to count 2000 ticks for each period of the PWM signal.
The choice of a clock frequency of 200 MHz is dictated by the fact the, as the above formula shows, increasing the clock frequency the number of counted ticks increases so it leads to have an higher resolution for signals, that for a counter count of 2000 ticks is about 11 bits.

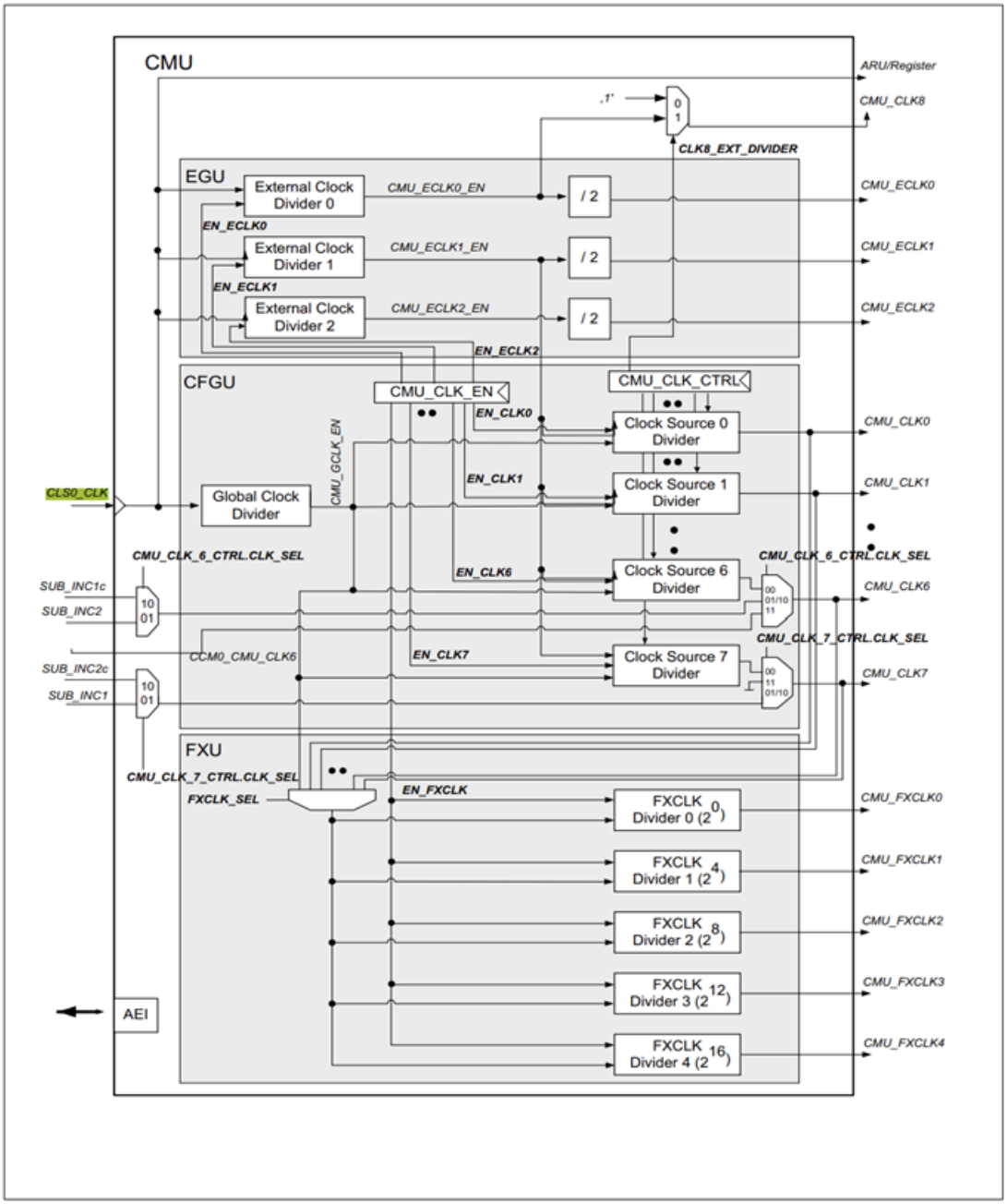

Figure 4.4: CMU detailed block diagram [11]

4.3 PWM generation task

The PWM signal must be realised through some settings, tuning the parameters behind the microcontroller logic for PWM realization. For the above purpose, the ATOM Signal Output Mode PWM (SOMP) is used. This mode is specific for PWM generation with different duty cycles and periods, that can be changed asynchronously or synchronously.
Every ATOM channel has a counter "CN0", that can work in two different modes depending on configuration of "UDMODE" bit in the control register of ATOM, and two registers: "CM0", that has the value of PWM period in ticks, and "CM1", the value of desired duty cycle in ticks. The initial signal output level for the channel is the inverse pulse level defined by the SL bit. By default, the counter counts only up until it reaches CM0 and is then reset to 0. It can also be reset by a trigger from the previous channel or by an external trigger from other submodules. Thanks to the comparison of the "CM1" value and "CN0" counter, a PWM is realized, according to the value of SL chosen. Different modes are available for the counter and the main are continuous counting up mode and continuous counting up down mode.
As shown in Figure 4.5, in continuous counting up mode, the counter counts up and then it is reset, "CM0" value must be set with the desired period and "CM1" with the desired duty cycle. By default, the counter has a starting value equal to zero, but is possible to set it to a different value, but causing a shift in the PWM signal.

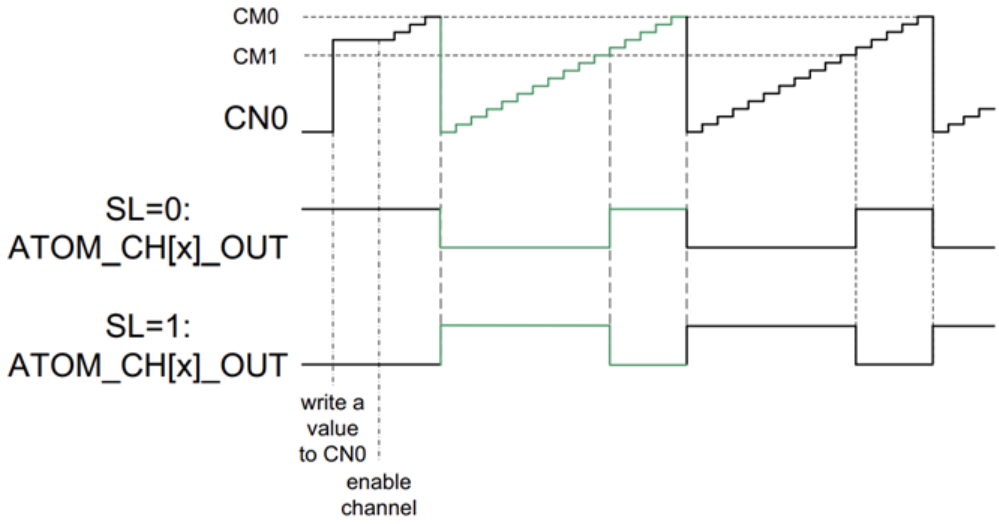

Figure 4.5: PWM output with up counting mode [11]

In Figure 4.6, in continuous counting up down mode, the counter switches between counting up and counting down, "CM0" value must be set with half of the desired period and "CM1" with half of the desired duty cycle. Also in this case, by default, the counter has a starting value equal to zero, but is possible to set it to a different value.

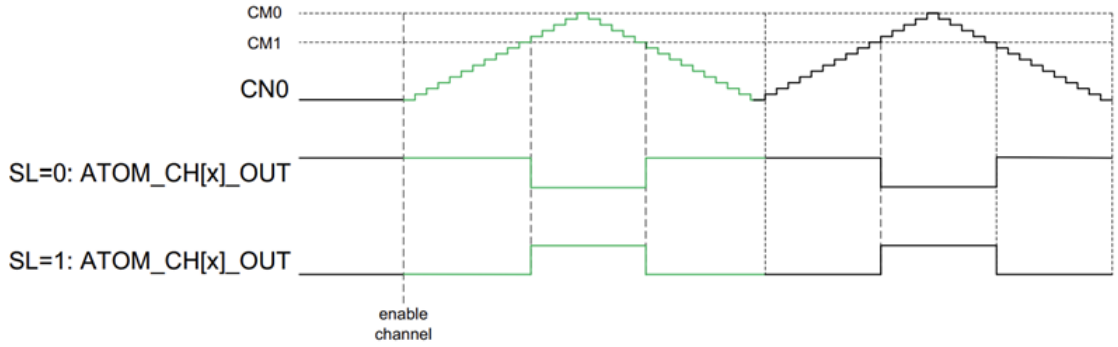

Figure 4.6: PWM output with up down counting mode [11]

To realize PWM signals the easiest way is to set these registers and the counter through some setting of the ATOM submodule. A general pseudo code for the configuration of PWM generation through ATOM is shown in Code 4.1.

```
    // This function creates a PWM and its inverse

   void initGtmATomPwm(PWM_PERIOD, dutyCycle, PIN_OUTPUT, PIN_OUTPUT_INVERSE) {

    Gtm_enable; // Enable GTM
    CMU_setClkFrequency(CMU_Clock_0, clock_frequency); // Set the CMU clock 0
    frequency
    CMU_enableClocks; // Enable the CMU clock 0
    Atom_Pwm_initConfig; // Initialize default parameters
    atom = atom_module; // Select the ATOM module
    atomChannel = channel_selected; // Select the channel
    mode = SOMP;
    period = PWM_PERIOD; // Set timer period
    dutyCycle = dutyCycle;
    pin.outputPin = PIN_OUTPUT;
    activestate= high; //SL bit state
    synchronousUpdateEnabled = TRUE; // Enable synchronous update
    Pwm_initialization;
    Pwm_start;

    // Initialization of a second channel with the PWM output inverse (same code but
     different channel and active state)
    ...
}
```

Pseudo Code 4.1: ATOM PWM generation implementation

It is also possible to update the duty cycle and the period of signals after their realization, as shown is the code created below for the duty cycle case.

```
// This function sets a new duty cycle for the specified signal PWM and for its
    inverse

void setDutyCycle(dutyCycle, signalPWM) {

    signalPWM.dutyCycle = dutyCycle; // Set new duty cycle
    Pwm_re-initialization;
    signalPWM_inverse.dutyCycle = dutyCycle;
    Pwm_re-initialization;
}
```

Pseudo Code 4.2: PWM duty cycle update implementation

In general, focusing on the codes, to realize a PWM with ATOM submodule it is necessary to create a structure that explains the main characteristics of the signal. For almost all applications, a preliminary step is always the enable of GTM and

the configuration and enable of the clock, choosing the type of clock needed and its frequency. First, the PWM can be constructed by default parameters and then, some of them, can be changed according to the requirements. Is important to set the cluster of ATOM and the channel, in which the PWM will be created. Also the mode, the values of period and duty cycle, the mode of synchronism, the mode of the counter and the physical pin must be indicated.
Choosing the value of active state, so the value of SL bit, the PWM will be created taking into account the duty cycle value inserted or its inverse, so changing this value is possible to create also the PWM inverse signal. In the end, there are several pre-set functions that are able to initialize and start the clock and the signal, with the setting chosen previously.

4.3.1 Multiple PWM generation

To create several signals, many methods are available but the simplest one is to do again the procedure just described, but using a different channel and physical pin on the board, keeping always in mind that a channel (and a physical pin on the board) can only create a single PWM signal at the time. Create several signals in a single cluster of ATOM introduces a synchronization between them because counters are all related to the same CMU clock; however, to ensure the synchronization it is better to create a timer that resets them simultaneously.
To add a timer, at the above procedure, must be added the creation of a timer in a ATOM/TOM channel belonging at the same cluster of the PWM signals. The timer has to be linked to the other channels as reset signal of their counters, in order to synchronize all the several PWM signals; ensuring synchronization but adding the need to use an additional channel for the timer realization. This method will be described better later, because there are some submodules, as the DTM, that use this logic.

4.3.2 Settings for up counting and up down counting mode

The counter "CN0" of every ATOM channel can be set in two different modes: up counting mode or up down counting mode, as mentioned before. In the control

register of ATOM, the value of "UDMODE" bits defines the chosen mode, only in SOMP mode, so to switch between a mode or another is sufficient to set these bits in different configuration:

- 00_B Up/down counter mode disabled: CN0 counts always up;
- 01_B Up/down counter mode enabled: CN0 counts up and down, CM0,CM1 are updated if CN0 reaches 0 (i.e. changes from down to up);
- 10_B Up/down counter mode enabled: CN0 counts up and down, CM0,CM1 are updated if CN0 reaches CM0 (i.e. changes from up to down);
- 11_B Up/down counter mode enabled: CN0 counts up and down, CM0,CM1 are updated if CN0 reaches 0 or CM0 (i.e. changes direction).

After the definition of these bits, another difference to focus on when changing mode, is that in up counting mode "CM0" and "CM1" are the registers where the value of PWM period and duty cycle in ticks must be inserted, while in up down counting mode in the registers must be written the half of period and duty cycle in ticks.

4.4 Dead Time Task

To prevent short circuits between the legs of a DC/DC converter, it is necessary to include dead time at the edges of PWM signals. This is due to the fact that the movement of switches is not instantaneous. To implement the insertion of dead time multiple applications are developed.

4.4.1 Dead Time Module usage

The Dead time Module (DTM) can be used to insert dead time between the PWMs. The DTM uses an ATOM/TOM timer to reset the counter of PWMs (to synchronize them also for multiple channels) and every PWM couple of signals is realized using two channels: one for bottom PWM and another one for top PWM, used to drive the switches in a complementary mode. The DTM has as input the PWM created by TOM or ATOM and as output 2 PWMs (normal and inverse) with the desired dead time, as shown in Figure 4.7, so to use DTM is needed to configure it and also

create separately a PWM signal with ATOM or TOM submodules.

Figure 4.7: Input-Output DTM block scheme

The simplest way to use DTM is with the use of pre-existent functions that changes CM0 and CM1 registers to insert the desired delay. To use them, some steps are needed to configure the DTM and the PWM settings, as shown in Code 4.3. To use functions some libraries are included as "IfxGtm_Atom_Timer.h", "IfxGtm_Atom_Pwm.h" and "IfxGtm_Atom_Dtm_PwmHl.h". To display DTM outputs, so Q and its inverse signals, is necessary to disable the outputs of ATOM, setting them to null pointers.

```
//Codes for the use of DTM for up counter (also a version for up down counter is
    available)

void initAtomPwm(void)
{
    Gtm_enable;
    Cmu_setClkFrequency(Cmu_Clk_0, CLK_FREQ);     // Set the CMU clock 0 frequency
    Cmu_enableClocks();                          // Enable the CMU clock 0

    //Definition of a timer to reset ATOM counters
    Atom_Timer_initialization();
    timer.atom = PIN_timer.atom;
    timer.Channel = PIN_timer.channel;
    timer.trigger = PIN_timer;
    timer.frequency = PWM_FREQUENCY; // CM0: clock reset, timer is set at the same
    frequency of PWM
    timer.counterDirection =CountDir_up;
    timer.trigger_enabled = TRUE;
    timer.trigger_outputEnabled = TRUE;
    timer. triggerPoint = PWM_PERIOD/2; // CM1
    timer.clock= Cmu_Clk_0;
```

```
    Atom_Timer_init();
    Atom_Timer_run();

   // To create multiple channels
    for (channelIndex = 0; channelIndex < NUMBER_OF_CHANNELS; channelIndex++){
        Atom_Pwm_init();
        atom[channelIndex].atom = atom;//atom cluster
        atom[channelIndex].atomChannel = channel;//atom channel
        atom[channelIndex].pin.outputPin = null pointers;//output pin
        atom[channelIndex].synchronousUpdateEnabled = TRUE;   // CM0&CM1 only update
     at CN0=0
        Period_ticks = CMU_CLK_FREQ_Hz / PWM_freq_Hz;
        atom[channelIndex].period = period_ticks ;
        atom[channelIndex].dutyCycle = period_ticks * duty_cycle;
        atom[channelIndex].signalLevel = ActiveState_high; // when arrive at period
    (cm0) turn high, when arrive at dc (cm1) turn low
        atom[channelIndex].dtmClockSource= cmuClock0;

        Atom_Pwm_init();
        Atom_Pwm_start();
    }
}


void initDtm(void)
{
    IfxGtm_Atom_Dtm_PwmHl_init();

    dtmPwmHl->atom          = atom;
    dtmPwmHl->deadTimeClock = dtmClock;
    dtmPwmHl->channelCount     = 2 channels;
    dtmPwmHl->outputMode       = pin.outputMode;
    dtmPwmHl->timer            = timer;

    Atom_Topchannels ccx[]      = {&CCX0, &CCX1};//output pins ATOM2_2 e ATOM2_3
    Atom_Bottomchannels coutx[] = {&COUTX0, &COUTX1};//output pins ATOM2_2N e
    ATOM2_3N

    dtmPwmHl.ccx            = ccx; //output pin top signals
    dtmPwmHl.coutx          = coutx;//output pin bottom signals

    dtmPwmHl.ccxOutputEnabled      = ActiveState_low;
    dtmPwmHl.coutxOutputEnabled    = ActiveState_high;//to create inverse signal
    dtmPwmHl.ccx.ActiveState        = ActiveState_low;
    dtmPwmHl.coutx.ActiveState      = ActiveState_high;
    dtmPwmHl.deadtime  = DEADTIME;
    Atom_Dtm_PwmHl_init();
}
```

Pseudo Code 4.3: DTM usage implementation

Code is formed by two functions, because this application involved both the usage of ATOM module and DTM module. The simplest steps to realize PWM with DTM are, first of all, to configure ATOM to create multiple PWM and a timer, that will reset ATOM's counters. Then, the DTM must be configured, passing it the timer and PWM created by ATOM.
Observing the pseudo-code:

- GTM module and a clock are set and enabled;
- An ATOM timer is defined and enabled with its module, channel, frequency (equal to PWM frequency) and trigger point (half of PWM period). It must be also set in trigger mode.
- Multiple ATOM PWM are created with their modules, channels, period and duty cycle as in Algorithm 4.1, but, in addition, the DTM source clock is specified and the physical pins of PWM are indicated as a null pointer because the output will be that of DTM, not ATOM.
- DTM is initialized by pointing at it the ATOM module (where PWM and timer are created), the clock used, how many PWM and inverse signals will be created, the timer for reset and the dead time desired. Two arrays are constructed: one for top channels (will contain PWM channels, that must be adjacent channels) and one for bottom channels (with the complementary PWM, that must be adjacent channels too). These two arrays contain the outputs that will be displayed and used for managing FETs.
- The ccx and coutx arrays are configured in order to guarantee that one is the inverse of the other, so configuring with opposite value their active state (SL bit).
- Functions are used to initialize and start the signal generations.

The library mentioned above links ATOM channels and DTM to create multiple channels with the desired dead time. It changes, with its functions, the registers CM0 and CM1, and synchronizes all signals by using the ATOM timer to reset the counters. The use of DTM is, thanks to the timer, very precise in terms of

synchronization of signals, but the delay between the input of DTM and outputs could create some problems in some applications. This delay can shift too much the carrier of modulation of PWM, also compromising the control algorithm.

The use of DTM above described, can be completed with some considerations applicable only in up counting mode, observing that in this mode the signal, with an high active state setting, when arrive at period (CM0) turn high, when arrive at dc (CM1) turn low. Taking into account these considerations, it is also possible to manage the shift task and mode of alignment (left, right, center) with a mathematical strategy, realizing a function for that.

Let's call:

- Dc: duty cycle desired;
- A: shift desired;
- DcStart: ticks for the start of high part of PWM;
- DcEnd: ticks for the end of high part of PWM;

Algorithm 1 shows the procedure for DC start calculation and Algorithm 2 for DC end calculation.
These values of dcStart and dcEnd can be put as period and duty cycle in PWM creation in order to achieve the desired shift and alignment. For the requirements considered in this thesis work, to generate 8 PWM, an application with 9 channels is required, using DTM module: this includes 8 channels for PWM and 1 channel for the timer. In order to guarantee the synchronization, it is better to create PWMs in a single cluster so this strategy can be applied only in TOM submodule, if the purpose is to create more than 7 channels of PWM, because it has sufficient channels (more than nine channels in each cluster).

Algorithm 1 Duty cycle start calculation

Use 0.5 * for center alignment;
Use 1 * for right alignment;
Use 0 * for left alignment;

```
dcStart = 0.5*(1-dc);                    ▷ Set center alignment
dcStart = dcStart + a;                   ▷ Add phase shift
dcStart = dcStart * period;              ▷ Convert to clock ticks
```

Ensure value is within range 0 to period (in ticks) to make sure count values are reached:

```
if dcStart is greater than (float32) period then
    dcStart -= period;
else
    if dcStart is smaller than 0.0 then
        dcStart += period;
    end if
end if

return dcStart;
```

Algorithm 2 Duty cycle end calculation

```
dcEnd = dcStart + (dc*period);
```

Ensure value is within range 0 to period (in ticks) to make sure count values are reached:

```
if dcEnd is greater than (float32) period then
    dcEnd -= period;
else
    if dcEnd is smaller than 0.0 then
        dcEnd += period;
    end if
end if

return dcEnd;
```

4.4.2 Use of the functions of PwmAtomHl.h instead of DTM

In order to avoid the mentioned above DTM delay between its inputs and outputs, another strategy for dead time introduction could be the use of the public library PwmAtomHl.h (or TOM). The standard interface PWM high/low (PwmHl) abstracts the hardware used for the pwm feature and it provides, after proper initialization, an hardware-independent way to interact with the PWM functionality like setting duty cycles, dead time, timer functionalities ext. The main concepts are the same of those of DTM, but in this case there is not a specific module that do the various steps (DTM), but everything is implemented by functions that manage the ATOM channels.
In order to utilize the library functions, an ATOM timer is set up to reset the counters of PWMs. In the case of up down counter the timer is not responsible for the counter reset, but for the change of count direction. ATOM counters of PWM and their inverse signals are initialized and the settings of duty cycle, dead time and centre alignment are done by standard functions.
With respect to the use of Dead Time Module no delays are introduced and dead time is realized internally by function that manipulates CM0 and CM1 registers. A general pseudo code is shown in Code 4.4, where two functions, one for up counting mode and one for up-down counting mode are described.

```
//Codes for library HL PWM generation and dead time
#include "IfxGtm_Atom_PwmHl.h"

void ATomPwmUpcounter(){

    GTM_enable();/* Enable GTM       */
    //SET THE CLOCK SOURCE TO ITS MAXIMUM SO 200 MHZ
    CTRL.RF_PROT=0; //RF_PROT bit cleared in order to set CLK_DIV
    CLS_CLK.CLS0_CLK_DIV=01; //Set cluster enable without clock divider
    Cmu_setClkFrequency(Cmu_Clk_0, 200000000);     // Set the CMU clock 0 frequency
    Cmu_enableClocks();                    // Enable the CMU clock 0

    //Configuration of the reset timer of atom counters
    Atom_Timer_init ();    //To set default parameters for timer
    Timer.atom = PIN_CLOCK.atom;
    timerChannel = PIN_CLOCK.channel;
    timer.trigger= &PIN_CLOCK;
    //Change this parameter to change the frequency of pwms displayed
    timer.frequency = PWM_FREQUENCY;                    // CM0: clock reset
    timer.countDir = CountDir_up;
```

```
    timer.trigger.enabled = TRUE;
    timer.trigger.triggerPoint = 0.5*200000000/PWM_FREQUENCY ; //CM1
    timer.clock= Cmu_Clk_0;
    timer.trigger.risingEdgeAtPeriod = TRUE;
    Atom_Timer_init_and_run();

    // Create ATOM configuration
    Atom_PwmHl_initConfig();       //For default configurations
       //For desired configurations
    pwmHl.timer=&g_timerDriver;
    pwmHl.outputMode = IfxPort_OutputMode_pushPull;
    pwmHl.channelCount=1;
    pwmHl.deadtime=0;// or 100e-9 to set 100 ns of dead time
    pwmHl.ccx = ccx;
    pwmHl.coutx = coutx;
    pwmHl.ccxActiveState   = ActiveState_high;
    pwmHl.coutxActiveState = ActiveState_high;

    // initialize the ATOM
    Atom_PwmHl_init();
    //Counters all have to start from 0
    ATOM[0].CH0.CN0 =0;
    ATOM[0].CH1.CN0 =0;

    //To set duty cycle of PWMs Ton must be set in ticks
    TimerValue onTime[1]; // assume configured for 1 HL channel
    onTime[0] =    0.5* 200MHz/PWM_FREQUENCY; //in ticks, so DC of 30% in a period
    of 2000 ticks must be 600 ticks
    Atom_PwmHl_setMode(Mode_centerAligned);
    Timer_disableUpdate(timer);
    PwmHl_setOnTime();
    Timer_applyUpdate(timer);
}


//Functions for up-down counting
void ATomPwmUpDowncounter(){

    //same code of up counting but with these lines in addition
    timer.frequency = PWM_FREQUENCY*2;

    //To  set up down counting mode of counters of ATOM
    ATOM[0].CH0.CTRL.UDMODE=3;
    ATOM[0].CH1.CTRL.UDMODE=3;
}
```

Pseudo Code 4.4: PWMAtomHl.h usage implementation

The pseudo code summarizes the main steps for the use of the functions of the library. After the enable of the clock, in a channel of the ATOM module, a timer must be

set and configured as trigger for ATOM counters. The timer has the same frequency of the desired PWM, a duty cycle of 50% and it is chosen with counting up mode. Then, functions and structures of the library are used and so it is necessary to link the timer to the counters, indicate the number of complementary PWM signals and the desired dead time and create two arrays with the pin, cluster and channel where signals will be created: "ccx" for top channels and "coutx" for bottom channels, as in Figure 4.8.

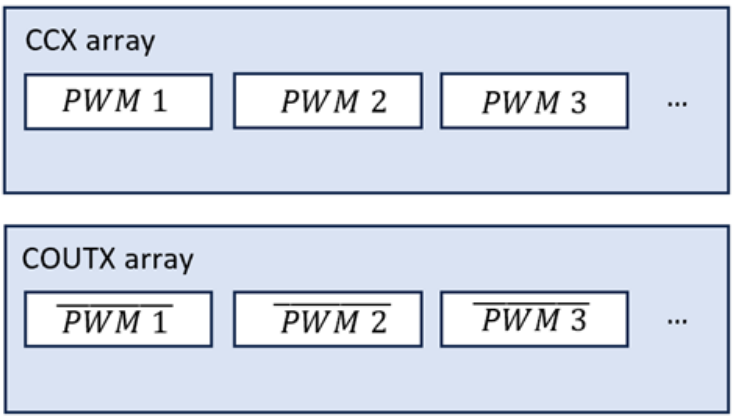

Figure 4.8: CCx and COUTx array structures

In order to establish PWM duty cycles, it is necessary to create an array with the same length as the number of signals where the "on time" is indicated. This array will contain duty cycle values in ticks, as shown in Figure 4.9. For example, the first PWM and its inverse have 600 ticks of duty cycle, the second one has 700 ticks, and the third one has 600 ticks.

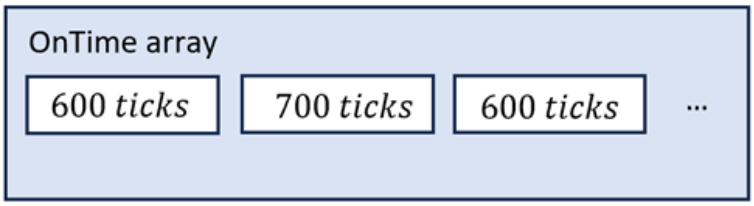

Figure 4.9: OnTime array structure

In order to apply the desired duty cycle vector to the ATOM channels, the timer must

be stopped first. Then, the on time must be set and an update should be applied to restart the PWM generation.
When dealing with up-down counting mode of counters, the main difference in settings is to change a bit in the control register (UDMODE bit) to set the counters in up-down mode. Additionally, the frequency of the timer (which always counts up) should be doubled to achieve the desired PWM frequency.

This method involves only the ATOM submodule and with different settings of active state (to create the inverse), manipulation of CM0 and CM1 (for dead time insertion) and the use of a timer to reset the counters creates the desired multiple signals with a very precise synchronization between them. Being involved only the ATOM submodule, the intrinsic delay of DTM is overcame, so this application is better than the previous one because it doesn't lead to a shift of the modulating carrier due to the delay. If the desired number of signals is over 7 is necessary to do the same steps but using the TOM submodules, because it is equipped with more channels than ATOM and for this application an additional channel must be considered for the creation of the timer.

4.4.3 Duty cycle manipulation

Another way to create dead time could be the handling of duty cycle register without using standard functions but realizing it step by step, taking into account some aspects: the first one is that 100 ns, the example desired dead time, corresponds to 20 ticks because the CMU clock is 200 MHz and the PWM frequency is 100 kHz so the period ticks are:

$$\text{Period ticks of counter} = \frac{Clock frequency}{PWM desired frequency} = \frac{200MHz}{100kHz} = 2000 \text{ ticks}$$

Consequently, if a period of 100 kHz corresponds to 2000 counter ticks and:

$$\text{PWM period} = \frac{1}{PWM frequency} = \frac{1}{100kHz} = 10\mu s$$

The dead time of 100 ns corresponds to:

$$\text{DeadTime}_{seconds} \div \text{Period}_{seconds} = \text{TicksOfDeadTime} \div \text{TicksOfPeriod}$$

$$100 \text{ ns} \div 10 \ \mu \text{ s} = \text{TicksOfDeadTime} \div 2000 \text{ ticks}$$

That done is equal to:

$$\text{TicksOfDeadTime}= \frac{100ns*2000(ticks)}{10\mu s} = \frac{200\mu s}{10\mu s} = 20 \text{ ticks}$$

So, a quantity of microseconds as that of this desired dead time corresponds to 20 counter ticks. The second step is to observe our purpose in Figure 4.10, so focusing on the graphic image of the insertion of dead time to understand how it changes the signals Q and Q' with respect to the simple initial PWM.

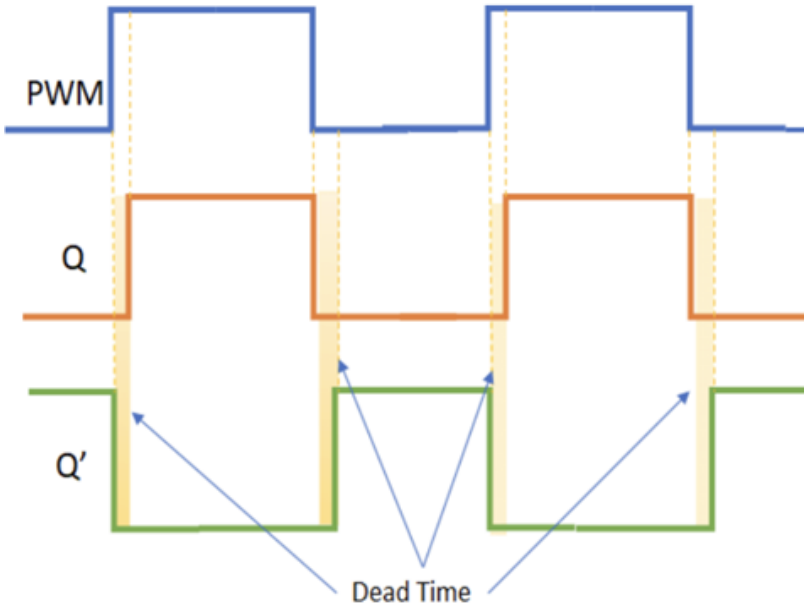

Figure 4.10: Dead time symmetric insertion

The image shows a symmetric insertion of dead time of Q and Q'. Focusing on Q with respect to the PWM signal, the rising edge of Q is shifted of the dead time value and the falling edge is aligned. Given these considerations, the duty cycle of Q signal is slightly decreased of dead time value so, in order to create the dead time insertion manually, the Q signal has a duty cycle of:

$$\text{For up counting case} \rightarrow \quad DutyCycle_Q = DutyCycle_{PWM} - 20 \text{ ticks}$$

$$\text{For up down counting case} \rightarrow \quad DutyCycle_Q = 1/2 * DutyCycle_{PWM} - 10 \text{ ticks}$$

For up down counting case the calculation is the same but everything is divided by 2 because the microcontroller, for this mode, wants in the duty cycle field half of the desired duty cycle.
Focusing on Q', the falling edge of Q' is aligned with the rising edge of PWM and the rising edge is shifted of the dead time value with respect to the falling edge of PWM. Given these considerations, the duty cycle of Q' is the complementary of that of Q and in addition twice dead time values, so:

$$\text{For up counting case} \rightarrow \; DutyCycle_{Q'} = (DutyCycle_{PWM} - 20 \text{ ticks }) + 40 \text{ ticks}$$

$$\text{For up down c. case} \rightarrow \; DutyCycle_{Q'} = (1/2 * DutyCycle_{PWM} - 10 \text{ ticks}) + 20 \text{ ticks}$$

For the considerations about the alignment of these signals is possible to slightly shift them changing the starting value of counters, as will be explained later, but this is not a significant problem because also asymmetric signals may be fine for many applications. These values of modified duty cycles must be set only for duty cycles different from 0% and 100% and, as said before, a little shift could be inserted to align the end of the main PWM and Q to create a symmetric PWM generation with dead time. An example of code is shown in the pseudo code below.

```
//Codes for manual insertion of dead time
//This function initializes the ATOM to create a PWM signal + its inverse signal

void initGtmATomPwm_inverse(PWM_PERIOD, dutyCycle, clkIndex, CLK_FREQ, OUTPUT,
    OUTPUT_INVERSE)
{
    Gtm_enable(); // Enable GTM
    ...
    Cmu_enableClocks();   // Enable the CMU clock 0

    //Configuration of the PWM signal
    Atom_Pwm_init();       // Initialize default parameters
    ...
    atom[i].period = PWM_PERIOD;   //Set timer period

    if(dutycycle!= 0 or 100)
        atom[i].dutyCycle = dutyCycle-10;// because it is an up down counter
    else
        atom[i].dutyCycle = dutyCycle;
```

```
    atom[i].signalLevel = ActiveState_low;
    atom[i].mode = Mode_outputPwm;
    atom[i].synchronousUpdateEnabled = TRUE; // Enable synchronous update
    atom[i].outputPin = &OUTPUT; //Set a PIN as output
    atom[i].outputMode = Mode_pushPull;

    Atom_Pwm_init();  // Initialize the PWM
    Atom_Pwm_start(); // Start the PWM


   //Initialization of a second channel with the PWM output inverse with same code
    of before

    Atom_Pwm_init();      // Initialize default parameters
    atom_inverse[i].atom = OUTPUT_INVERSE.atom;  // Select the ATOM cluster
    ...
    atom_inverse[i].period = PWM_PERIOD; // Set timer period

    if(dutycycle!= 0 or 100)
        atom_inverse[i].dutyCycle = dutyCycle-10+20;
    else
        atom_inverse[i].dutyCycle = dutyCycle;

    atom_inverse[i].signalLevel = ActiveState_high;  // Set the SL bit= 0 so to have
     an inverse signal

    Atom_Pwm_start();     // Start the PWM

    //Index (of signal) increment and check to not exceed fixed dimension of vectors
     if(i<4)
             i++;

    //To set up-down counting mode of all channels
    ATOM[0].CH0.CTRL.UDMODE=0x3;
    ATOM[0].CH1.CTRL.UDMODE=0x3;

}
```

Pseudo Code 4.5: Dead time insertion with duty cycle manipulation implementation

The procedure for creating PWM is the same as previously explained, with the only difference being the control over the duty cycle value (if-else check), which determines whether or not the duty cycle value changes.
A problem of this application could be that, if used without the use of a timer that resets the counter of PWMs, the synchronization of PWMs must be guaranteed in some way, for example triggering a channel by the previous channels, ext. Instead, if a timer is added at this application to synchronize, this method turns out to be very

effective.

4.5 Shift task

The task of shifting signals requires to realize a shift between PWMs of some given value $(1/2 * T_{sw}, 1/4 * T_{sw}, 3/4 * T_{sw})$, as in Figure 4.3. For the shift the simplest strategy is to set the initial value of counters as:

- CNT 1 = 0 ticks;
- CNT 3 = 1000 ticks;
- CNT 4 = 500 ticks;

in order to give them different initial values, the counters then count up and this leads to a signal shift.
CNT 2 is a special case because only up-counting is possible in ATOM/TOM, when a starting value is given, so it is initially set to a random value and then, after $T_{sw}/4$, it is set to zero in order to create the right shift after the first period on.
This strategy uses a WaitTime function and it could be not so precise then, so a better method is to set the counter number 2 to 500 ticks but with the opposite active state than the usual (complementary duty cycle).
The described strategy is visible in detail in figure 4.11.
Other methods could be the use of DTM shift skill or triggering a channel by the previous channel but for these multiple shift requirements the above strategy is the simplest and better one.

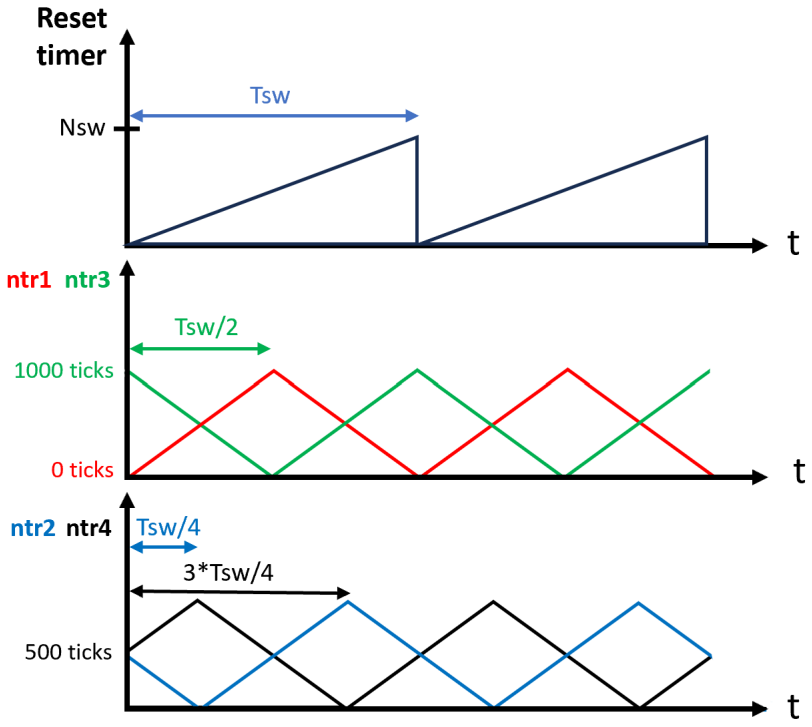

Figure 4.11: Detail of shifting of signals

Chapter 5

Experimental results

5.1 Workbench and setup

The simulations utilize the Infineon TC3x9 board and a digital oscilloscope, as shown in Figure 5.1. Cables and probes are also used to lead the signals from the board to the oscilloscope. The used integrated development environment (IDE) is AURIX Development Studio, equipped also by a debugger to check code quality.
Quite all development tests are conducted with a CMU clock frequency of 200 MHz and a PWM frequency of 100 kHz.

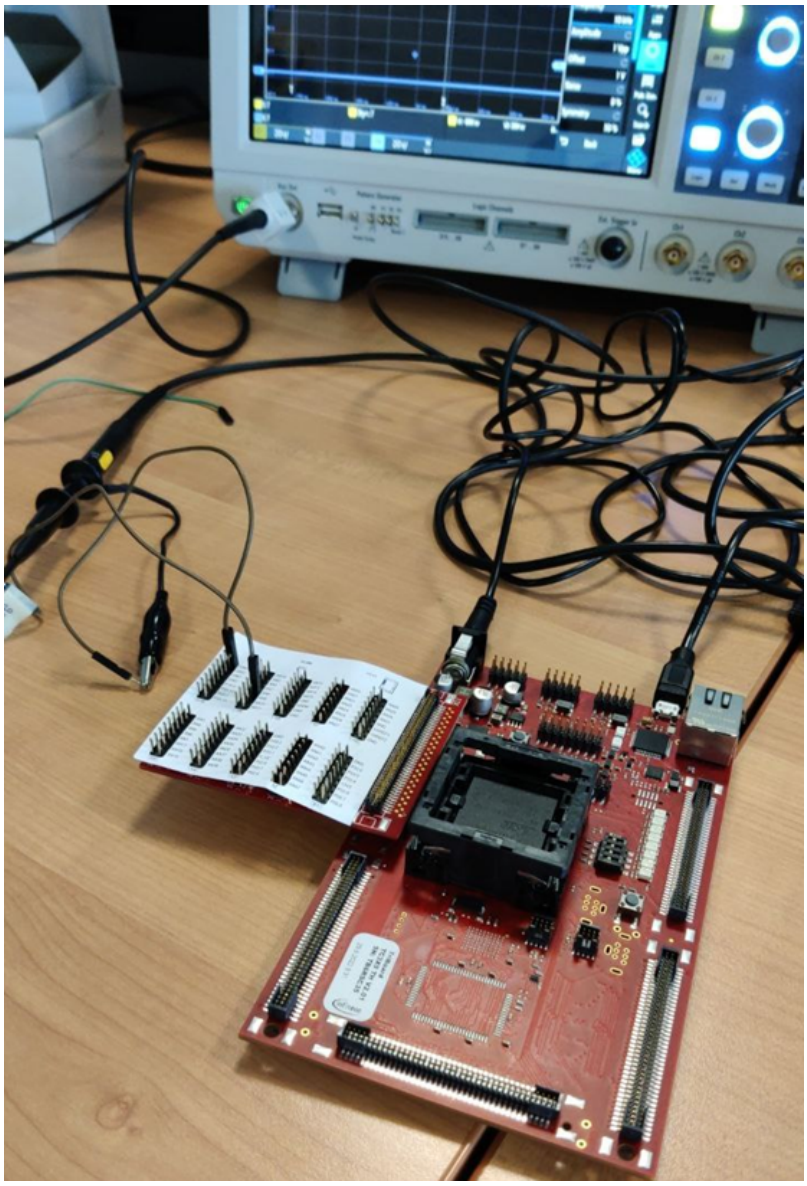

Figure 5.1: Workbench with board and oscilloscope

5.2 Simulations of PWM generation

The simulations are realized running the code on the board and visualizing the two output signals on the oscilloscope, with the use of two probes. Using the Algorithm 4.1, so a standard PWM generation, the experimental result shown in Figure 5.2 is obtained. It shows a PWM and its complementary signal, both at 200 Hz, with a

CMU frequency of 1 MHz, because in this preliminary test it was decided to work with lower frequencies initially. The C1 signal, the yellow one, is a PWM signal with duty cycle of 20% and C2 signal, the green one, is its complementary PWM, indeed it has a duty cycle of 80%. The experimental result is good, because corresponds to the commands set in the code.

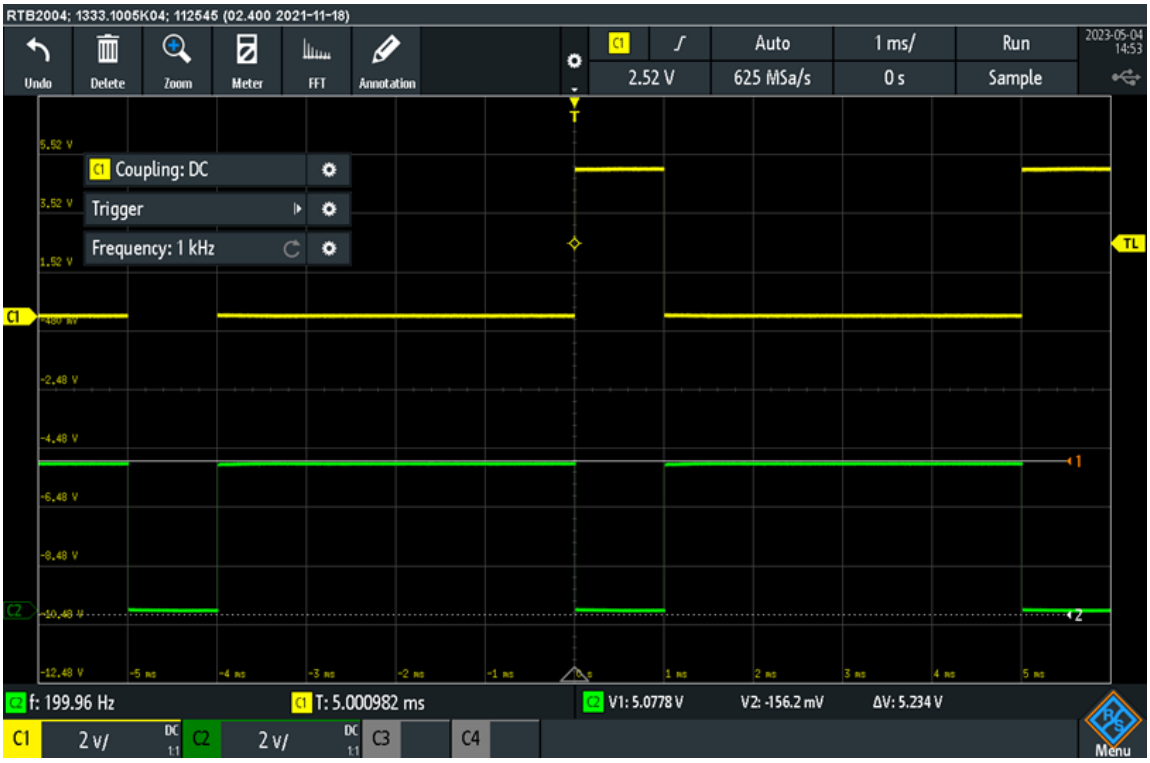

Figure 5.2: Oscilloscope view of signals for standard PWM generation

It is possible to generate multiple PWM signals in two separate channels by slightly modifying pseudo code 4.1 and rewriting twice the code adjusting the duty cycle, period or everything is needed. A test to check if multiple PWM can be created is shown in Figure 5.3. It shows two PWM signals edge aligned, both at 200 Hz, with a CMU frequency of 1 MHz, but it also possible to change the frequency of one of the two. The C1 signal, the yellow one, is a PWM signal with duty cycle of 40% and C2 signal, the green one, is another one with duty cycle equal to 20%. The experimental result is good, because corresponds to the commands set in the code.

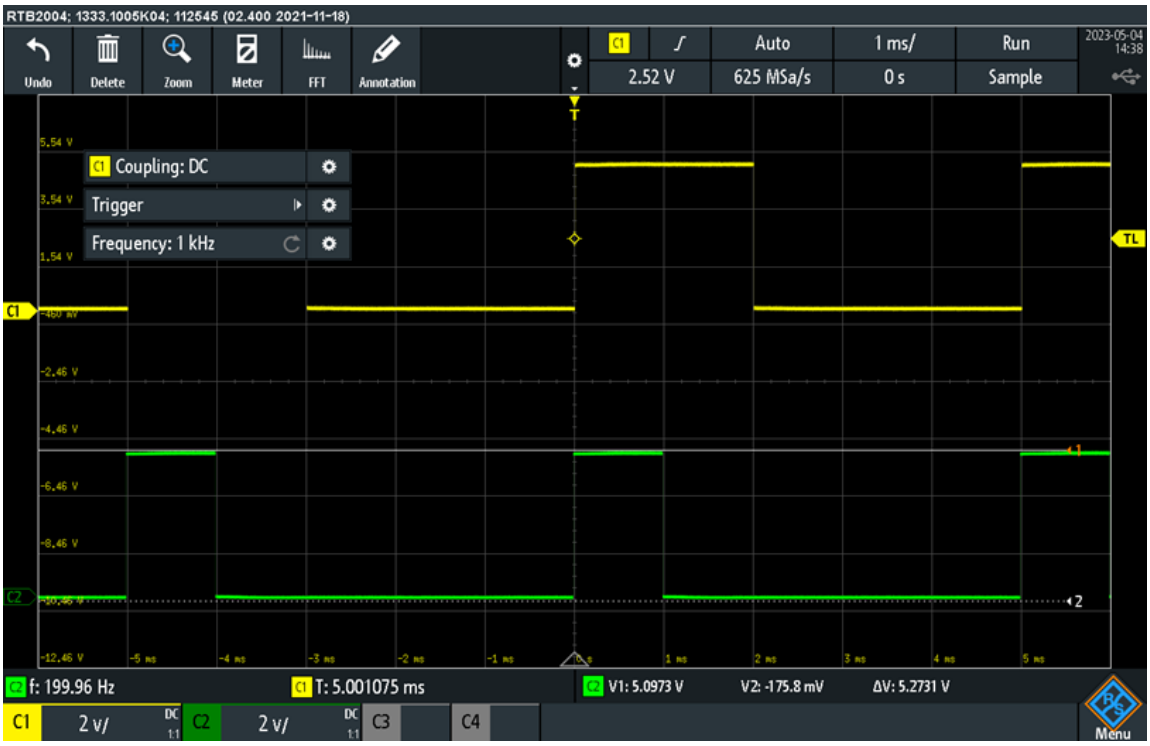

Figure 5.3: Oscilloscope view of edge aligned signals

The generation of complementary PWM signals is also possible using the Dead time Module (DTM) and the experimental result is quite the same of Figure 5.2 if a dead time equal to zero is set. Making a focus on the realised signals, the standard PWM generation leads to problem of synchronization because the counters of ATOM don't start together instantaneously, so a little delay of ns is present between signals, due to the fact that the code is executed sequentially and no reset timer exists to synchronize counters. The generation of complementary PWM signals done by DTM implements, instead, a timer that reset the counters simultaneously and so with this strategy no delays are present between signals but they are totally synchronized.
So, in general, applications with a timer that resets ATOM counters is preferable.

5.3 Simulations of dead time insertion

Using the pseudo code 4.3, or 4.4, or 4.5, the experimental result is quite the same and it is shown in Figure 5.4 and more in detail in Figure 5.5. It shows a PWM and

its complementary signal with dead time insertion (of 100 ns), both at 100 kHz, with a CMU frequency of 200 MHz. The C1 signal, the yellow one, is a PWM signal with duty cycle of about 20% and C2 signal, the green one, is its complementary PWM. The value of duty cycle is not precise because the insertion of dead time slightly modifies the duty cycle value of both PWM signals. The experimental result is good, because corresponds to the commands set in the code, and observing the Figure 5.5 a focus on the dead time of 100 ns is shown.

Figure 5.4: Oscilloscope view of complementary PWM with dead time

Figure 5.5: Oscilloscope view of dead time insertion detail

Also in this case, using the pseudo code 4.3, since there is a timer, the signals are synchronized, but the modulation triangola is slightly shifted for the intrinsic delay of DTM. With pseudo code 4.4, so the use of PwmAtomHl.h library, the delay of DTM is not present and the signal are synchronized thanks to a timer.
The duty cycle manipulation, as in pseudo code 4.5, is the less precise method because, without a timer, signals are not synchronized so it would be necessary to add a way to reset counters together such as an external or internal trigger.

5.4 Simulations of signal shifts

For the shifting of signals, the experimental result is visible in Figure 5.6, 5.7, 5.8. It shows two PWM signals shifted of some given values, both at 100 kHz, with a CMU frequency of 200 MHz and a duty cycle of 50% to make clearer the quantity of the shift realized. In Figure 5.6 a shift of $\frac{3}{4}$ Tswitch is realized, where the yellow signal has a starting value of its counter equal to zero and the purple one equal to 500 ticks.

Figure 5.6: Shift of 3/4 between signals

In Figure 5.7 a shift of $\frac{1}{2}$ Tswitch between signals is realized, where the yellow signal has a starting value of its counter equal to zero and the purple one equal to 1000 ticks.

Figure 5.7: Shift of 1/2 between signals

In Figure 5.8 a shift of $\frac{1}{4}$ Tswitch between signals is realized, where the yellow signal has a starting value of its counter equal to zero and the purple one equal to 500 ticks, but in a complementary duty cycle mode or opposite active state.

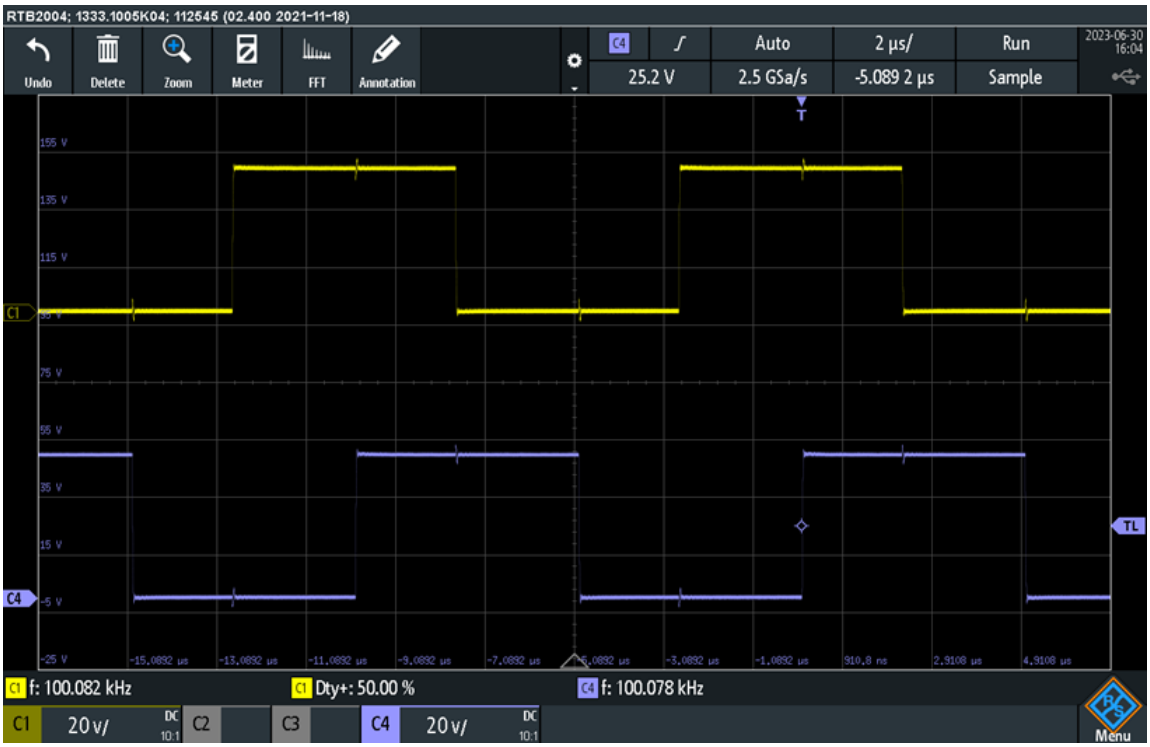

Figure 5.8: Shift of 1/4 between signals

5.5 Up counter case

The up counter case, thanks to the architecture of ATOM Module and the generation structure, is an easier case that can be treated separately, using also mathematical tricks as those shown in Algorithm 1 and 2, so the duty cycle start and end calculation functions that set the desired duty cycle, alignment and shift only with a mathematical calculus. Figure 5.9 shows two PWM signals with duty cycle equal to 20% and 30% center aligned, both at 100 kHz, with a CMU frequency of 200 MHz, while in Figure 5.10 two signals with 30% of duty cycle are realized with a shift of +0.3 between them.

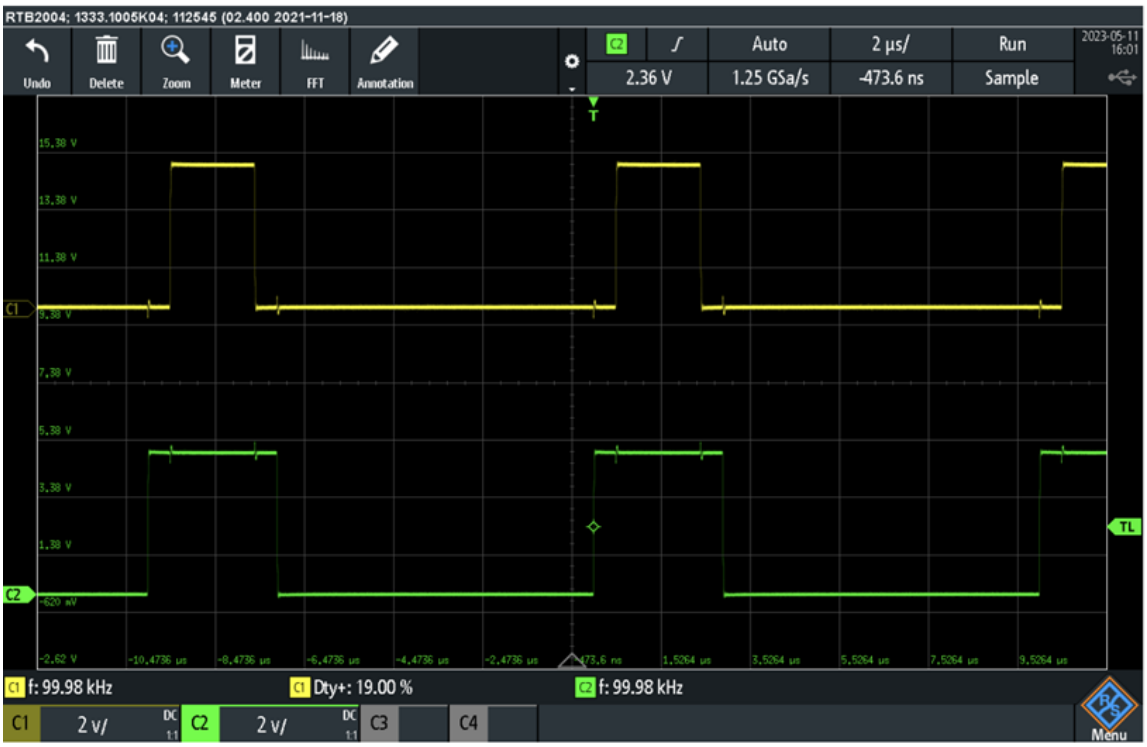

Figure 5.9: Center alignment example with up counter

Figure 5.10: Shift example with up counter

5.6 Limit of frequency

Using a random application such as that of PwmHl library for up counters, simulations are done in order to check the frequency existence domain, always maintaining a clock frequency of 200 MHz. It's not necessarily a problem to have a low frequency of PWM. However, it could be worthwhile to investigate the upper limit of PWM frequency. This is because a physical switch takes time to close and open, and doesn't do so instantly. It's important to note that increasing frequency can lead to a decrease in signal resolution (counter counts fewer ticks). Therefore, checks on the signal's integrity at high frequencies are performed. In figure 5.11 a signal of 1 MHz and 99% of duty cycle is shown: counter counts about 200 ticks and the signal is quite not clear. Figure 5.12 shows the detail of the previous signal.

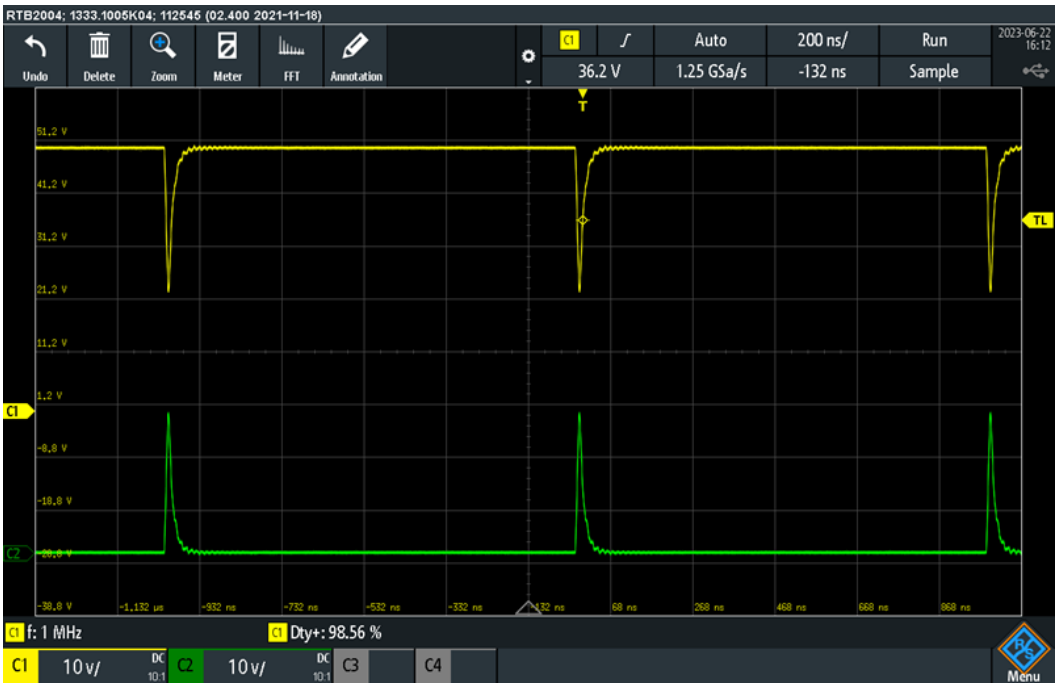

Figure 5.11: 1 MHz signal and 99% duty cycle

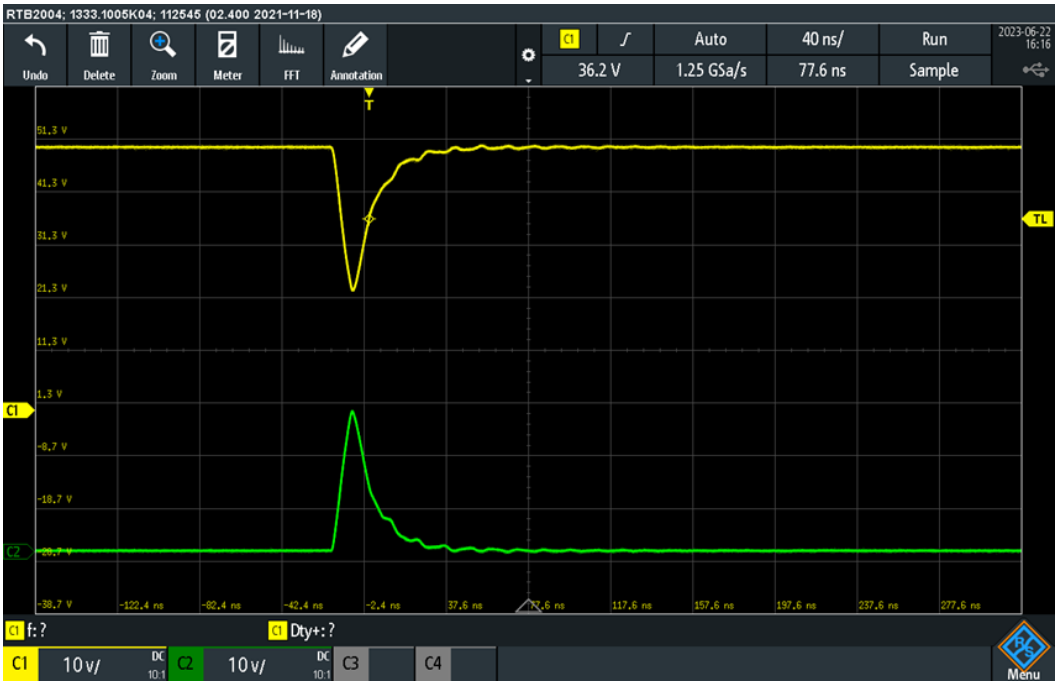

Figure 5.12: Detail of 1 MHz signal and 99% duty cycle

In figure 5.13 and 5.14 signals of 3 MHz and 30% and 90% of duty cycles are shown: counter counts about $200/3 = 66$ ticks and the signal is quite not clear, especially when duty cycle is higher and so the opening and closing movement is faster.

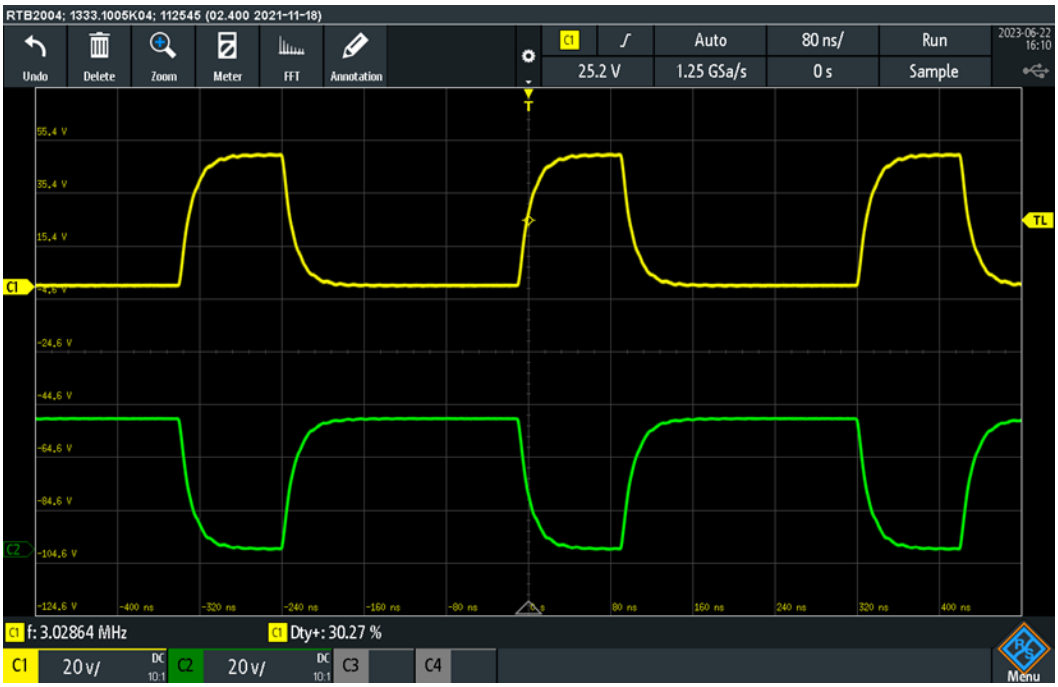

Figure 5.13: 3 MHz signal and 30% duty cycle

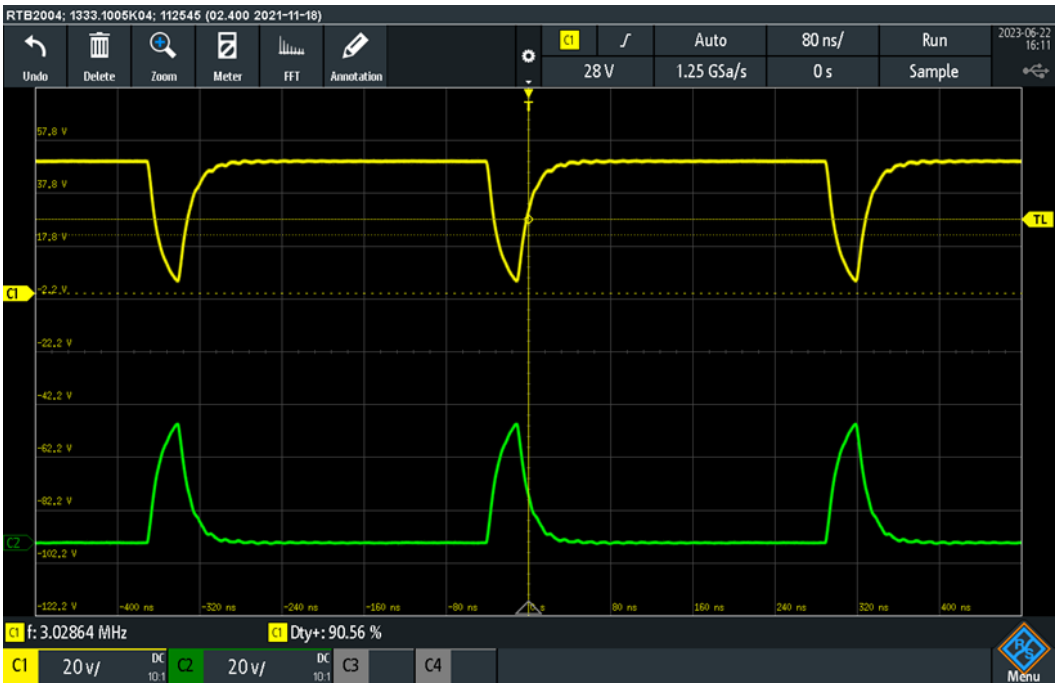

Figure 5.14: 3 MHz signal and 90% duty cycle

In figure 5.15 a signal of 5 MHz and 95% of duty cycle is shown: counter counts about $200/5 = 40$ ticks and the signal is very compromised.

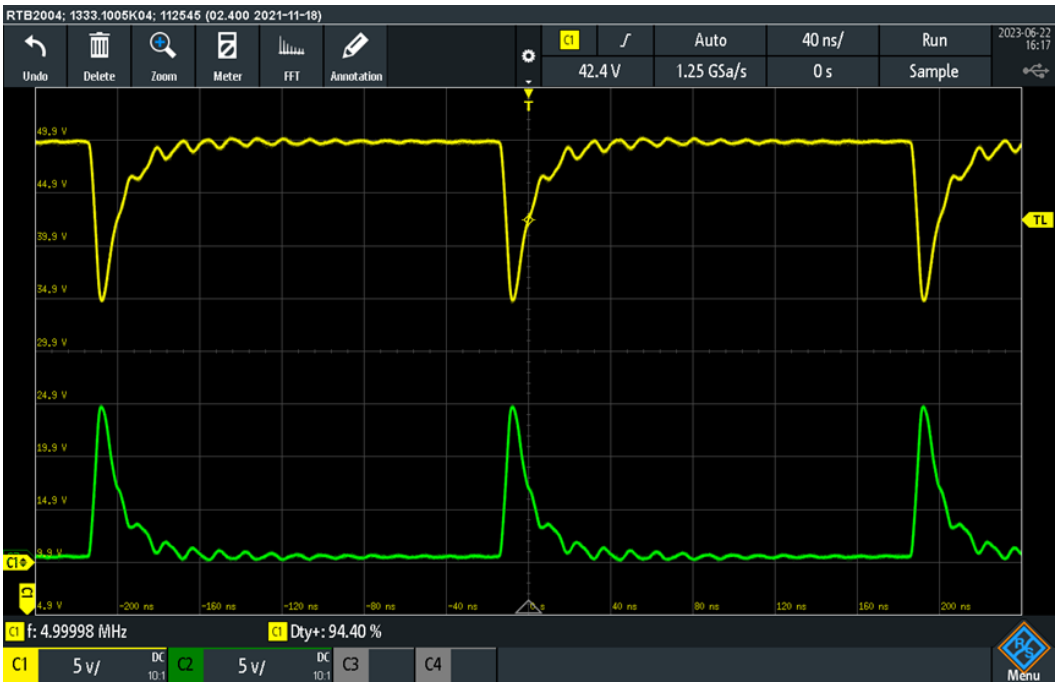

Figure 5.15: 5 MHz signal and 95% duty cycle

In figure 5.16 and 5.17 signals of 8 MHz and 50% and 90% of duty cycle are shown: counter counts about $200/8 = 25$ ticks and signals are very not clear.

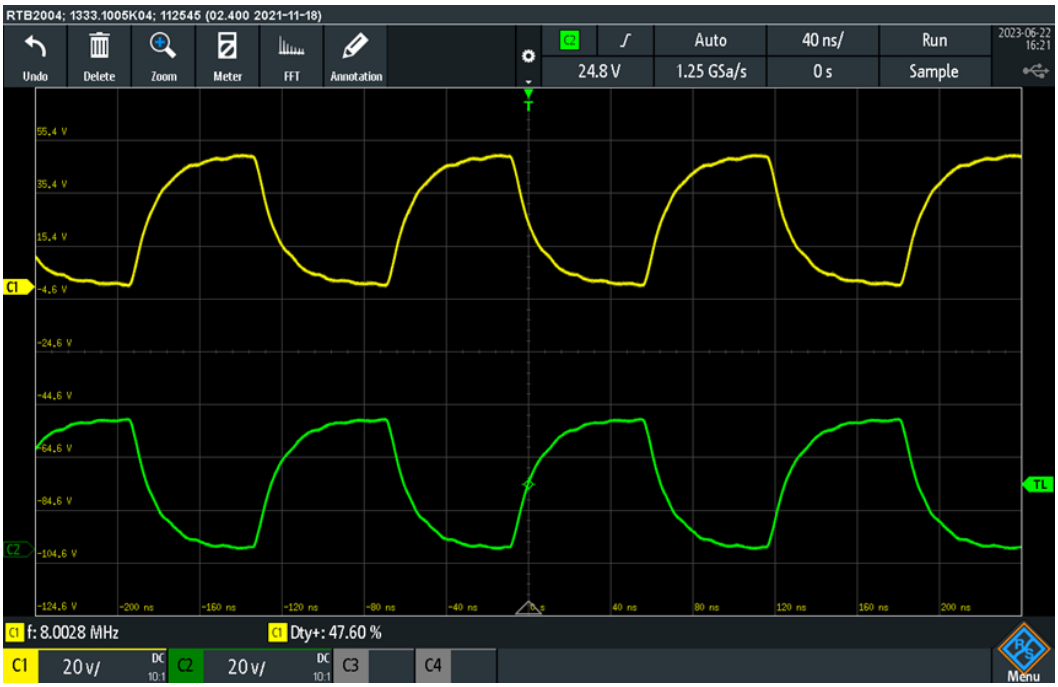

Figure 5.16: 8 MHz signal and 50% duty cycle

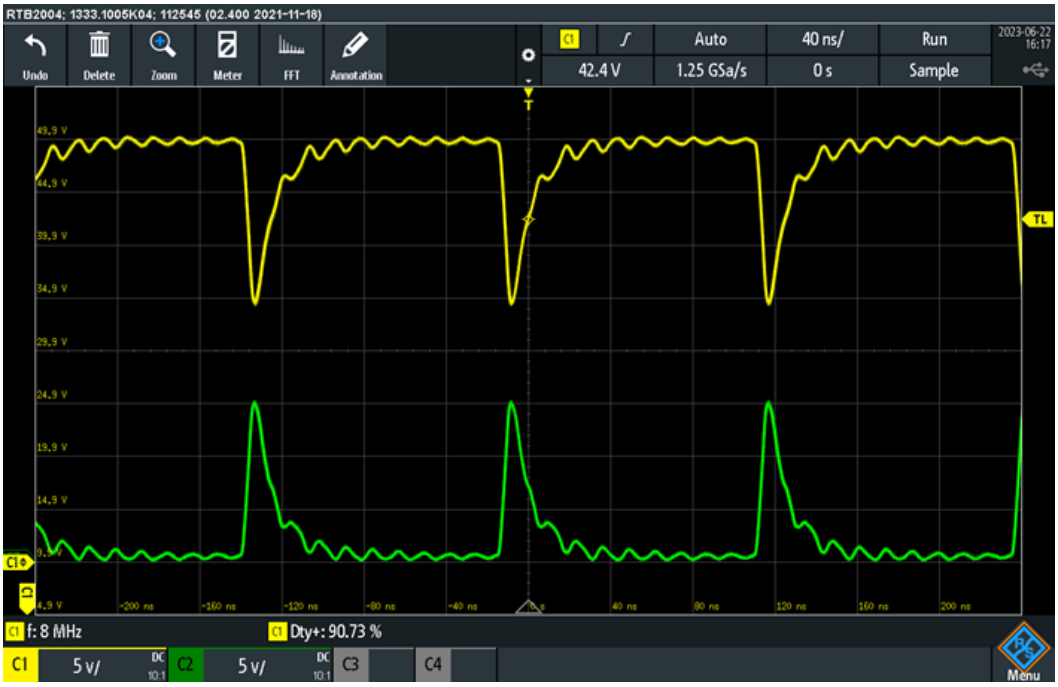

Figure 5.17: 8 MHz signal and 90% duty cycle

In figure 5.18 a signal of 10 MHz and 80% of duty cycle is shown: counter counts about 200/10 = 20 ticks and the signal can no longer be considered as a clear and precise PWM signal.

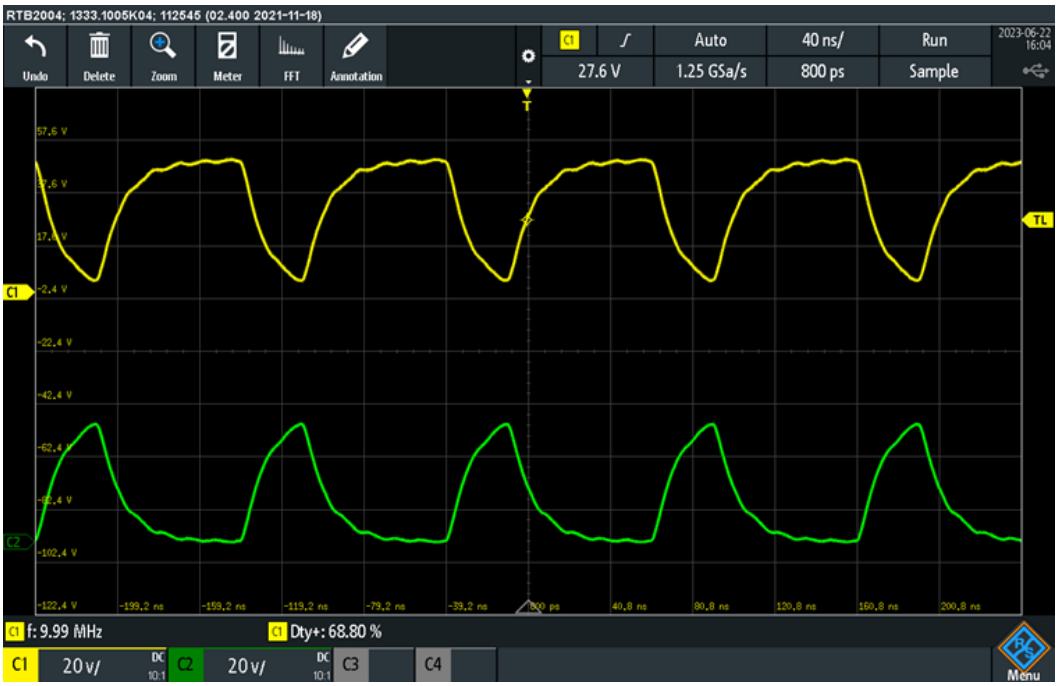

Figure 5.18: 10 MHz signal and 80% duty cycle

In figure 5.19 and 5.20 signals of 50 MHz and 50% of duty cycle are shown: counter counts about 200/50 = 4 ticks and definitively the signal can no longer be considered as a PWM signal that can manage a switch precisely, indeed also the duty cycle desired is not that achieved in the simulations.

Figure 5.19: 50 MHz signal and 50% duty cycle

Figure 5.20: Detail of 50 MHz signal and 50% duty cycle

Chapter 6

Conclusion and future works

Considering the overview of methods to generate signals and adding dead time and shift, some considerations could be made in order of complexity and precision of the generated signals.
In terms of creation of a single PWM, nothing changes between one method and another, but multiple creation of PWM methods differs in terms of synchronization. For multiple-channel generation, the synchronization of signals is not guaranteed for signals created in different clusters, unless they are all triggered by an external trigger. For the above reasons, in this study, all signals are created in a single cluster, that has a different number of channels for TOM or ATOM. In general, all the methods that involved a timer to synchronize realize very precise signals because it resets them at the same time, such as with the DTM use or library PWMhl, but it introduces the need of an additional channel for the timer.

Also the dead time task is crucial and so requires a study of the pros and cons of the various methods, while the shift task is more or less efficient in all developed applications. Focusing on dead time, the DTM usage is to avoid if it is possible because of the little shift that introduces in the triangula of modulation and so in the outputs that manage the switches of the converter. Also for dead time task, the application with a timer is preferable to others.

The usage of a up counter, instead of an up down counter, leads to a better manipulability of the signal and makes everything easier in terms of dead time insertion,

adding of shift ext.

From experimental result an observation can be made regarding the maximum limits of existence of the PWM signal and it is around 3-4 MHz to be clear and useful for a real switch, this is due to the fact that in any case a good resolution of the signal is necessary and it is not possible to have a too fast closing of the switch at too high frequencies. These tests are however of an experimental nature because for the purposes of a real application on a DC/DC converter it would not make sense to work at such high frequencies, but a good working frequency is around 100 kHz, as it could have problems of instability or lead to ripple unwanted in the various signals flowing in the converter.

In general each method may be fine depending on the application, for example, as the conclusion of these methods, to achieve an example of goal of 8 complementary signals the following considerations must be done about the hypothesis for PWM channels placement:

- The presence of 8 (4 signals + 4 inverse signals) channels for PWM is required;
- Atom cluster has 8 channels available (24 bit counter value);
- Tom cluster has 16 channels available (16 bit counter value).

Resume of available applications:

1. DTM module usage → discarded option for the delay introduced;
2. Atom/Tom PwmHl.h library (additional master-counter introduction) → 9 channels
3. Atom/Tom realization of dead time (without timer) → 8 channels.

Final considerations and choice:

- With the use of one ATOM cluster (8 PWM without a master up counter) → Application 3 applicable

- With the use of one TOM cluster (8 PWM without/with a master counter + additional channels) → Application 2 or 3 applicable

Considering the obtained results, TOM usage could be the best choice when it has the need for more than 8 channels, that are not available in an ATOM cluster.

Generation methods can definitely be improved: for example the use of a timer, as written before, is preferable but introduce the need of an additional channel, so in terms of use of channel and memory it would be better to explore a method that, with a master slave application, use the previous channel to trigger the following one. It might even be possible to realize the desired signals programming the MCS, the core of GTM, in assembly language, in order to realize at low level all the requirements about shift and dead time insertion.
Another future work that can be explored is about the use of CCU6, a capture compare unit that automatically generates 3 center aligned signals, that for some applications could be useful and more immediate, or the use of DTM to add shift or other submodules specialized with their own structure in automotive application such as that described in goals.
It could be possible also to link PWM signals to other modules, such as the ADC converters, in order to trigger them for the conversion: several use of signals are possible, it depends on the control algorithm requests.
In order to complete the presented work, it might be interesting to connect several boards at the system level; interesting in order to generate several PWM signals, synchronised for example by the clock of one of the boards or with an external clock, which are the key points that will be developed for the creation of the control algorithm of the electric vehicle in question.

Bibliography

[1] *AURIX Infineon SoC*. URL: https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc3xx/.

[2] Muhammad Kazim Hafeez Ehsan ul haq. "FPGA implementation of a low power processor independent and reusable System on Chip platform". In: *IEEE 2009 International Conference on Emerging Technologies*. IEEE. 2009, p. 337.

[3] *AURIX ™ TC3xx User Manual part1*. Infineon, 2021.

[4] *AURIX ™ TriBoard TC399*. URL: https://www.infineon.com/cms/de/product/microcontroller/32-bit-tricore-microcontroller/.

[5] N.A.Rahim Saad Mekhilef and A. M. Omar. "Microprocessor Implementation of Three Phase PWM Switching Strategies". In: *IEEE 1999 International Conference on Power Electronics and Drive Systems*. IEEE. 1999, pp. 161–163.

[6] Cheng-Shion Shieh. "Quick implementation of Pule Wise Modulation(PWM), Pulse Frequency Modulation(PFM) and mixed PWM/PFM on FPGA chip". In: *2020 International Symposium on Computer, Consumer and Control (IS3C)*. IEEE. 2020, pp. 444–447.

[7] Jin-Woo Ahn Dong-Hee Lee. "A Simple and Direct Dead-Time Effect Compensation Scheme in PWM-VSI". In: *IEEE TRANSACTIONS ON INDUSTRY APPLICATIONS, VOL. 50, NO. 5, SEPTEMBER/OCTOBER 2014*. IEEE. 2014, pp. 3017–3025.

[8] Xibo Yuan Qingzeng Yan. "A Double-Modulation-Wave PWM for Dead-Time-Effect Elimination and Synchronous Rectification in SiC-Device-Based High-Switching-Frequency Converters". In: *IEEE TRANSACTIONS ON POWER*

ELECTRONICS, VOL. 35, NO. 12, DECEMBER 2020. IEEE. 2020, pp. 13500–13512.

[9] *GTM cookbook*. Bosch, Rev. 0.5, 27/03/2014.

[10] *Infineon-TriBoardManual-TC3X9-UM-v02*$_0$1 − EN. Infineon, 2017.

[11] *AURIX ^{TM} TC3xx User Manual part2*. Infineon, 2021.

[12] Yuhong Fu Zheng Chen Ziling Nie. "A bidirectional power converter for battery of plug in hybrid electric vehicles". In: *IEEE 2010*. IEEE. 2010, pp. 3049–3053.