UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

DIPARTIMENTO DI INGEGNERIA ENZO FERRARI

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

____

PROVA FINALE

SMART PLUG-TOP DIGITAL

Hamza Ilyas

RELATORE

Prof. Nicola Bicocchi

ANNO ACCADEMICO 2021 – 2022

Introduzione

Questo lavoro di tesi è stato sviluppato presso la sede di Carpi della Federal Mogul Powertrain Italy. Il core business aziendale è la produzione di sistemi di accensione per motori a combustione interna, con accensione comandata.

La Federal Mogul è stata assorbita nel 2018 dalla azienda Tenneco, una multinazionale leader nel settore automotive, e con una solida reputazione nel campo dell'Ingegneria meccanica.

A partire dal 19 settembre del 2022, ho avuto l'opportunità di lavorare a stretto contatto con esperti della divisione "Powertrain", e seguire il processo che ha portato alla realizzazione del prodotto in questione, la Smart Plug-Top Digital (SPTD).

La SPTD è un dispositivo elettronico innovativo, progettato per offrire funzionalità aggiuntive, come la diagnostica, attualmente non presente, soprattutto in una versione digitale, per prodotti come candele d'accensione e bobine prodotti dentro l'azienda.

Questi componenti sono essenziali per l'accensione dei motori a combustione interna. Le candele permettono di generare una scintilla ad alta tensione che innesca la combustione della miscela carburante-aria all'interno della camera di combustione. Processo reso possibile dalla presenza di un arco elettrico altamente performante, posizionato tra gli elettrodi della candela.

In questo sistema, le bobine sono responsabili di generare la tensione richiesta dalle candele, a partire da una tensione continua nominale di 14 V generata dalla batteria del veicolo. Queste Bobine sono composte da un trasformatore con avvolgimento primario ed un secondario. Inoltre, è presente una semplice elettronica che lascia passare corrente sull'avvolgimento primario, caricando così il trasformatore, all'attivazione di un segnale di trigger. Più nel dettaglio possiamo dire che il segnale di trigger, accende un IGBT che permette il passaggio di corrente, questa corrente alimenta il campo magnetico all'interno del trasformatore. Ad un certo punto, quando la carica è sufficiente, il circuito viene aperto, la carica interrotta, e il campo magnetico accumulato andrà a scaricarsi sul circuito secondario, che è in grado di generare tensioni fino a 50 kV. L'alta tensione trasmessa alle candele tramite cavi appropriati genererà l'arco voltaico tra gli elettrodi della candela, fino a quando l'energia immagazzinata nel trasformatore non si esaurirà.

Nonostante questo sistema sia essenziale per il funzionamento del motore a combustione, questi componenti non presentano strumenti o tecnologie che ne permettano di rilevare malfunzionamenti.

Ciò è dovuto alle difficili condizioni di lavoro in cui operano, trovandosi all’ interno del motore, sono sottoposti a campi elettrici intensi, temperature elevate e vibrazioni continue.

Con il progredire della tecnologia e lo sviluppo dei microcontrollori, si è potuto realizzare un'unità di elaborazione digitale, in grado di lavorare in queste condizioni.

La SPTD si inserisce in questo contesto come una soluzione intelligente che permette il monitoraggio e una diagnostica sofisticata dei comuni guasti presenti su questi prodotti, oltre alla segnalazione di un fallimento nell’accensione della miscela.

Le SPT nascono dalla crescente domanda, nell’ ultimo decennio, di aumentare l’affidabilità dei veicoli, automobili, mezzi pesanti e motori per la generazione di energia.

Questo prodotto è realizzato in esclusiva da Tenneco, e attualmente non sono presenti soluzioni simili fornite dalla concorrenza.

Si parla di SPTD in quanto, nonostante richieda una stretta integrazione con l’elettronica del veicolo, includendo connessioni fisiche e protocolli di comunicazioni standardizzati, risulta di applicazione semplice.

Necessita infatti di un solo cavo per la comunicazione con il veicolo, che utilizza come protocollo di comunicazione, lo standard LIN, permettendone così un’applicazione rapida e semplice, sostituibile direttamente con le plug top standard presenti sul mercato.

Questo lavoro di tesi analizzerà l’implementazione logica del sistema, e la sua simulazione comportamentale. Tuttavia, essendo un prototipo in attesa di brevetto, l’elaborato avrà un tono prettamente teorico, dove verranno analizzati e descritti i principali moduli impiegati.

Spostando così il focus di questa tesi verso un’analisi schematica dei moduli impiegati, delle loro funzionalità, con alcuni esempi pratici per semplificarne la comprensione, oltre a una breve introduzione all’algoritmo, senza scendere nei dettagli specifici del prototipo in oggetto.

L’elettronica è controllata da un microcontrollore a 16 bit, realizzato dalla Microchip. Nota multinazionale americana leader nel settore. Offrono una maggiore potenza di calcolo rispetto agli omologhi a 8 bit. La loro architettura RISC, li rende in grado di eseguire istruzioni più specifiche in tempi molto brevi, dell’ordine di decine di nano secondi. Sono inoltre dotate di un alto numero di periferiche input/output.

Tra le altre feature, spicca una buona capacità di memoria flash (non-volatile), permettendo così lo sviluppo di codice più complesso e algoritmi sofisticati, senza dover limitare le dimensioni del codice.

Poi, la possibilità di gestire un alto numero di interrupt, con l'ausilio di una struttura denominata Interrupt Vector Table, ha svolto un ruolo cruciale nella scelta.

La Vector Table è una tabella a puntatori a funzioni, il cui fine è la gestione efficiente dei vari flag associati a ogni registro. A ogni entry della tabella si può associare un particolare interrupt, questo a sua volta conterrà l'indirizzo della funzione che deve essere eseguita al verificarsi di tale interrupt.

Nel caso dei microcontrollori a 8 bit, la gestione degli interrupt avviene mediante gli interrupt vectors predefiniti, dove ogni vettore rappresenta un interrupt specifico. Qui l'indirizzo della funzione che deve essere eseguita, è associata direttamente al vettore. Questo approccio risulta meno flessibile e limitato nella gestione degli interrupt, che invece sono di grande rilievo per il nostro scopo.

Tutto questo portò alla scelta di un MCU a 16 bit, invece di uno a 8 bit, come avrebbe invece previsto il progetto iniziale.

Il costo contenuto e molto competitivo è stato un altro motivo di decisione.

Dopo un'attenta valutazione e un approfondito confronto con tecnici e ingegneri specializzati della Microchip, la scelta di utilizzare un dsPIC33CDVL64MC106 si è rivelata la soluzione ottimale per le esigenze del progetto.

Questo microcontrollore offre una serie di funzionalità peculiari, tra cui la presenza del protocollo di comunicazione della LIN (Local Interconnect Network), usato in ambito automotive. Reso possibile per via dell'integrazione, nel processore, del LIN Transceiver ATA663211. Quest'ultimo è infatti il physical layer del protocollo LIN, che consente quindi la trasmissione e la ricezione dei dati sulla rete.

Un dispositivo che garantisce oltre a una buona immunità al rumore e interferenze, la protezione da sovratensione. Per maggiori dettagli sul dispositivo in questione è consigliata la consultazione del relativo datasheet.

La LIN è un protocollo di comunicazione seriale, a bassa velocità, che si appoggia sulla UART (Universal Asynchronous Reciever & Transmitter) per la conversione di flussi di bit di dati da un formato parallelo a un formato seriale.

Questa comunicazione mediante LIN consente alla SPT di interagire con altri dispositivi collegati all' interno dello stesso nodo. Nel corso della tesi, verranno analizzati i singoli pacchetti della LIN, e come questi vengono poi trasmessi e ricevuti tra i vari dispositivi presenti all' interno della stessa rete.

Le altre periferiche significative che verranno analizzate sono, i timer, l'interrupt Handler, che serve a gestire i vari interrupt, e il modulo SCCP (Source Capture Compare Pulse Width Modulation), nonché i moduli DAC (Digital to Analog Conversion) e ADC (Analog to Digital Conversion).

I timer a 16 bit sono una risorsa preziosa sia per la sincronizzazione delle operazioni che per la gestione del tempo. Permettono di programmare e controllare con precisione gli intervalli temporali e l'esecuzione delle funzioni.

Successivamente abbiamo la possibilità di associare eventi di trigger ad altri moduli periferici del microcontrollore. Questa funzionalità favorisce la sincronizzazione delle attività dei vari moduli precedentemente citati, oltre a creare un ambiente di cooperazione che ci permette di raggiungere l'obbiettivo prestabilito.

I Moduli SCCP offrono una vasta gamma di funzioni avanzate, come la generazione di segnali di PWM ad alta precisione, cattura di eventi esterni e il confronto dei segnali ricevuti in input con valori di riferimento.

Infine, i moduli ADC e DAC utilizzati (analog to digital converter e digital to analog converter), hanno interfacciato i segnali di tipo analog in input e output, necessari per il controllo della bobina.

L' obbiettivo principale del microcontrollore è quello di calcolare dati di diagnostica, dati che racchiudono informazioni relative alle condizioni di lavoro e di conseguenza legati allo stato di usura e alle prestazioni di candele e bobine.

Questi dati devono poi essere elaborati per verificare il grado di accuratezza che rappresentano in termini di tempo, tenendo in considerazione i vari ritardi dovuti alla trasmissione, limitazioni hardware, e ai disturbi, per poi essere trasmessi attraverso il BUS della LIN. Dati che vengono ricavati con opportuni algoritmi, il cui funzionamento può essere descritto da una semplice FSM (Finite state machine), un automa a stati finiti.

Questo elaborato, dopo aver affrontato i tratti teorici dei due macro-argomenti sopra citati, MCU e LIN, conclude con una breve spiegazione teorica del codice e del suo funzionamento principale.

Index

1. Introduction

This project took place at the research and development office of Tenneco Inc., a multinational specialized in the automotive field. The Powertrain division of Carpi is concerned with the design and manufacturing of ignition systems for internal combustion engines, specifically those featuring controlled ignition.

1.1 Application Field

The resolute team of engineers, whom I had the pleasure of working with, are fully committed to the development of Spark Plug Modules as a breakthrough in ignition system technologies. These modules, along with other interconnected components such as ignition coils, play a crucial role in the combustion process that leads to the ignition of the air-fuel mixture in the engine's cylinders.

Within this system, ignition coils generate the necessary voltage required by the spark plug module upon receiving a 14 Volt nominal input from the vehicle battery. Such circuitry is composed of a transformer, that charges through the flow of current in the primary coil, generating a magnetic field, when a precise trigger is received. The discharge of this transformer, when a certain threshold is reached, onto the secondary coil, induces up to 50 kV of tension that must be transferred to the spark plugs.

These ignition spark plug modules are responsible for generating the electrical spark that starts the process of combustion. A spark created by the arc between the two electrodes of this module, fueled by the energy accumulated in the transformer. Such discharge persists until it dissipates all the energy.

1.2 Necessity of the product

Due to the position of these ignition components, and the harsh conditions they work under, that vary from elevated temperatures to relentless vibrations, it has always been challenging to integrate a device that could process diagnostical information about their condition and state. Traditional diagnostic methods may not provide comprehensive insights into the health and performance of these critical components.

The development of the Smart Plug-Top Digital (SPTD) aims to address this limitation by fostering enhanced reliability and efficiency of ignition systems.

1.3 Purpose of the SPTD

By enabling accurate diagnostic capabilities and real-time monitoring, the Smart Plug-Top Digital empowers automotive technicians and engineers to identify potential issues and restore optimal performance of the engine, promoting longevity of the ignition system components, as well as the combustion process itself.

The integration of this device has another goal, as the name suggests, being a smart plug-top it is of simple appliance and could promptly substitute the standard plug-tops currently available in the market.

1.4 Scope

This thesis follows the process that led to the implementation of the SPTD.

The final goal is to have a working prototype to bring to market.

Unfortunately, due to the nature of the product, and the proprietary information of Tenneco Inc., this project will not explore the specific code that was implemented.

Instead, it will go through the process that led to the achievement of such firmware,
written in c language, using MPLAB X IDE.

The first part of the document intends to explore the microcontroller environment through a brief description of its architecture and the modules it is composed of. This is followed by a comparison with the world of microprocessors, so that the reader has the theoretical knowledge to comprehend its structure and usage, enabling an analytical description of the chosen MCU (Microcontroller Unit), the dsPIC33CDVLMC106.

Then we have the description of this 16-bit microcontroller, emphasizing particularly on the implemented modules, such as the timers, the interrupt handler, the SCCP modules as well as the ADC and DAC modules.

The theoretical part concludes with the introduction to the LIN protocol, a key concept for our product, which allows interconnectivity among various instruments present in the vehicle.

Goal achieved through a physical LIN Bus that connects these components to the ECU (Engine Control Unit), enabling the transmission of data and commands.

The last chapter will go through the organization of the code, with some examples on how to use the previously mentioned modules, and the description of the FSM (Finite State Machine) that runs the device.

2. Microcontrollers

A microcontroller is an electronic device, integrated on a single circuit, as shown in figure 1.1, that can execute a specific set of instructions enabling the interpretation and processing of data.

Fig. 1.1: A Microcontroller Unit.

The core features offered by such processing units can be identified as:

- Integrated memory (including both data memory and program memory).
- I/O capabilities, handling both analog and digital signals.
- Built-in peripherals (Timers, ADC & DAC modules …).

2.1 History

One of the first microcontrollers was the Intel 8048, developed in the early 1970’s after the release of the first microprocessors.

While the microprocessors were taking a leap towards general purpose applications the microcontrollers were going in the opposite direction, integrating a self-contained system with a processor, memory, and configurable pins. Pins that can serve both as input and general-purpose output, through an appropriate configuration, depending on the specific requirements of the application.

After this release, many other companies pursued similar goals, developing different microcontrollers, with different module integrations and new features aimed at improving the architecture and the overall speed of microcontrollers.

However, the Intel architecture remained the most prominent, as most of the widely spread microcontrollers nowadays still follow this architecture.

2.2 Architecture

Microcontroller units are mostly designed based on the Harvard architecture, featuring a physical separation between instruction and data memories.
This implicates the presence of two separate buses to access these different memories, as we can observe in the right side of Figure 1.2.

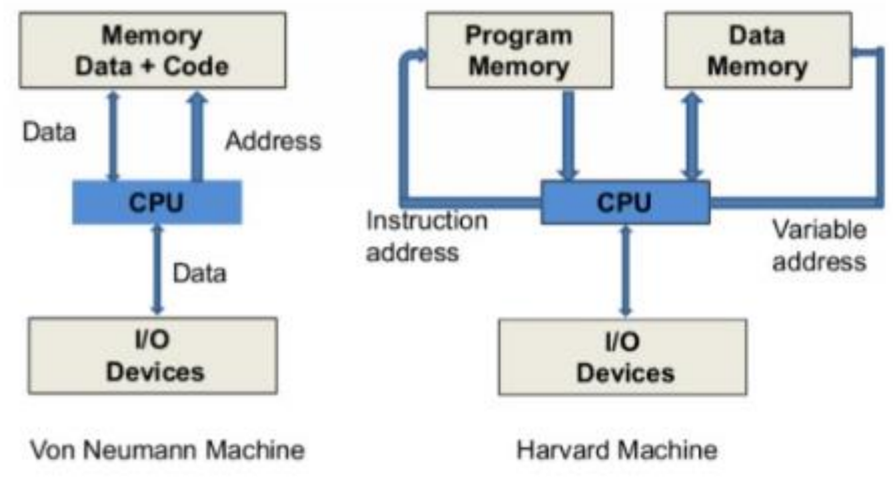


Fig. 1.2: Comparison between Harvard Architecture and Von Neuman

An approach that allows parallel execution of instructions and simultaneous data retrieval.
Such layout is not present in the Von Neuman architecture (shown in the left side of figure 2.5) used by the microprocessors, as the BUS that connects to the instruction and data memory is only one.
A separation that leads to higher execution speed int the MCU.

More in detail, Harvard architecture has also different memory types. RAM for the data, as it requires bidirectional data access, while for the code a ROM is sufficient.
Another key factor that leads to higher response speed is the shorter length of the pipelines that connects the CPU to the memories.
All these qualities ensure that such architecture is well-suited for real time systems, requiring predictable and deterministic execution, while at the same time minimizing delays caused by data memory access conflicts.
However, these benefits come at the expense of increased complexity, along side a relatively higher cost.

2.3 Microcontrollers vs Microprocessors

To provide a comprehensive understanding of microcontrollers, it is essential to analyze and compare them with microprocessors. This comparison is particularly valuable as Microcontrollers are derived from the development of microprocessors.

Analyzing these devices, we can identify a few key distinctions in:

- Purpose: microprocessors are primarily used in general-purpose computational devices such as personal computers and server infrastructures, allowing them to execute a considerable number of complicated tasks. Meanwhile, microcontrollers are used to manage devices with much simpler but specific tasks.
- Size: microcontrollers are usually bigger, in size, than microprocessors, as they incorporate all the features in a single chip or a small board. Microprocessors on the other hand, depend heavily on the external resources they need to integrate.
- Power consumption: microcontrollers are optimized for low power consumption whereas microprocessors prioritize performance.
- Instruction set: Microcontrollers typically have a reduced instruction set, RISC, compared to the microprocessors CISC architecture.

Nonetheless microcontrollers are faster in the execution of their instructions. Reasons for such behavior can be found in their simplicity and small size that integrates all the required modules very closely.

Even though there are many fundamental differences, some similarities can also be found among these computational devices as suggested by figure 1.3:

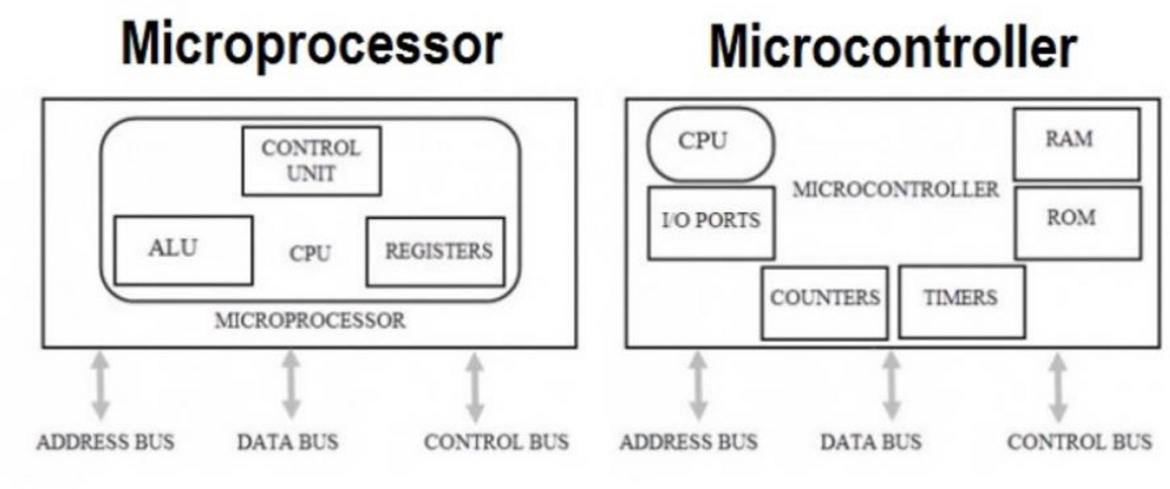


Fig 1.3: Comparison of Microprocessors with microcontrollers

- CPU (Central Processing Unit): both devices have a CPU serving as the core component for executing instructions and performing computations.
- Program Execution: MCU and MPU follow a fetch-decode-execute cycle.
- Development tools: both microcontrollers and microprocessors rely on similar development tools, such as integrated development environments (IDEs) and compilers, to write, debug and test software.

2.4 MCU Classification

The classification of microcontrollers is primarily based on the width of their data bus, which determines the size of data they can process at a time.

This classification has the goal of demonstrating the peculiarities of such division and what it entails in terms of resources for each class, regarding the specific microcontrollers sold by Microchip.

It provides a clear understanding of the distinct features offered by each category, facilitating the selection of the most suitable microcontroller for this project.

These microcontrollers can be classified in the following five categories:

- Baseline:

  Baseline PIC microcontrollers feature 12-bit instruction word and provide a balanced set of features at an economical cost. With a simple architecture, they are the easiest to work with and comprehend among the 8-bit family. Key features include a straightforward 33-instruction set, 2 K word addressable program memory, 144 bytes of RAM, a 2-level hardware stack, and a single 8-bit file select register.

- Mid-Range:

  Mid-Range PIC microcontrollers offer enhanced performance compared to Baseline models while incorporating their features. These devices employ a 14-bit instruction word and provide rich peripheral integration, making them suitable for applications requiring advanced embedded control and greater memory capacity. They offer thirty-five easy-to-learn instructions, 8 K word addressable program memory, 368 bytes of RAM, an 8-level hardware stack, a 9-bit file select register, hardware interrupt handling, and a highly integrated feature set including EEPROM, LCD, mTouch™ sensing solutions, and serial communications.

- Enhanced Mid-Range:

  The enhanced mid-range core represents the latest addition to the PIC microcontroller family, it combines the best elements of the mid-range core with enhanced performance and reduced power consumption. These devices feature an expanded program memory and higher operating speeds. They provide forty-nine assembly commands, 32 K word addressable program memory, 4 KB of RAM, a 16-level hardware stack, two 16-bit file select registers, hardware interrupt handling with content saving, and an advanced feature set including multiple serial communications and motor control capabilities.

- High-End:

  The High-End category, identified by the PIC18 prefix, offers the highest level of performance and integration within the 8-bit architecture. These microcontrollers feature up to 16 MIPS of processing power and advanced peripherals such as CAN, USB, Ethernet, LCD, and CTMU. With the largest pin count and memory size among 8-bit parts, they are optimized for C programming. Key features include eighty-three assembly instructions, up to 2 MB addressable program memory, 4 KB of RAM, a 32-level hardware stack, an 8-bit file select register, integrated 8x8 hardware multiply, and the highest performance within the 8-bit architecture.

- dsPIC family:

  These microcontrollers were designed for advanced digital signal processing (DSP) applications.

  They offer high-performance processing capabilities, making them suitable for applications requiring precise control and signal management.

  The dsPIC33C family features:

  - ❖ 16-bit wide instruction set with extensive DSP instructions for efficient signal processing.
  - ❖ Up to 512 KB addressable program memory.
  - ❖ Up to a maximum of 48 KB RAM for data manipulation.
  - ❖ 32-level hardware stack.
  - ❖ Two (16-bit) Timers
  - ❖ Advanced peripherals including CAN, USB, LIN, ADC, PWM, Ethernet.
  - ❖ Integrated 16x16 hardware multiply
  - ❖ Optimized architecture for C programming and DSP algorithms.

3. dsPIC33CDVL64MC106

For this project, the dsPIC33CDVLMC106 was chosen. A 16-bit microcontroller with digital signal controller capabilities.

The core of this dsPIC is a 100 MIPS high-performance dsPIC33CK64MC105 device with peripherals, such as high-speed ADCs, op amps and analog comparator, Pulse-Width Modulators (PWMs) with built in Fault protection, triggering and synchronization features. The layout of this MCU can be seen in figure 2.1.

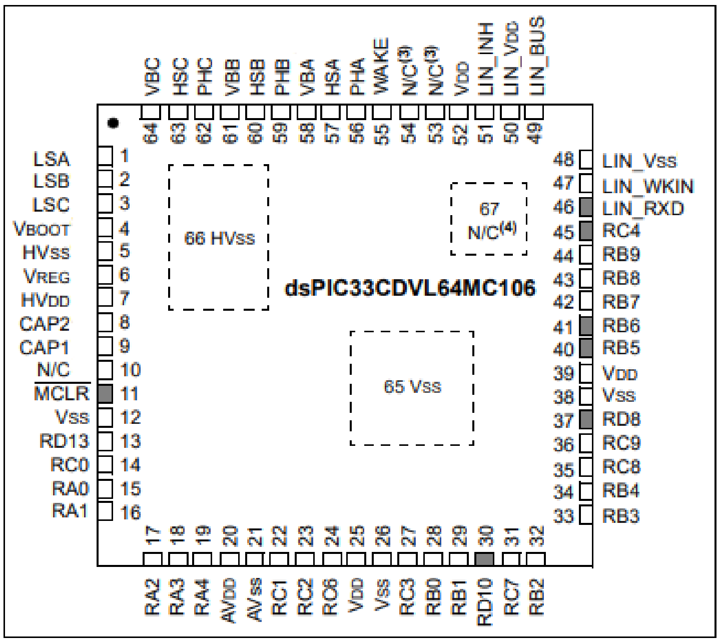


Fig. 2.1: Pin diagram of dsPIC33CDVL64MC106 [3]

3.1 Micro choice

While our goal could have easily been reached through the usage of an 8-bit microcontroller, as the initial project expected, ultimately, the 16-bit dsPIC33CDVL was deployed instead. This choice had the goal of leaving space for future features to be implemented and incorporated in the SPTD.

The reasons behind this selection are many, including:

- The integration of the LIN transceiver ATA663211, in the micro, eliminating the necessity to incorporate, through appropriate circuitry, the physical layer of the LIN network.

- The Vector Table of 16-bit MCUs serves as a pointer-based function table, which offers extreme versatility in the handling of the interrupts. Each entry can be associated with any interrupt, containing the address of the function to be executed when the interrupt occurs.
- The greater FLASH memory, enabling the development of more complex algorithms.

These unique features offered by the micro, alongside the integration of peripherals, like the SCCP module, timers and DAC and ADC modules, lead to its preference.

Furthermore, the cost-effectiveness and high competitiveness of such device played a crucial role in this decision-making process.

3.2 Introduction to Micro

Microchip products can be easily recognized through their naming systems which provides insight into the specific features offered by each product.

In the case of the dsPIC33CDVL64MC106, for example, we can observe that by breaking down the name in parts, we can identify the specific features that it has to offer:

- dsPIC: this implies the Microchip trademark.
- 33C: this is the architecture of the dsPIC, a 16-bit digital signal controller.
- D: stands for the MOSFET Gate Driver.
- V: Voltage regulator.
- L: LIN Transceiver.
- 64: number that indicates the program memory size in Kbytes.
- MC: This refers to the product group.
- 106: this number references the Pin count, meaning 64-pins.

While this microcontroller boasts a wide range of features, it is important to note that not all of them have been implemented in this project.

The comparator module, along with Digital-to-Analog Converter (DAC) can be used for detecting overcurrent faults to protect the micro.

The dsPIC DSC has three operational amplifiers; these can be configured by connecting external input and feedback resistors for amplifying currents sensed by the shunt resistor, a low-value resistor used to measure and monitor the electrical current flow through the circuit it is placed in.

Then there are communication peripherals such as SPI, I2C and UART, for communicating with the host PC, central controller, or main controller.

This dsPIC also presents highly sophisticated sensors, like the Hall Sensor and the Quadrature Encoder, however it is important to note that the utilization of these modules does not fall under the scope of this thesis project.

3.3 Demo Board

The dsPIC33CDVL64MC106 microcontroller is accompanied by a versatile demo board that serves as an essential tool for its development. This board showcases the microcontroller's capabilities and allows for firsthand exploration and testing of its features. Equipped with various connectors and interfaces, as shown in figure 3.2 and explained in table 3.3, the demo board can seamlessly integrate with external devices, enabling comprehensive evaluation and validation of the microcontroller's performance.

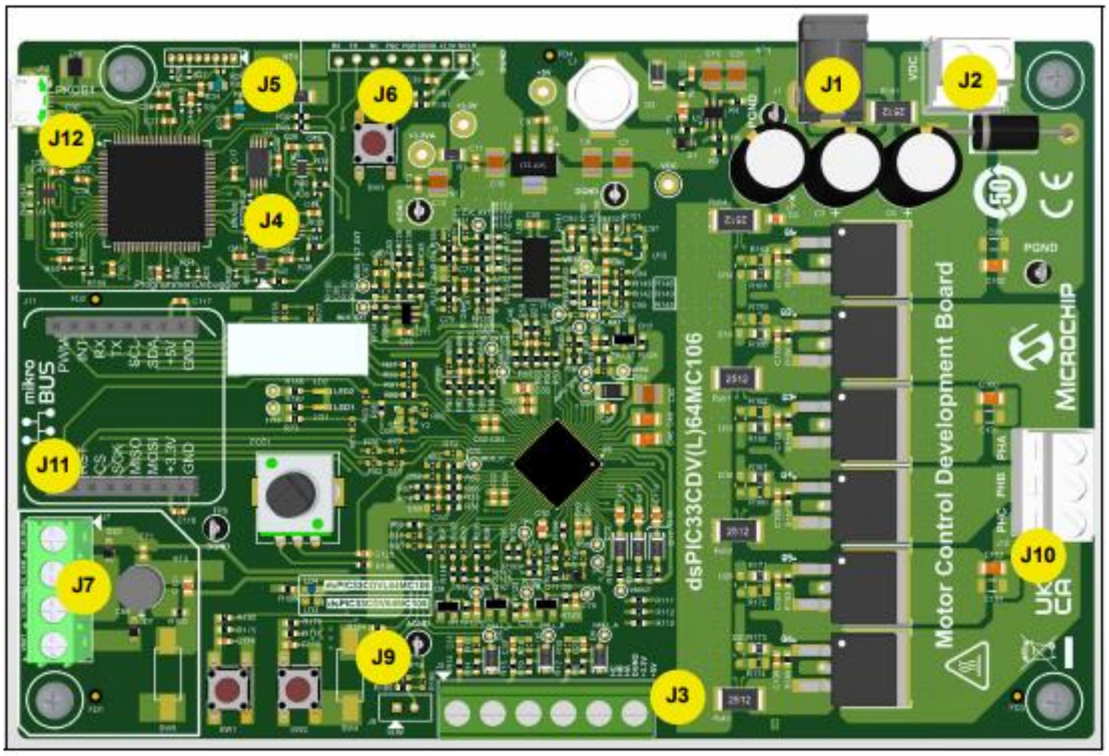
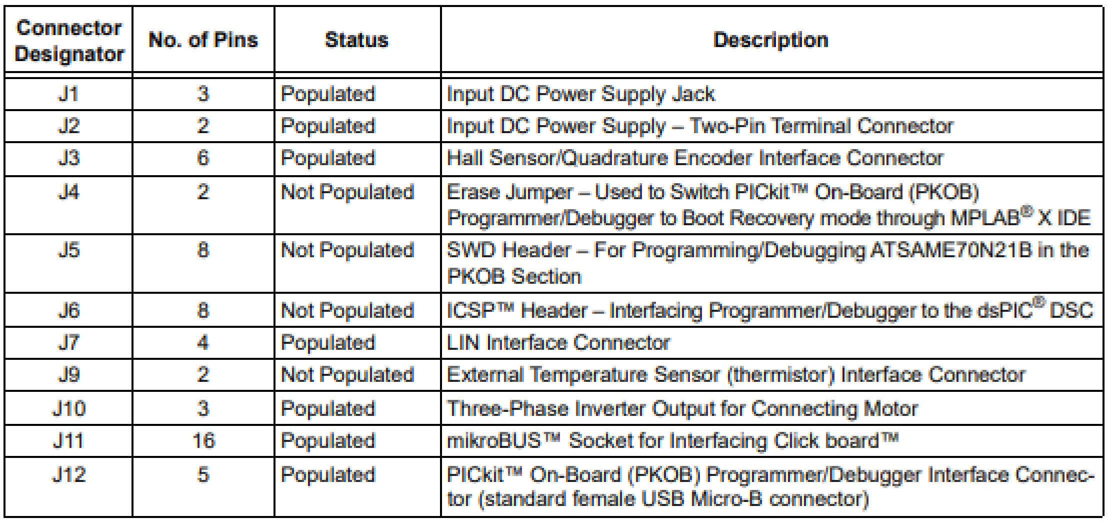


Fig. 2.2: Top side model of the demo board highlighting connector interfaces [4]



Connector Designator	No. of Pins	Status	Description
J1	3	Populated	Input DC Power Supply Jack
J2	2	Populated	Input DC Power Supply – Two-Pin Terminal Connector
J3	6	Populated	Hall Sensor/Quadrature Encoder Interface Connector
J4	2	Not Populated	Erase Jumper – Used to Switch PICkit™ On-Board (PKOB) Programmer/Debugger to Boot Recovery mode through MPLAB® X IDE
J5	8	Not Populated	SWD Header – For Programming/Debugging ATSAME70N21B in the PKOB Section
J6	8	Not Populated	ICSP™ Header – Interfacing Programmer/Debugger to the dsPIC® DSC
J7	4	Populated	LIN Interface Connector
J9	2	Not Populated	External Temperature Sensor (thermistor) Interface Connector
J10	3	Populated	Three-Phase Inverter Output for Connecting Motor
J11	16	Populated	mikroBUS™ Socket for Interfacing Click board™
J12	5	Populated	PICkit™ On-Board (PKOB) Programmer/Debugger Interface Connector (standard female USB Micro-B connector)

Tab. 1: Table describing the connector interfaces of Fig. 3.1 [4]

The demo board provides a user-friendly environment for rapid prototyping, allowing developers to bring their ideas to life quickly and evaluate the microcontroller's functionality in real-world scenario. It includes onboard peripherals such as buttons, LEDs, and a potentiometer, which can be easily accessed and controlled, though the configuration of the general purpose I/O pins associated with them, facilitating interactive experimentation and testing.

The J7 connector, from figure 3.2 and table 3.3 serves as the LIN (Local Interconnect Network) interface connector. This connector provides the necessary connections for LIN communication, including the LIN bus and three additional inputs. These inputs allow for the integration of LIN transceivers and enable communication between various electronic components within a vehicle. The detailed configuration and usage of the J7 connector, along with LIN communication, will be explored in-depth in the upcoming chapter regarding LIN.

With this demo board, developers can explore the full potential of the dsPIC33CDVL64MC106 DSC and push the boundaries of innovation. Whether it is prototyping a new product, like in this case, developing a proof-of-concept, or conducting experiments and research, the demo board serves as an invaluable platform for turning ideas into reality. Its versatility, combined with the power and capabilities of the digital signal controller, enables the creation of sophisticated applications across various industries.

3.4 Development environment

One of the key components of the development environment is MPLAB X, a powerful and widely adopted integrated development environment (IDE) also developed by Microchip. MPLAB X offers advanced features and tools that facilitate efficient code development, debugging, and simulation. It provides a seamless development experience with its intuitive user interface and support for various microcontrollers, including the dsPIC33CDVL64MC106.

In addition, Microchip Code Configurator (MCC) is another powerful tool within MPLAB X. MCC allows for the automatic generation of code based on the selected peripherals and configurations. While the generated code can be further customized and modified, MCC significantly simplifies the initial setup and configuration process, saving valuable development time and effort. This tool adds an extra layer of efficiency to the development workflow.

3.5 Programming and debugging

The J12 connector, from figure 3.2 and table 3.3, serves as the PICKit on-board (PKOB) programmer and debugger interface connector. This interface facilitates the programming and debugging of the dsPIC33CDVL64MC106 microcontroller directly on the demo board.

This is possible through the usage of the PGC3 and PGD3 pins.

These pins, after appropriate configuration, allow the communication of such interface to the DSC.

It offers a convenient means of uploading firmware and debugging code during the process of development, as it was designed to be integrated with MPLAB X.

Its integration allows for agile prototyping and rapid iteration through code.

4. Peripheral I/O Modules

This chapter will provide detailed insight into the integrated peripherals of the microcontroller that allowed us to achieve our goal. This alongside explaining their basic operation modes, and benefits.

4.1 Timer

The timer modules are an essential peripheral for our application, which enable precise timing and event generation capabilities. It consists of multiple timer modules, each with its own set of features and configuration options. These timers can be utilized for a variety of tasks, such as measuring time intervals, generating accurate delays, or synchronizing events within a system.

The timer module offers different modes of operation, including simple timer mode, gated timer mode, synchronous counter mode and Asynchronous counter mode.

- In simple timer mode, the timer counts up or down based on a clock source and can generate an interrupt or trigger an event when a specific count value is reached.
- In Gated timer mode, the counter is based on an external gate signal. This signal acts as a control signal, allowing increment or decrement only when the gate signal is active.
- In synchronous counter mode the timer functions as a counter that increments and decrements based on an external clock signal. Most used for event counting or synchronization with external signals.
- In Asynchronous counter mode, the timer operates as a counter but uses an external asynchronous clock signal as the clock source. This allows the timer to count at different frequency from the system clock, providing flexibility in counting events at a non-uniform rate.

To configure and control the timer module, various registers and control bits are available, more precisely the ones in figure 3.1, regarding the timer1 T1CON register.

R/W-0	U-0	R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0
TON[(1)]	—	SIDL	TMWDIS	TMWIP	PRWIP	TECS1	TECS0
bit 15							bit 8

R/W-0	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	U-0
TGATE	—	TCKPS1	TCKPS0	—	TSYNC[(1)]	TCS[(1)]	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Fig. 3.1: The registers associated with Timer1, T1CON [3]

The different configuration of the registers allows customized behavior according to specific requirements. These registers control parameters such as timer source selection, prescaler settings, interrupt enable/disable, and trigger conditions as shown in figure 3.2.

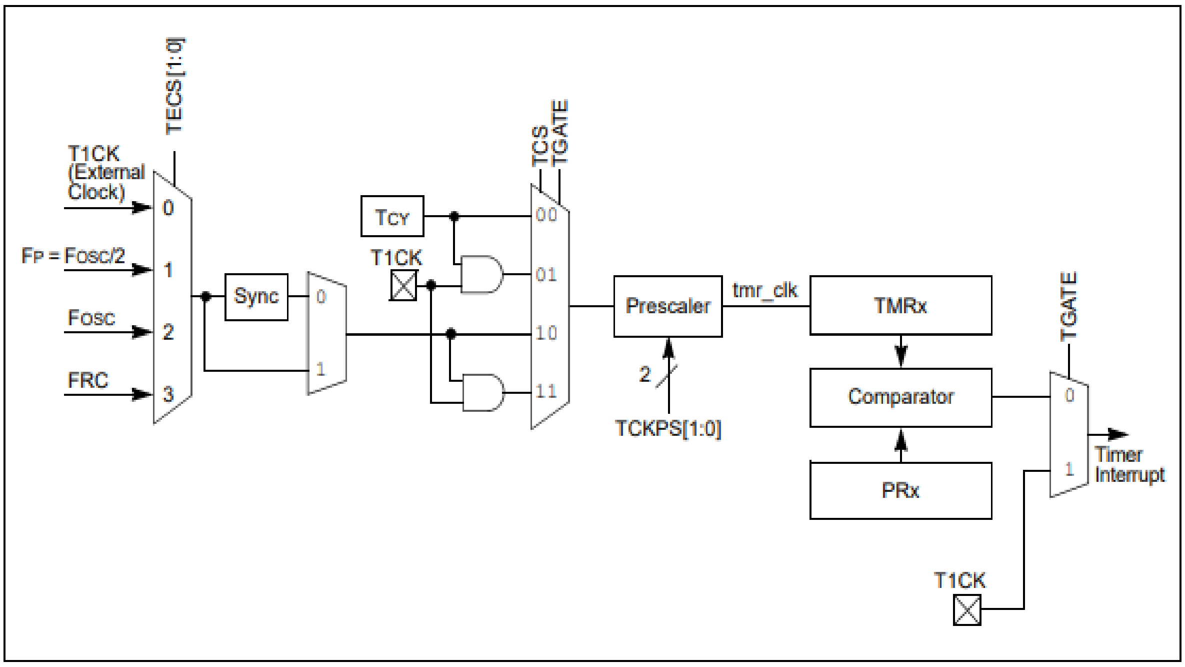


Fig. 3.2: Block Diagram of the Timer module [3]

The timer module plays a crucial role in time-sensitive applications, such as real-time systems, event-driven systems, and motor control applications, where accurate timing and synchronization are vital for proper operation.

4.2 Interrupt Vector Table

The Interrupt Vector table (IVT) is a data structure within the dsPIC33CDVL64MC106 microcontroller that manages and organizes the interrupt handling mechanism. It consists of a list of memory locations, known as interrupt vector addresses, where the starting addresses of individual interrupt service routines (ISRs) are stored.

When an interrupt event occurs, the microcontroller consults the interrupt table to determine the corresponding ISR address and jumps to that address to execute the associated interrupt service routine. This allows the microcontroller to quickly respond to and handle various interrupt events, such as timer overflow, external interrupts, or peripheral-specific interrupts.

The interrupt vector table is usually situated at a predetermined address in the program memory, with each entry in the table corresponding to a particular interrupt source. By modifying the contents of the interrupt table, developers can define the desired outcome and priority of each interrupt source. This IVT can be seen in Fig.3.3 alongside the six non-maskable vectors and the 246 sources of interrupts. Each of these interrupts has its own vector containing a 24-bit wide address. The value that has been programmed in each interrupt vector, will be the starting address of the associated ISR.

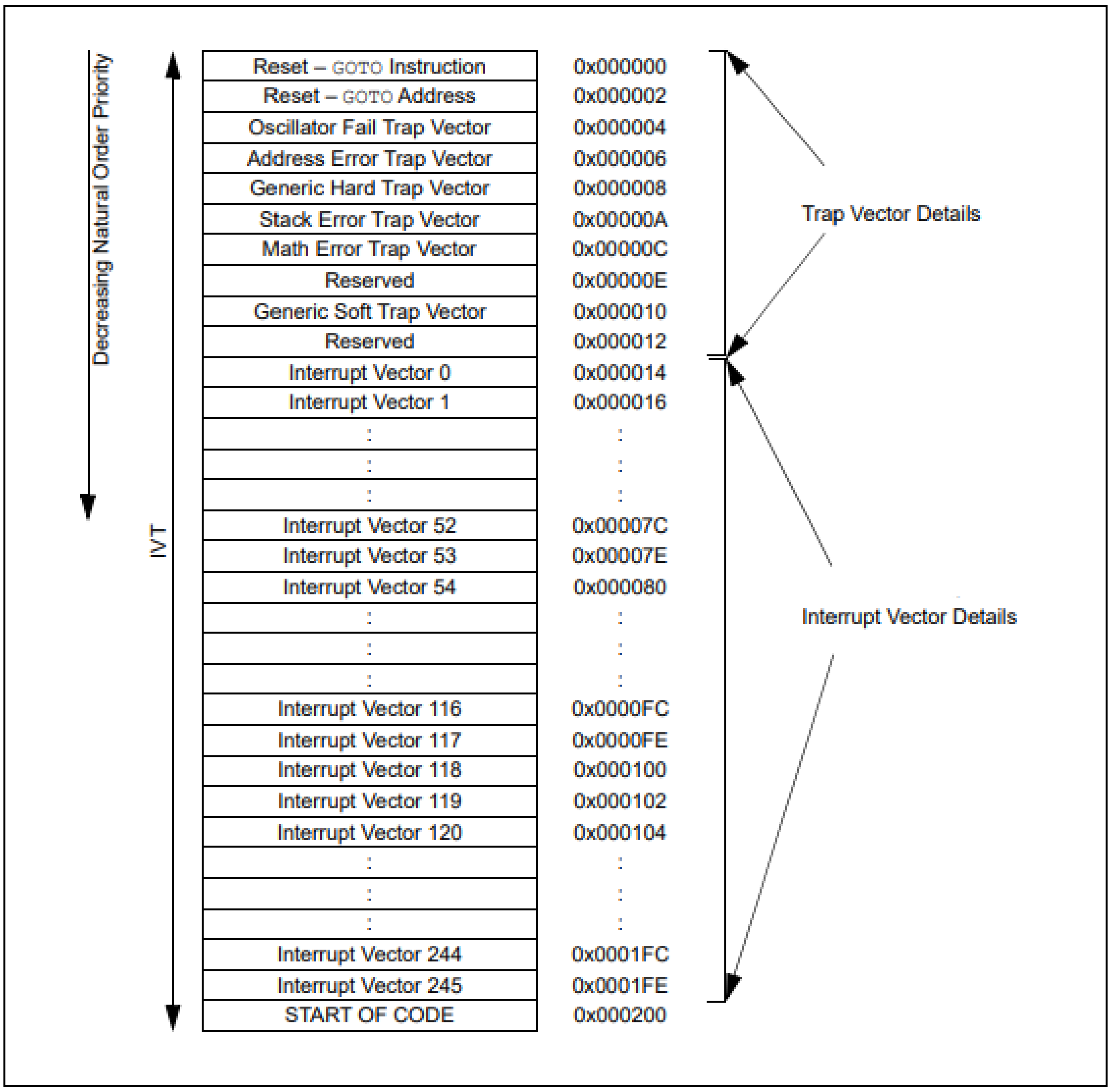

Decreasing Natural Order Priority
IVT
Reset – GOTO Instruction 0x000000
Reset – GOTO Address 0x000002
Oscillator Fail Trap Vector 0x000004
Address Error Trap Vector 0x000006
Generic Hard Trap Vector 0x000008
Stack Error Trap Vector 0x00000A
Math Error Trap Vector 0x00000C
Reserved 0x00000E
Generic Soft Trap Vector 0x000010
Reserved 0x000012
Interrupt Vector 0 0x000014
Interrupt Vector 1 0x000016
Interrupt Vector 52 0x00007C
Interrupt Vector 53 0x00007E
Interrupt Vector 54 0x000080
Interrupt Vector 116 0x0000FC
Interrupt Vector 117 0x0000FE
Interrupt Vector 118 0x000100
Interrupt Vector 119 0x000102
Interrupt Vector 120 0x000104
Interrupt Vector 244 0x0001FC
Interrupt Vector 245 0x0001FE
START OF CODE 0x000200
Trap Vector Details
Interrupt Vector Details

Fig. 3.3: Interrupt Vector Table Area detail [3]

Proper management and configuration of the interrupt table are crucial for efficient interrupt handling and ensuring that critical events are processed in a timely manner. Assigning appropriate priorities to different interrupts and implementing efficient ISR code, can allow to effectively manage the flow of interrupts and prioritize their execution based on the system's requirements.

This structure was crucial for the input signals received by the micro, responding with a high-speed interrupt that immediately enters the functions saved in its vector table.

4.3 Interrupt Handler

The interrupt handler is a software routine that is executed when an interrupt event occurs. It is responsible for handling and responding to the interrupt event by executing specific tasks or operations related to that event.

When an interrupt is triggered, the microcontroller pauses its current execution and transfers control to the corresponding interrupt handler routine. The interrupt handler performs the necessary actions to oversee the interrupt, such as saving the context of the interrupted code, processing the interrupt event, and restoring the context to resume the interrupted code execution seamlessly.

The interrupt handler routine typically consists of several steps, including interrupt flag checking, clearing the interrupt flag, performing the necessary operations, and returning from the interrupt. It is essential to keep the interrupt handler code as short and efficient as possible to minimize the impact on the overall system performance.

Developers can define their own interrupt handler routines for specific interrupt sources by writing the corresponding code and linking it to the appropriate interrupt vector address in the interrupt table. By customizing the interrupt handler routines, developers can tailor the system's response to specific events and implement desired functionalities.

Proper understanding and utilization of interrupt handlers are essential for effective and efficient interrupt-driven systems. Carefully designing and implementing interrupt handlers can ensure that critical events are promptly and accurately managed, minimizing system latency, and maximizing responsiveness.

When implementing an interrupt handler, several considerations should be made. First, the interrupt handler should focus on performing only the necessary tasks related to the interrupt event. Unnecessary operations or lengthy computations should be avoided to minimize the interrupt's duration and prevent delays in other system processes.

Furthermore, interrupt handlers should be designed to be re-entrant, meaning they can safely handle nested interrupts. This is particularly important in systems where multiple interrupts can occur simultaneously, like in the code developed for this SPTD. By carefully managing shared resources, such as variables or hardware registers, within the interrupt handler, developers can ensure proper operation even in complex interrupt scenarios.

Another critical aspect of interrupt handler design is interrupt prioritization. The microcontroller's interrupt table typically assigns different priorities to interrupts based on their importance or urgency. Carefully analyzing our system requirements allows us to assign appropriate priorities to different interrupts to ensure that higher-priority interrupts are serviced before lower-priority ones.

An example of such high priority can be found in the code of the SPTD, assigned to the LIN_RX pinout, configured as input, thus allowing us to manage the received LIN frame packet.

In addition to handling individual interrupts, interrupt handlers can also utilize interrupt flags or status registers to identify and oversee multiple interrupt sources within a single handler routine. This allows for efficient handling of related interrupts and reduces code duplication.

Overall, a well-designed and properly implemented interrupt handler mechanism is crucial for harnessing the full potential of the dsPIC33CDVL64MC106 microcontroller's interrupt capabilities. By effectively managing interrupts and utilizing interrupt handlers, robust and responsive systems can be created, capable of handling a wide range of real-time events and tasks.

4.4 SCCP

The dsPIC33CDVL64MC106 microcontroller is equipped with a powerful SCCP module, which stands for Synchronized Capture/Compare/PWM. This module offers versatile functionality for precise timing control, waveform generation, and event capture. It provides several independent capture/compare/PWM units, allowing simultaneous and independent operation of multiple functions.

The SCCP module supports various modes of operation, including capture mode, compare mode, and PWM mode.

- In capture mode, the module can capture the value of a specific input signal at a defined event, such as a rising or falling edge. This feature is particularly useful for measuring the duration or frequency of external events.
- In compare mode, the SCCP module compares the value of a selected input signal with a predefined reference value. This comparison can trigger an interrupt or generate an output signal based on the comparison result. Compare mode is commonly used for tasks such as event triggering, pulse width modulation, or duty cycle control.
- The PWM (Pulse Width Modulation) mode of the SCCP module allows for the generation of precise and adjustable waveforms with varying duty cycles. This capability is often utilized in applications such as motor control, power regulation, or audio synthesis.

The microcontroller's SCCP module provides flexible configuration options for PWM generation, enabling fine-grained control over the generated waveforms, an oversight of its Block Diagram can be seen in figure 3.4.

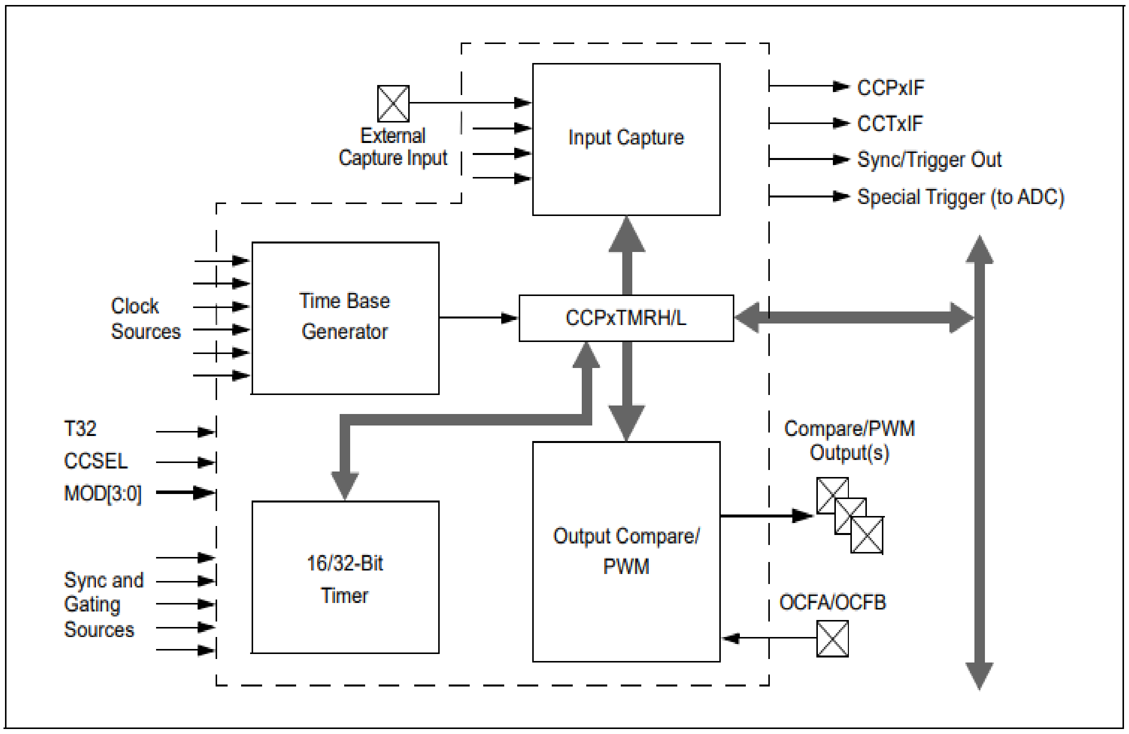


Fig. 3.4: Block Diagram of the SCCP module [3]

4.5 DAC

The aforementioned microcontroller includes a built-in DAC module, which stands for Digital-to-Analog Converter. The DAC module enables the conversion of digital data into analog signals, allowing for accurate and controlled analog output generation.

The DAC module supports multiple channels, each capable of independently converting digital data to analog signals. It utilizes a high-resolution architecture, providing precise and fine-grained output resolution. This makes it suitable for applications requiring accurate analog voltage generation, such as audio reproduction, waveform synthesis, or control signal generation.

To generate analog output, a configuration of the DAC module's input registers with digital values representing the desired analog voltage levels. The module then converts these digital values into corresponding analog voltages, which can be outputted through dedicated DAC pins. The output voltage range and resolution can be adjusted to meet specific application requirements.

The DAC module of the dsPIC33CDVL64MC106 microcontroller offers a reliable and efficient solution for generating analog signals. By utilizing its multiple channels, shown in figure 3.5, high resolution, and flexible configuration options, can achieve precise analog output control and meet the demands of various analog-based applications.

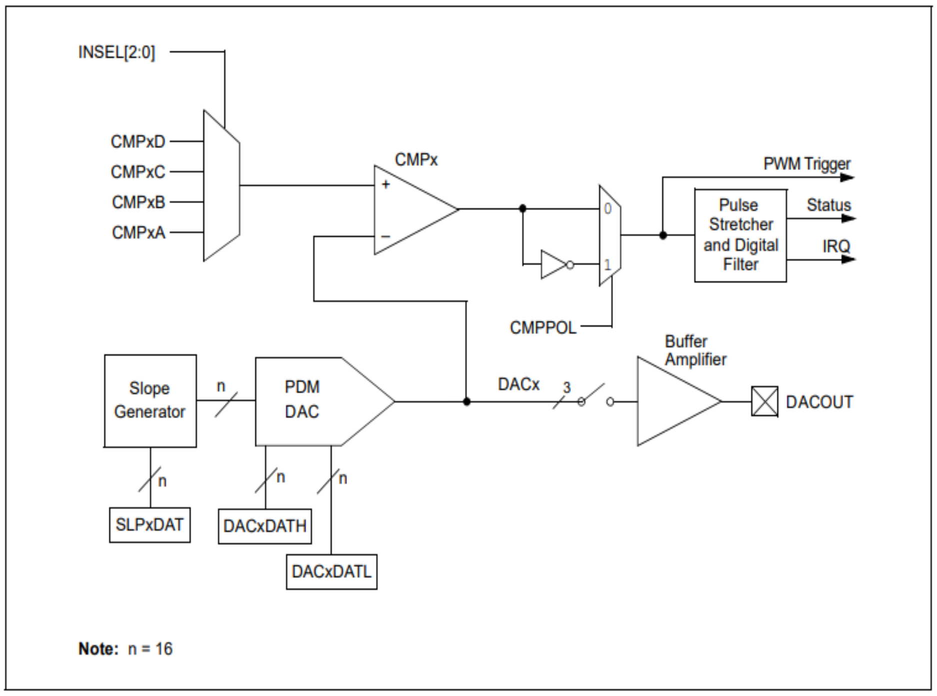


Fig. 3.5: Block Diagram of the DAC module [3]

4.6 ADC

The dsPIC features a powerful ADC module, which stands for Analog-to-Digital Converter. The ADC module enables the conversion of analog signals into digital data, allowing for accurate and precise measurement of analog inputs.

The ADC module supports multiple channels, allowing simultaneous conversion of multiple analog signals. Each channel is equipped with its own sample-and-hold circuitry, which captures and holds the input voltage during the conversion process. This ensures accurate and stable measurements, even in the presence of varying input signals.

It supports different sampling rates and conversion modes, enabling adaptation to various application requirements. The module also provides advanced features, such as hardware oversampling and averaging, to further enhance the accuracy and quality of the converted digital data.

Using the ADC module effectively allows the capture and conversion of analog signals into digital data with high precision and reliability. This data can then be processed, analyzed, or utilized for further control and decision-making, such as sensor interfacing, data acquisition, or signal analysis.

The module consists of one shared SAR ADC core, as shown in figure 3.6, where SAR is the key component of the ADC that performs the actual conversion and stands for Successive Approximation Algorithm.

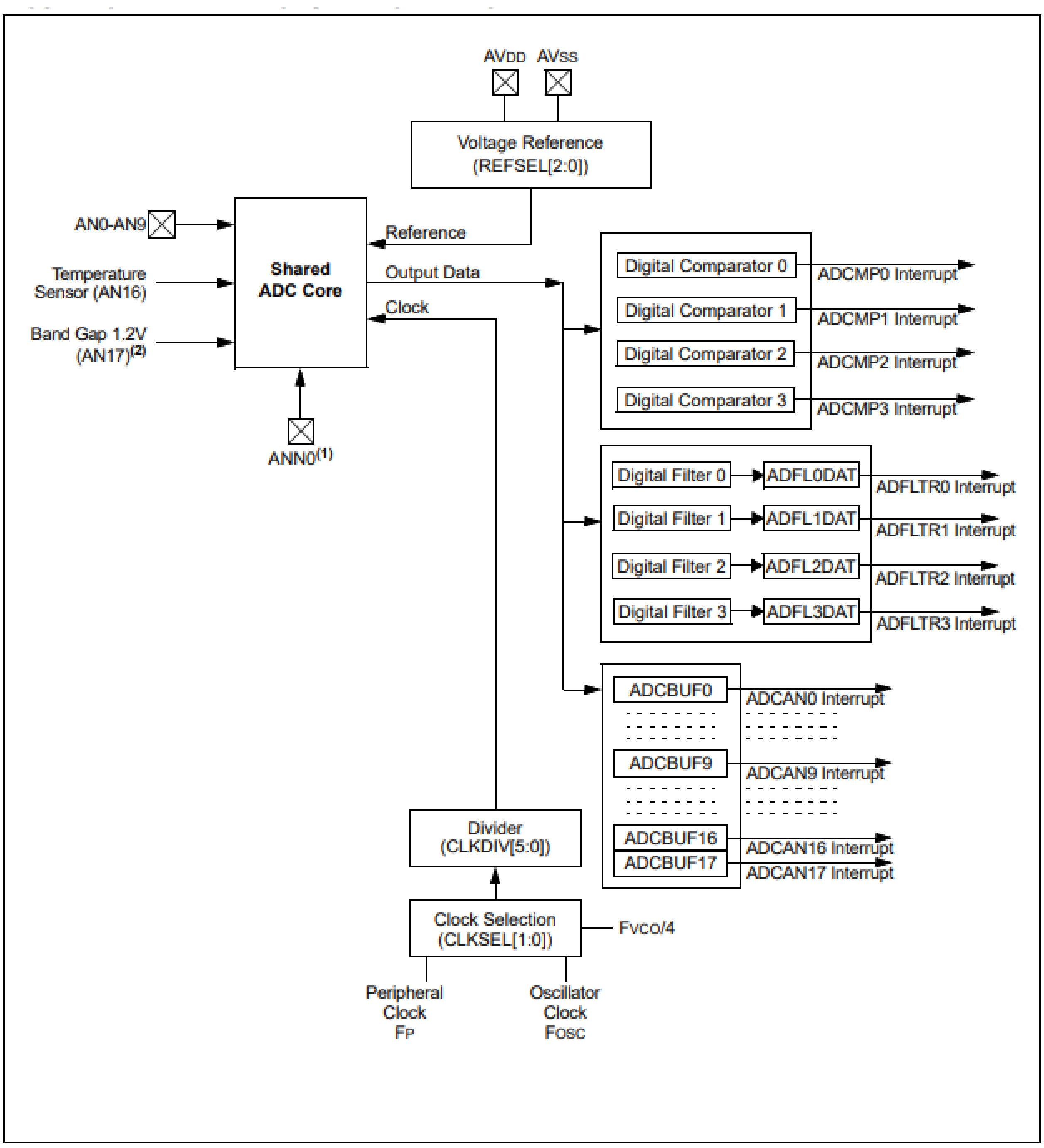


Fig. 3.6: Block Diagram of the DAC module [3]

5. LIN

The Lin (Local Interconnect Network) is a serial communication protocol specifically designed for automotive applications.

LIN offers a cost-effective and reliable solution for in-vehicle networks, working as a sub-bus for the Controller Area Network (CAN).

It enables communication between various electronic components within a vehicle, such as those shown in FIG. 4.1.

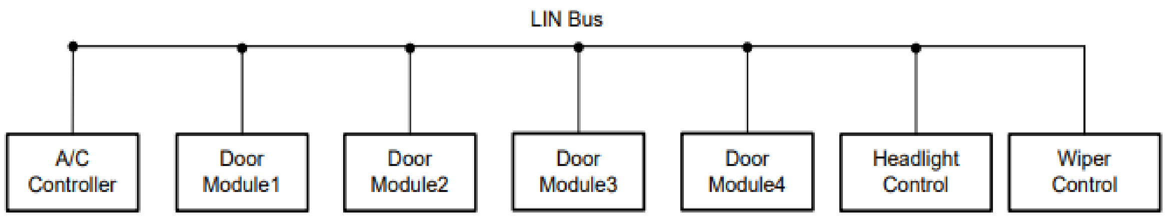


Fig. 4.1: LIN BUS connections to various car components.

The LIN protocol operates on a single-wire bus, reducing the number of physical connections.

It is a low-cost network based on ISO 9141 (K-LINE) simplifying not only the wiring but also reducing the overall complexity of the vehicles' electrical system.

5.1 Introduction to LIN Protocol

The LIN protocol offers severe key features for

- Single master and multiple slaves: The LIN network consist of a single master node that controls communication with multiple slave nodes. This hierarchical structure enables efficient data exchange between master and slaves (max 16).
- Speed up to 20 kbps: This lower speed compared to other protocols is best suitable for applications that do not require high data transfer rates, contributing to the cost-effectiveness.
- Hierarchical network: The LIN network operates in a hierarchical manner, with the master node responsible for initiating communication and the overall network. The slave nodes can only respond to the master's command and transmit the required data.

5.2 LIN Frames

LIN communication is organized into frames, which are the basic units of data transmission in the LIN protocol. Each frame consists of a header and a response, containing relevant information such as the sender's identifier, data length, and the actual payload.

As we already mentioned it is a hierarchical network, so the headers can only be sent by the masters, while the slaves cannot initiate communication.

A LIN frame packet is shown in figure 4.2 and will follow its analysis in two steps.

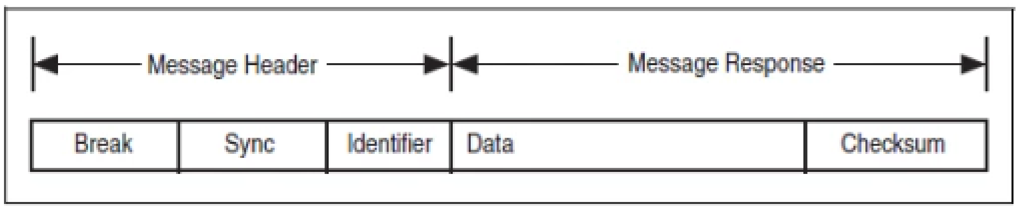


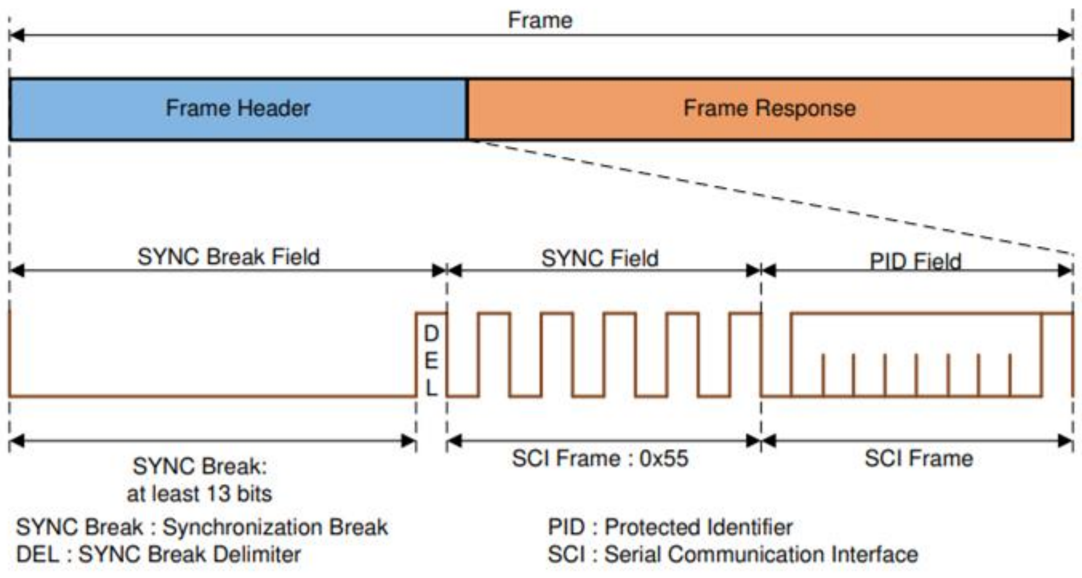
Fig. 4.2: LIN Frame composition [6]

The LIN Master initiates the communication by sending the header.

5.3.1 LIN header

The LIN header is a crucial part of the LIN frame, containing essential information for the recipient to identify and process the incoming data. It includes the sender's identifier, which helps the receiving node determine the source of the data transmission. Additionally, the header specifies the data length, indicating the number of bytes to be received in the response.

Proper interpretation and handling of the LIN header are vital for successful communication within the LIN network. Understanding the structure and significance of the LIN header is essential for developers working with LIN-based systems.



Frame
Frame Header
Frame Response
SYNC Break Field
SYNC Field
PID Field
D
E
L
SYNC Break:
at least 13 bits
SCI Frame : 0x55
SCI Frame
SYNC Break : Synchronization Break
DEL : SYNC Break Delimiter
PID : Protected Identifier
SCI : Serial Communication Interface

Fig. 4.3: LIN Header description [6]

The Header of this serial transmission protocol is compose of:

- Break field: this is a 13-bit dominant signal, followed by a break delimiter of 1-bit recessive signal. This field has the goal to announce the beginning of a transmission.
- Sync field: as the name suggests it allows the synchronization of the slaves with the master transmission, through an autobaud detection. This sync field is the representation of 0x55 character.
- Identifier: this is the last field transmitted by the master and plays a significant role in LIN communication. It contains information about frame type, message source and destination. This ID allows the receiving node to identify the recipient and interpret the transmitted data accordingly.

All the slaves are continuously listening and through the verification of their parity bits are able to determine if they are subscribers or publishers for the specific ID.

As we can see from figure 4.3 an 8-bit PID is sent through the communication channel, rather than the ID, and it contains in the lower six bits the raw ID, and the upper two bits contain the parity.

Per our LIN specification the parity bit is calculated as follows:

- Parity bit 1 = ID (1) ^ ID (3) ^ ID (5).

- Parity bit 2 = ID (0) ^ ID (2) ^ ID (4).

While the specific calculation of the PID field may vary depending on the LIN version, its purpose does not change, it serves as an error detecting mechanism and for some versions Error correction as well.

Such field is used to ensure data integrity during transmission and to enhance the reliability of the communicated information.

After the transmission of the LIN header, the communication can take two directions based on the assignment of the 6-bit ID of the frame. This ID assignment determines whether the frame acts as a subscriber or a publisher within the LIN network.

- If the frame is designated as a subscriber, the master node will receive data from the corresponding slave node(s). In this case, the slave node(s) actively sends information to the master node, allowing for data exchange and updates.
- On the other hand, if the frame is configured as a publisher, the master node takes the role of transmitting data to the slave node(s). The master node initiates the communication and sends relevant information or commands to the designated slave node(s) by filling in the Frame response as well as the header.

This distinction between subscriber and publisher frames enables efficient and targeted communication within the LIN network. Subscriber frames facilitate the distribution of data from slave nodes to the master node, ensuring that the master node stays updated with the latest information. Publisher frames, on the other hand, enable the master node to transmit essential data or commands to the slave node(s), controlling their actions or requesting specific information.

By supporting both subscriber and publisher frames, the LIN protocol enables bidirectional communication between the master and slave nodes.

5.3.2 LIN Response

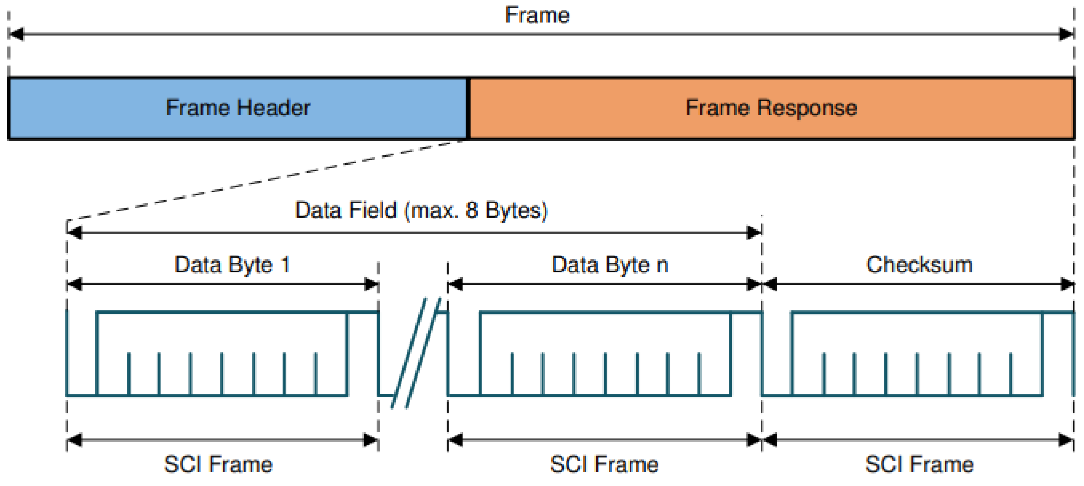


Fig. 4.4: LIN Response Description [6]

The LIN response serves as the reciprocal part of the LIN header in the communication procedure. Once the header has been transmitted, by the master node, the slave node(s) within the LIN cluster respond with the corresponding data.

The response frame contains valuable information that the master node needs to interpret and process the received data. It includes the data payload, which carries the actual information being transmitted, and the checksum for error detection.

The response frame as shown in figure 4.4 consists of the following components:

- Data Length: This field specifies the number of bytes included in the response frame. It allows the receiving node to determine the length of the data payload and properly extract the transmitted information.
- Data Payload: The data payload contains the actual information transferred from the slave node to the master node. The content and format of the payload depend on the specific application and the data being exchanged.
- Checksum: The checksum plays a critical role in the response frame. It is calculated based on the data payload and serves as an error detection mechanism. The receiving node recalculates the checksum using the received data payload and compares it to the checksum value included in the response frame. If the calculated checksum matches the received checksum, the data is considered valid and error-free.

The master node must accurately process the received response frame, extract the transmitted data, and verify its integrity using the checksum.

It is important to note that the structure and interpretation of the response frame may vary depending on the specific LIN version and protocol being used. The LIN specification provides detailed guidelines and algorithms for constructing and interpreting the response frame to ensure consistent and reliable communication across different LIN devices and systems.

6. Code Structure and Organization

The code structure and organization of the project were designed to ensure efficiency in the implementation of the firmware.

In addition to modularity, the code structure underlines code reusability, where common functions or modules, regarding the configuration and usage of the deployed peripherals, are designed to be easily reused in various parts of the firmware. This is mostly due to the fact that the root of the code was generated by MCC.

The initial code generation by the MCC tool served as a valuable starting point for the project, providing a good foundation that established optimal coding conventions.

Building upon this existing codebase ensured a consistent structure and organization throughout the firmware.

6.1 Architecture

The architecture follows a modular approach, allowing for easier maintenance and

scalability for future upgrades, as per the requirements of the project.

Furthermore, it is a heavily interrupt-driven architecture leveraging the capabilities of the dsPIC33C microcontroller, through the usage of ISR’s, interrupt service routines.

6.2 FSM

The various states that rule the behaviors of the SPTD can be easily described through a Finite State Machine.

A simple example is shown in Fig. 5.1, where the normal working state is IDLE.

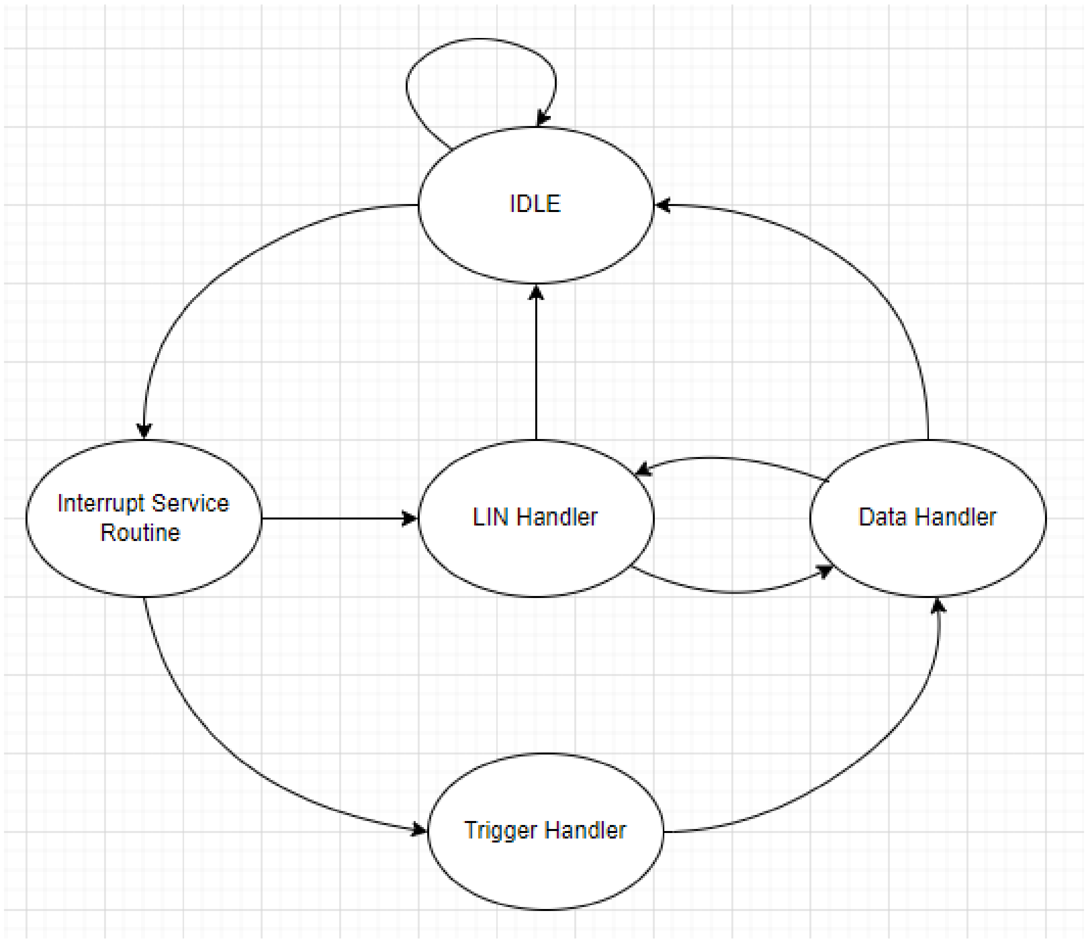


Fig. 5.1: Finite State Machine of the SPTD

The states of the machine are:

1) IDLE
2) Interrupt service routine.
3) Trigger Handler
4) Data Handler.
5) Lin handler

The FSM operates based on an interrupt-driven architecture, where external events trigger specific transitions and state changes. These events can be generated by various sources, such as sensor inputs, communication interfaces (LIN). Upon receiving an event, the FSM transitions to a new state, which determines the system's behavior and the actions to be executed, the interrupt service routine. The ISR is associated with the interrupt vector table, a specific lookup table that maps interrupt events to their corresponding ISR addresses in memory.

These interrupts can occur at any time during the execution of the program and can also be triggered by internal conditions and timers. When an interrupt event occurs, it temporarily suspends the normal program flow and transfers control to the corresponding ISR.

This finite state machine design emphasizes the codes modularity, with two key modules: the LIN handler and the Trigger Handler.

The LIN handler module is responsible for processing LIN communication messages. It handles the reception and transmission of LIN frames, parses the received data, and generates appropriate responses. This module ensures seamless communication between the microcontroller and Engine Control Unit (ECU).

The Trigger Handler module manages trigger events generated by external sources that are the same ones that are received by the ignition coil and the spark plug modules.

It captures and compares trigger signals, enabling specific actions or state transitions within the FSM. This module is responsible for coordinating timing-related operations and synchronizing the system's behavior with such external events, with the purpose to calculate data about the state of these ignition components. Data that will be accessed and transmitted by the LIN handler.

The modularity of the LIN handler and Trigger Handler modules allows them to be independently developed, tested, and replaced, if necessary, without affecting the overall functionality of the FSM.

6.3 The main Function

The main function of the firmware provides a simple approach to configure and initialize all the necessary modules discussed in the previous chapters. Starting with the system module, a vital part of the MCU, it allows to configure the clock source. Followed by the configuration of the appropriate pins as input/output to ensure all the functionalities of the SCCP are being correctly used with their relative DAC modules as well.

Then the Interrupt Vector Table is configured assigning priorities one through seven to each of the various modules. Highest priority is given to the signal inputs received from the ECU, interfaced with the SCCP module, followed by the LIN RX pin where the LIN messages are received.

The main algorithm proceeds with other essential tasks, such as setting up timers for various operations, including trigger event synchronization and timing calculations. These timers play a crucial role in calculating the required data.

The main algorithm also invokes the LIN Handler module and Interrupt Handler module, critical components that handle LIN communication and manage interrupt-driven events, respectively.

The simplicity of this main function showcases the result of the modular approach and the advantages of the Interrupt Vector Tables.

6.4 Data extraction

In the data extraction process, the firmware utilizes the SCCP (Synchronized/Capture/Compare/PWM) module to extract and process signals from the ignition system. Specifically, the firmware captures and converts four signals received from the Engine Control Unit (ECU) and inputs them into the SCCP modules. These signals correspond to the same ones received by the ignition coil module, given as output by it, and the ones received by the plug top.

The SCCP module plays a key role in calculating timing parameters and comparing the received signals. They provide advanced timing control and synchronization capabilities, enabling precise measurement and analysis of the complete ignition system.

To provide a visual representation of the received signals, Figure 5.2 illustrates the captured signals. This diagram helps visualize the characteristics and variations of the signals as they are processed by the SCCP module.

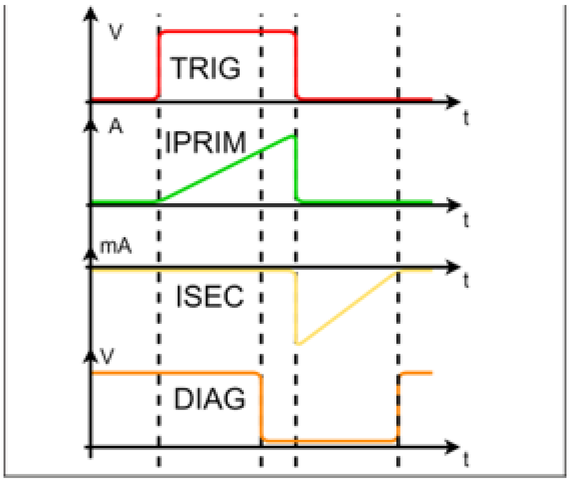


Fig. 5.2: Signals received as input to SCCP module.

The data extraction phase forms a critical step in the overall functionality of the firmware. By capturing and analyzing the ignition signals using the SCCP module, the firmware obtains valuable information about the timing and characteristics of every ignition cycle. This data serves as a basis for further calculations and decision-making within the firmware, ultimately influencing the control and optimization of the engine's performance.

6.5 Data transmission

The firmware utilizes the LIN handler module to manage the LIN communication process.

It is important to note that the specifics of the LIN communication protocol, including the frame structure, message format, and transmission protocols, are standardized and follow industry specifications, the ISO17897 specifications.

The LIN Handler module is responsible for populating the response header of the LIN frame with appropriate data. This response header contains the necessary information to convey the desired response to the receiving device.

Through the LIN communication interface, the firmware transmits the response header along with any accompanying data to the target device.

It is worth mentioning that the details of the specific data being transmitted, as well as the content of the response header, are project specific.

7. Conclusion

The performance of an internal combustion engine is, to some extent, depending on the spark plug working conditions which can be modified during engine life. This is due to the ageing process, caused by the spark gap wearing. Other abnormal conditions that lead to a reduction of engine performance are spark gap fouling due to the presence of engine oil, water or moisture coming from combustion's residuals.
This gap fouling can create a short circuit of the spark gap which suppresses the spark or creates anomalous path to ground, thus promoting an inefficient combustion system.
The present device will allow the engine ECU to monitor the correct functioning of these ignition components and extrapolate useful information of the spark plug gap conditions.

In conclusion, this invention presents a device capable of effectively communicating the real-time status of ignition coils and spark plugs to the ECU using a robust digital diagnostic signal. The utilization of a digital signal offers a lot of advantages, including superior noise immunity, ensuring reliable and accurate data transmission. Moreover, the digital signal's scalability and flexibility enable the transmission of multiple signals simultaneously, further enhancing the functionality and performance of the system. Overall, this invention represents a significant advancement in diagnostic signal transmission, contributing to the overall efficiency and reliability of automotive systems.

Bibliography

[1] Microcontroller. In Wikipedia. Available at: [https://en.wikipedia.org/wiki/Microcontroller#Embedded_design]

[2] Brian Bailey. "MPU vs MCU" (2016). Available at: [https://semiengineering.com/mpu-vs-mcu/]

[3] Microchip Technology Inc. "dsPIC33CDVL64MC106 Family Data Sheet" (2021). [Accessed: 05/17, 2023]. Available: [https://ww1.microchip.com/downloads/aemDocuments/documents/MCU16/ProductDocuments/DataSheets/dsPIC33CDVL64MC106-Family-Data-Sheet-DS70005441.pdf]

[4] Microchip Technology Inc. "dsPIC33CDV64MC106 Motor Control Development Boards User's Guide" (2021). [Accessed: 05/15, 2023]. Available: [https://ww1.microchip.com/downloads/aemDocuments/documents/MCU16/ProductDocuments/UserGuides/dsPIC33CDVL64MC106-and-dsPIC33CDV64MC106-Motor-Control-Development-Boards-Users-Guide-DS50003060.pdf]

[5] Microchip Technology Inc. "LIN Specification Package." revision 2.2A (2016). Available at: [https://microchipdeveloper.com/local--files/lin:specification/LIN-Spec_2.2_Rev_A.PDF]

[6] Eric Hackett. "LIN Protocol and Physical Layer Requirements." Texas Instruments Application Note, SLLA383A, (2018). Available at: [https://www.ti.com/lit/an/slla383a/slla383a.pdf?ts=1684431200094]

A. Digital Signal Microcontroller

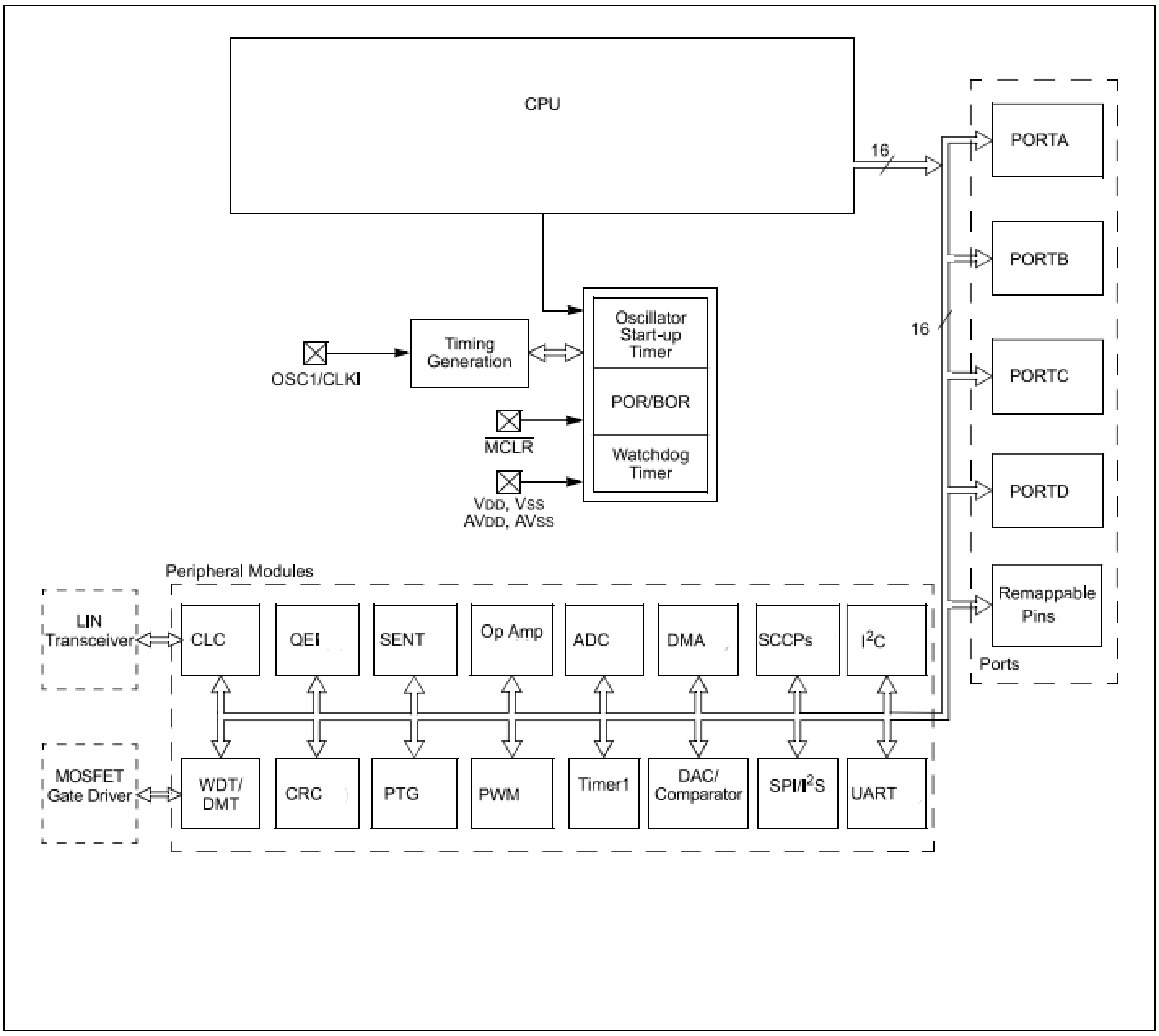


Fig. A.1: Block Diagram of the dsPIC33CDVL64MC106 [3]

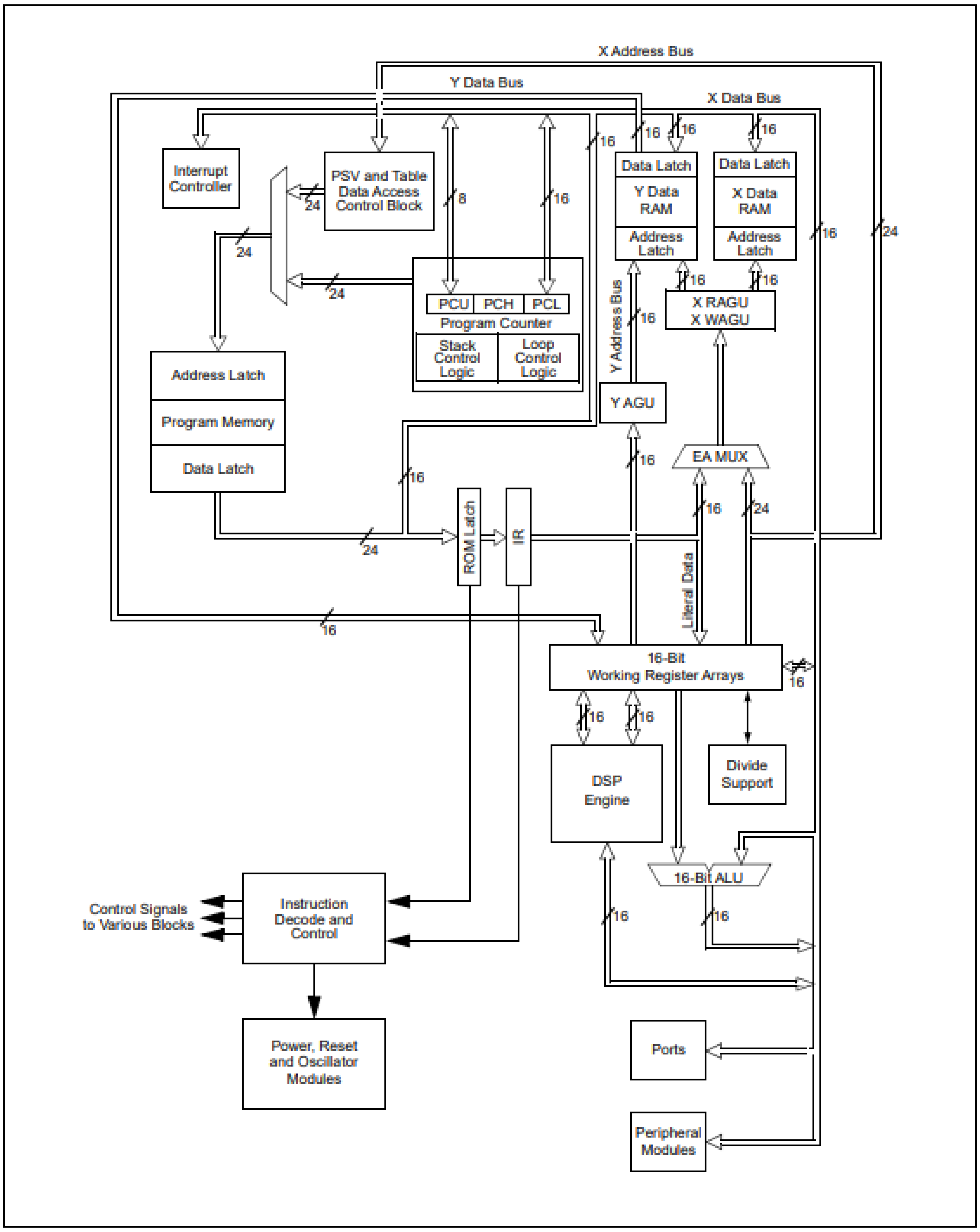


Fig. A.2: Block Diagram of the 16-bit CPU of the dsPIC [3]

B. Development Board

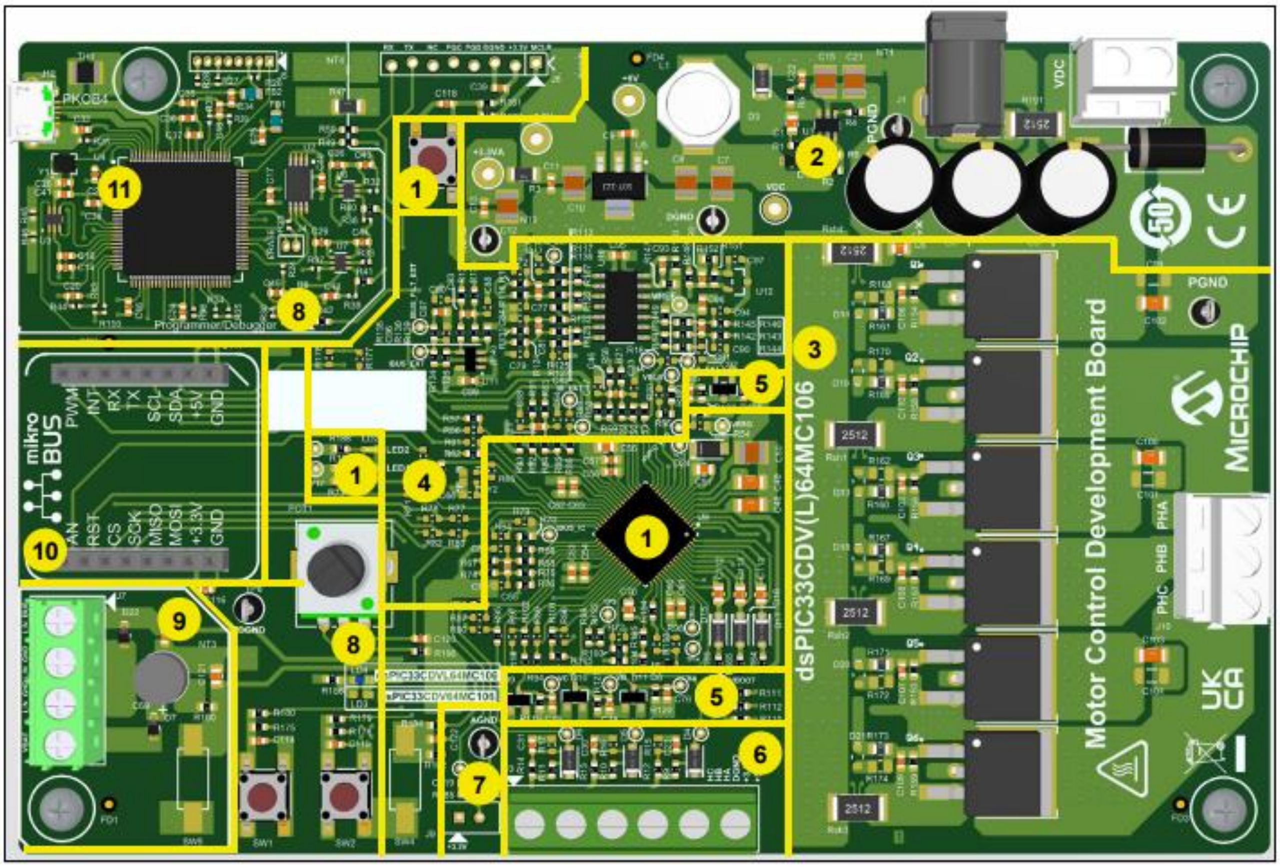


Fig. B.1: Top side layout with hardware division in sections [4]

Section	Hardware Section
1	dsPIC33CDVL64MC106 and Auxiliary Circuits
2	Power Supply Section
3	Three-Phase Motor Control Inverter
4	Current Sensing Circuits
5	Voltage Sensing Circuits
6	Speed or Position Feedback Interfaces
7	External Temperature Sensor Interface
8	User Interface
9	LIN Bus Interface
10	mikroBUS™ Socket
11	PICkit™ On-Board (PKOB)

Tab. B: Top side layout with hardware division in sections [4]

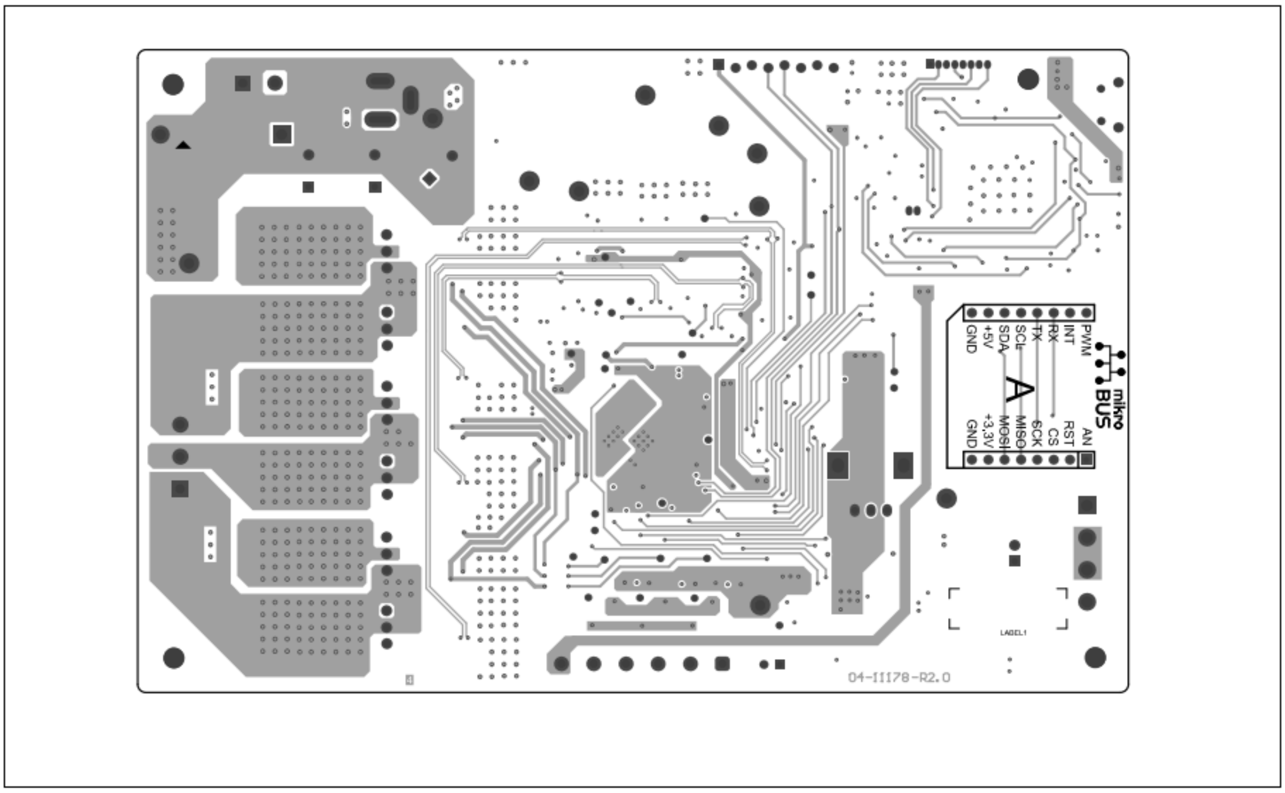


Fig. B.2 Bottom side layout of the development Board [4]

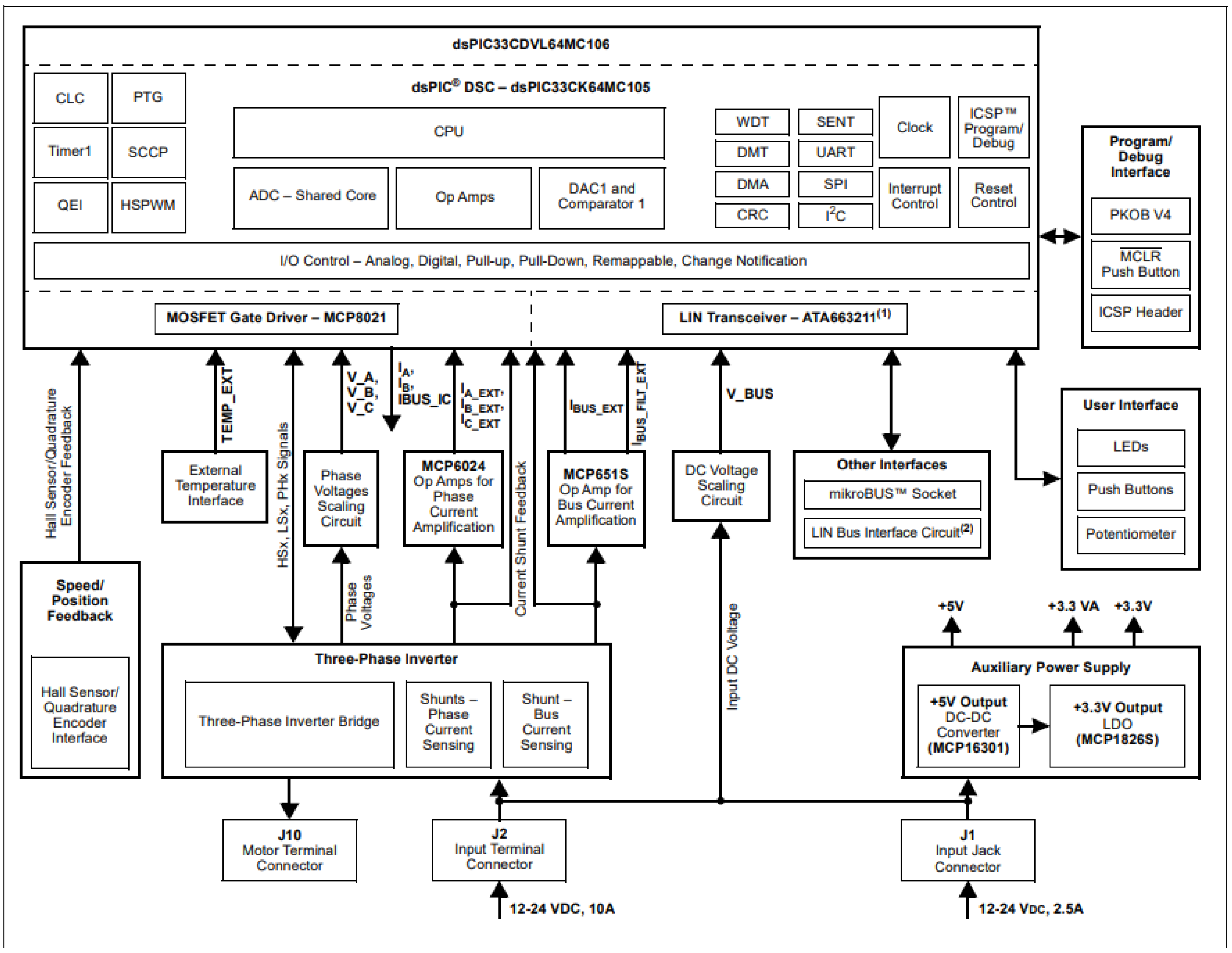

dsPIC33CDVL64MC106
dsPIC® DSC – dsPIC33CK64MC105
CLC
PTG
Timer1
SCCP
QEI
HSPWM
CPU
ADC – Shared Core
Op Amps
DAC1 and Comparator 1
WDT
SENT
DMT
UART
DMA
SPI
CRC
I²C
Clock
ICSP™ Program/ Debug
Interrupt Control
Reset Control
I/O Control – Analog, Digital, Pull-up, Pull-Down, Remappable, Change Notification
MOSFET Gate Driver – MCP8021
LIN Transceiver – ATA663211(1)
Program/ Debug Interface
PKOB V4
MCLR Push Button
ICSP Header
User Interface
LEDs
Push Buttons
Potentiometer
Hall Sensor/Quadrature Encoder Feedback
TEMP_EXT
HSx, LSx, PHx Signals
V_A, V_B, V_C
IA, IB, IBUS_IC
IA_EXT, IB_EXT, IC_EXT
Current Shunt Feedback
IBUS_EXT
IBUS_FILT_EXT
V_BUS
External Temperature Interface
Phase Voltages Scaling Circuit
MCP6024 Op Amps for Phase Current Amplification
MCP651S Op Amp for Bus Current Amplification
DC Voltage Scaling Circuit
Other Interfaces
mikroBUS™ Socket
LIN Bus Interface Circuit(2)
Speed/ Position Feedback
Hall Sensor/ Quadrature Encoder Interface
Phase Voltages
Input DC Voltage
Three-Phase Inverter
Three-Phase Inverter Bridge
Shunts – Phase Current Sensing
Shunt – Bus Current Sensing
+5V
+3.3 VA
+3.3V
Auxiliary Power Supply
+5V Output DC-DC Converter (MCP16301)
+3.3V Output LDO (MCP1826S)
J10 Motor Terminal Connector
J2 Input Terminal Connector
J1 Input Jack Connector
12-24 VDC, 10A
12-24 VDC, 2.5A

Fig. B.3: Block Diagram of the development Board [4]

Acknowledgments

I would like to express my deepest gratitude and appreciation to my incredible mother. Her unwavering love, support, and encouragement throughout my academic journey, have been immeasurable. Her sacrifices and belief in me have been the driving force behind my success. I am forever grateful for her guidance and presence in my life. Everything that I have achieved in my life, I owe it to Her, so thank you Fatima.

Furthermore, I extend my heartfelt appreciation to my sisters. Their constant encouragement, wisdom, and assistance have been fundamental in shaping not only my academic path, but also my life.

I would like to acknowledge the incredible support I received from my friends, who have formed a village of support around me. Through their friendship, they have uplifted me during challenging times and always celebrated my successes. Their guidance and presence in my life have made all the difference, and I am grateful for their unconditional support.

I am indebted to my Erasmus friends who made my study abroad experience truly unforgettable, despite the challenging circumstances posed by the COVID-19 pandemic. Their friendship, camaraderie, and shared experiences have enriched me. Their support and their laughter have made this journey even more meaningful, and I am grateful for the lifelong memories we have created together. Memories that start from Study-Inn and lead you to Northfield.

A special shoutout goes to my tutor, Eng. Simone Daniele, whose passion and encouragement shaped my intellectual curiosity and made me want to further develop my studies in this field.

Lastly, I would like to extend my appreciation to all the professors, mentors, and educators who have contributed to my academic growth and development.

I am profoundly grateful to everyone who has played a role in this journey and has contributed to my personal and professional growth. I now understand the saying “it takes a village to raise a successful individual”, and I am proud to say that my journey has indeed taken an entire village to reach its destination.

Your support, love, and guidance have made this achievement possible, and I am forever thankful.