MASTER

A Real-time Network for a Solid-State Transformer

Felkaroski, Nikola

Award date:
2022

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Electrical Energy Systems Research Group

A Real-time Network for a Solid-State Transformer

Graduation project
MSc Embedded Systems
30 ECTS

Nikola Felkaroski

Assessment committee:

Assistant Professor Dr. Vladimir Ćuk
Full Professor Dr. Korneel Wijnands
Associate Professor Dr. Mircea Lazar
Doctoral Candidate ir. Bram van Dam

Version 1.0

Eindhoven, August 2022

Abstract

Solid-state transformers (SSTs) are recognized as a potential solution for numerous grid challenges caused by the emergence of renewable energy sources, electromobility, and growing energy demands. The SST, a power electronics-based replacement for the conventional transformer, introduces control and intelligence into the network through power management, reactive power compensation, power quality regulation, and additional fault protection. However, the benefits of the SST come at the expense of high complexity, which is often mitigated through modularity. Consequently, modular SSTs require real-time control and synchronization to operate correctly, necessitating real-time networks (RTNs). This work recognizes the gap in the literature concerning real-time network requirements for SSTs. Furthermore, it contributes to the field by introducing a requirement analysis and a proposed real-time network design supporting full integration and interface with Simulink to facilitate further SST research and development. The identified and proposed SST RTN requirements include network performance, synchronization, data frame, network topology, transmission medium, scalability, fault tolerance, and ease of use. The proposed design utilizes the module controller architecture to decouple communication and control to enable Simulink integration and increase determinism. Finally, the design is verified on a pragmatic SST prototype developed at the TU/e. Experimental measurements and analysis show that the proposed real-time network can support up to 35 modules, i.e. a 94.4% increase compared to the original 18-module design, without impacting its performance.

Acknowledgement

I would like to express my deepest gratitude to my graduation supervisor, Vladimir Cuk, and the assessment committee members for their excellent guidance, support, and feedback throughout this project. Furthermore, this endeavour would not have been possible without the guidance of Bram van Dam, my graduation tutor, who generously provided his knowledge, expertise, and moral support. Bram, I cannot overstate how much I enjoyed working with you and how much I am grateful for everything. You have been an outstanding mentor and a friend, and I hope we will be colleagues again in the future.

Lastly, I would like to thank my family for their unlimited support throughout my journey and for helping me overcome all my problems and worries. The last two years have been very challenging for me given all the circumstances, but their belief in me has kept my spirits and motivation high during this process.

Contents

List of Figures

List of Tables

Abbreviations

AC	Alternating current
ADC	Analog to digital converter
CHB	Cascaded H-bridge
CPU	Central processing unit
DAB	Dual-active bridge
DC	Direct current
DMA	Direct memory access
ePWM	Enhanced pulse width modulator
ESC	EtherCAT Slave Controller
FET	Field-effect transistor
GPIO	General purpose input/output
HFT	High-frequency transformer
HVDC	High-voltage DC
IGBT	Insulated-gate bipolar transistor
IPC	Interprocessor communication
INT16	16-bit signed integer
I/O	Input/output
LFT	Low-frequency transformer
LSB	Least significant bit
LV	Low voltage
MMC	Multilevel modular converter
MSB	Most significant bit
MV	Medium voltage
PLC	Programmable logic controller
PWM	Pulse width modulation
SSB	System sensor board
SSC	EtherCAT Slave software stack
SST	Solid-state transformer
TI	Texas Instruments
TU/e	Eindhoven University of Technology

Chapter 1

Introduction

The emergence of renewable energy sources to solve increasing energy demands brings new challenges to existing electrical grid infrastructure. Moreover, the gradual shift from centralized to distributed energy production introduces additional unpredictability and fluctuations in the generated energy. Conventional low-frequency transformers (LFTs) are used as passive interfaces between the low-voltage (LV) grid and the medium-voltage (MV) grid. Therefore, fluctuations in power quality at one side of the LFT are also coupled to the other, which may cause problems and possibly breach limits of voltage, power, and frequency levels. Furthermore, traditional LFTs cannot achieve desired requirements such as power flow control and further transmission losses.

The Solid-State Transformer (SST) is introduced as a potential solution to several significant issues that conventional transformers exhibit. The conceptualization of a power electronics-based transformer started nearly 60 years ago [3, 4] and various terminology has been used such as 'electronic transformer,' 'energy router,' 'power electronics transformer.' The SST solution replaces the bulky LFT with a high-frequency transformer (HFT) and power electronics. The power electronics interface introduces control and intelligence to the SST, allowing for power management, reactive power compensation, power quality regulation, and added fault protection [5]. The inclusion of such features makes the SST an enabling technology for the proposed 'energy internet' [6]. Additionally, the gradual improvement of the used power electronic components would eventually make the SST preferable for weight and volume-constrained applications.

Recognizing the significance of the SST technology, the Eindhoven University of Technology (TU/e) is building a prototype of its own to enable further research and possibly build a distribution network-ready solid-state transformer. This proposed graduation project is part of the ongoing TU/e SST research project conducted by Bram van Dam, a doctoral candidate.

Many variations and topologies of the SST have been proposed in the literature, in which a common trend is the use of modularity to cope with higher voltages than the available ratings of currently available semiconductor switches such as SiC FETs and Si IGBTs. Even with the inevitable developments and improvements in the semiconductor industry, modularity is still very desirable in SSTs as it allows expansion and adjustments of the SSTs to higher power demands and introduces redundancy. However, the cost of modularity is the increased complexity and necessity for control. Due to the vulnerability of available semiconductor components to unbalanced high voltages or currents, the control application is time-critical. Therefore, a real-time network enabling module communication and control is necessary to operate a modular SST properly.

This study is organized as follows: chapter 1 introduces the TU/e SST and the concepts of real-time networks, specifically EtherCAT. In chapter 2, a literature review of the state-of-the-art is presented and the research problem is defined. Next, in chapter 3, the requirements for real-time networks in SSTs are investigated and defined. Then, chapter 4 presents the proposed design and implementation of the SST real-time network, and chapter 5 demonstrates experimental verification and analysis of the result. Finally, chapter 6 summarises conclusions and future work.

1.1 The TU/e Solid-state transformer

1.1.1 System design

The TU/e Solid-state transformer (from now on referred to as the SST) is a proposed SST design, uniquely developed to comply with the grid standards for conventional MV transformers, i.e., the IEC60076-3 standard. The design is modular to handle the high MV voltage and power using power electronics, containing six modules per phase, or 18 total, as proposed in [7]. Each module contains three stages: an input rectifier stage, an isolation stage, and an output inverter stage. The fundamental electrical design of the SST module is shown in Figure 1.1. As seen in the figure, the first stage is a cascaded H-bridge (CHB) composed of 4 IGBTs that rectifies the input voltage. The second stage includes a dual-active bridge (DAB) and a high-frequency transformer. As a result, power can be transferred from the MV isolation side to the LV side with up to 97.5% efficiency [7]. The third stage is an inverter which converts the LV DC voltage to LV AC voltage. The design emphasis of the modules in the TU/e SST project is on the input rectifier and isolation stages, whereas the inverter stage is implemented using off-the-shelf conventional inverters. The presented module is part of the three-phase modular design depicted in Figure 1.2.

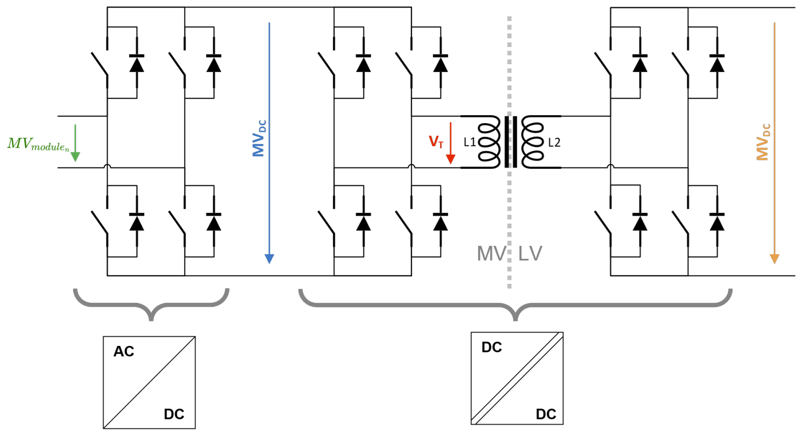


Figure 1.1: The SST module design.

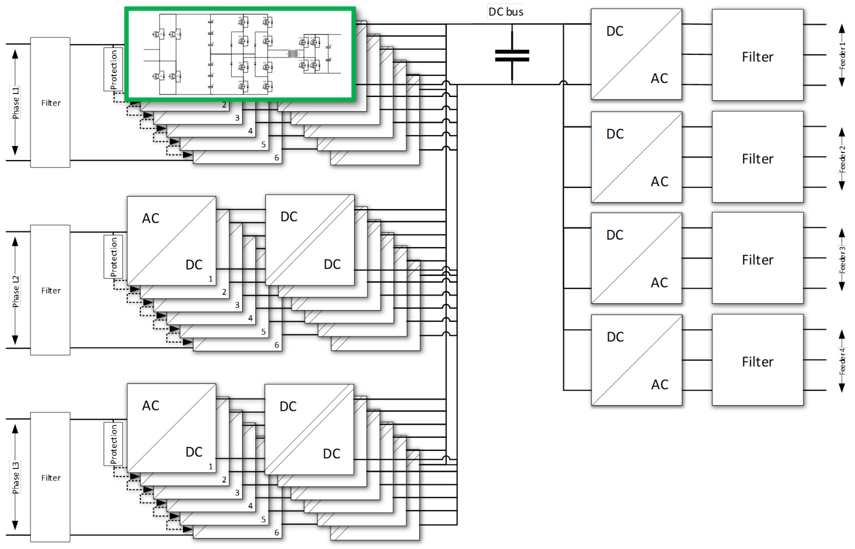


Figure 1.2: The full TU/e SST modular design.

1.1.2 Control

The control strategy of the TU/e SST is hybrid, meaning that parts of the control are done locally by the module controllers, and another part is done on a system level. While selecting a control strategy is out of this project's scope, it significantly influences the real-time network's requirements.

At a module level, the module controller is responsible for controlling the voltages, currents, and transferred power through the module. Various sensors measuring these parameters are read through the controller's ADCs. In addition, the switches are interfaced to the GPIOs of the module controller, allowing for various controllable switching patterns. Each SST module has two independent local control loops. The first control loop regulates the input current i_{CHB} through the CHB and the voltage V_{cap} in the capacitor banks between the CHB and the DAB. The capacitor voltage is controlled through PWM switching patterns for the input current. The second control loop regulates the DC bus voltage V_{DCbus} to a reference voltage of $700V$ under different load conditions through phase-shift control. As a result, power flows through the two sides of the bridge with a phase angle difference, which can be adjusted to control the amount of transferred power. A simplified module-level control diagram is presented in Figure 1.3.

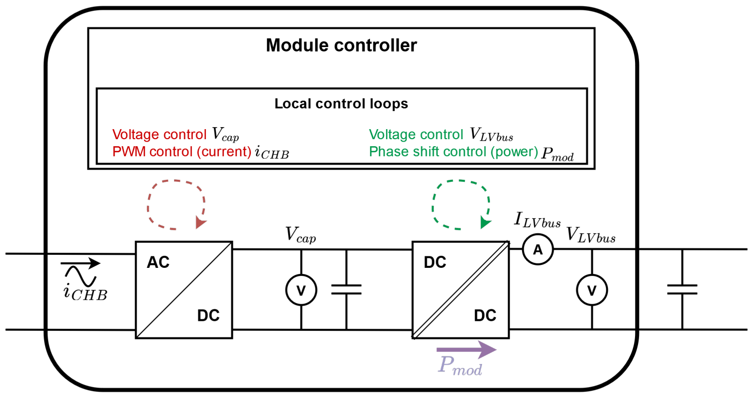

Module controller
Local control loops
Voltage control V_{cap}
PWM control (current) i_{CHB}
Voltage control V_{LVbus}
Phase shift control (power) P_{mod}
i_{CHB}
AC
DC
V_{cap}
V
DC
DC
I_{LVbus}
A
V_{LVbus}
V
P_{mod}

Figure 1.3: Simplified module-level control

At a system level, a system controller implements two additional control loops to ensure the proper operation of the entire SST. Figure 1.4 shows a simplified diagram of the two system-level control loops by separating the inputs and outputs for both loops.

The first control loop, the MV voltage loop, regulates the input current incoming to each module through PWM (in conjunction with the local CHB current loop) and ensures a sinusoidal drawn current from the grid. Additionally, the control loop ensures that the MV voltage is equally divided between the participating modules in a single phase., i.e., each module generates $\frac{1}{n} \cdot V_{MV_{grid}}$, where n is the number of modules in a phase. Finally, the system controller reads the grid voltage and current and sends a sinusoidal PWM to the modules, synchronizing the local CHB control.

The second system-level control loop ensures balancing between the MV capacitors of the modules in a phase since a significant imbalance could lead to a catastrophic cascading failure of the modules due to the voltage and power limitations of the switches. The system controller reads the actual MV capacitor voltages of the modules and sends a capacitor voltage setpoint for each module so that the local voltage control loop can adjust the voltage.

The main impact on the real-time network requirements and design comes from the system-level control. The frequencies of system-level sensing and actuation directly influence the required network performance and synchronization. chapter 3 contains the analysis and definition of these requirements concerning the control strategy.

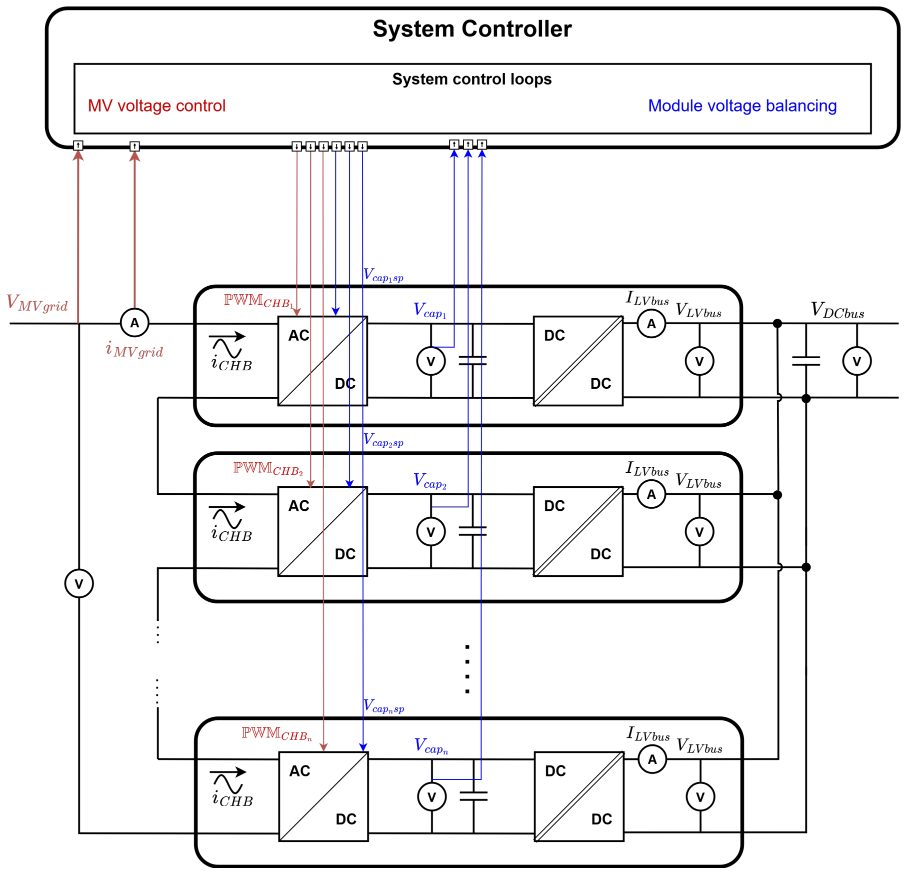


Figure 1.4: Simplified system-level control

1.1.3 Operation

The SST aims to replace a conventional MV transformer in the power grid. While the complex structure of an SST makes it considerably more expensive and introduces numerous additional points of failure, it could significantly stabilize the power grid.

The TU/e SST aims to convert a three-phase $10.5kV$ medium voltage grid (MV) to a three-phase $230V$ low voltage grid (LV). By design, an intermediate step transforms the $10.5kV$ AC voltage to a $700V$ DC voltage, i.e., a DC bus. Each phase of the MV is to be individually controlled by strings of 6 modules in series per phase. Each module takes a peak voltage of $1.6kV$, which is rectified by an H-bridge composed of high-power IGBTs. Next, the rectified voltage is transformed to $700V$ using a DC/DC converter known as a dual active bridge (DAB), isolating the MV and LV sides through a high-frequency transformer. Finally, the $700V$ DC voltage of the connected module outputs in parallel is inverted to a $230V$ AC voltage using one or several industrial off-the-shelf inverters.

The switching frequency of the cascaded H-bridge is $f_{CHB} = 1.6KHz$. The dual-active bridge, on the other hand, is $f_{sw} = 20KHz$. The switching patterns are generated using the module controllers.

1.1.4 Research constraints

The proposed project is part of ongoing research and development of an SST prototype. As a result, some hardware and software design decisions have been established and are treated as research constraints. The preselected concepts include hardware, software, and communication protocols. A summary of the preselected concepts is presented in Table 1.1.

Most importantly, the real-time industrial fieldbus EtherCAT has been selected as the network protocol, and relevant hardware has been acquired. This decision choice can be justified by EtherCAT's high performance, integrated synchronization mechanism, and low jitter, as discussed in section 2.1. Furthermore, the system controller's hardware has been preselected. The module controllers and the system sensor board controller are based on the F28379D controlCARD from Texas Instruments. The F28379D controlCARD is a development board for the F28379D MCU, which includes memory components and peripherals such as ADCs. In addition, EtherCAT Slave Controllers (ESC) piggyback boards (based on Beckhoff ET1100) are available and have been acquired for the F28379D controlCARD. The selected system controller is the Beckhoff CX2040 Embedded PC, including a two-port EtherCAT switch (Beckhoff EK1122). Since the SST modules include a $50kV$ isolation barrier, fibre optics must be selected as the transmission medium. Consequently, fibre optic communication requires media converters (Beckhoff CU1561).

Moreover, the envisioned TU/e SST prototype is highly integrated with Mathworks Simulink as a development environment and interface. Texas Instruments provides libraries and tools for programming the F28379D through code generation from Simulink models.

Table 1.1: A summary of preselected hardware and concepts for various SST parts.

SST part	**Preselected concept (constraints)**
Real-time network protocol	EtherCAT
System controller	Beckhoff CX2040 Embedded PC
Module controller	Texas Instruments F28379D controlCARD
System sensor board controller	Texas Instruments F28379D controlCARD
Real-time network medium	Fiber optics (Beckhoff CU1561 Media converters)
Development environment and interface	Mathworks Simulink
Control strategy	Hybrid control (module-level and system-level)

1.2 Real-time networks

The term real-time network describes any communication network used in real-time distributed systems. The 'real-time' distinction comes from the timeliness and data validity requirements imposed on the communication protocol. While general-purpose communication networks focus on throughput and bandwidth, real-time networks prioritize deterministic and predictable communication. The strictness of the real-time network requirements is contingent on the requirements of the underlying distributed real-time system. Typically, real-time systems are classified in literature as either soft, firm, or hard real-time, depending on the criticality of the timeliness requirement. In hard real-time systems, a missed or incorrect operation, e.g., because of a missed message, may lead to catastrophic events for the system and possibly its environment. On the other hand, soft or firm real-time systems will only suffer a quality downgrade but should be able to recover and continue regular operation. Therefore, real-time networks must comply with the requirements of the system they are servicing [8].

A real-time network is composed of a number of communicating nodes such as controllers, sensors, and actuators, connected through a physical link and communication interfaces, or layers, which translate the raw signals into meaningful data interpreted by the network protocol. A high-level abstracted example is shown in Figure 1.5. The role of the nodes in a real-time network is usually predetermined to ensure predictability in the flow of data. Typical real-time network performance requirements are bounded latency and jitter, system level synchronization, short communication cycles and cabling redundancy. However, various combinations of requirements exist for different applications. Real-time networks for industrial distributed control systems are commonly referred to as fieldbuses. Examples of industrial fieldbuses for real-time distributed systems include EtherCAT, CANopen, PROFINET, and SERCOS.

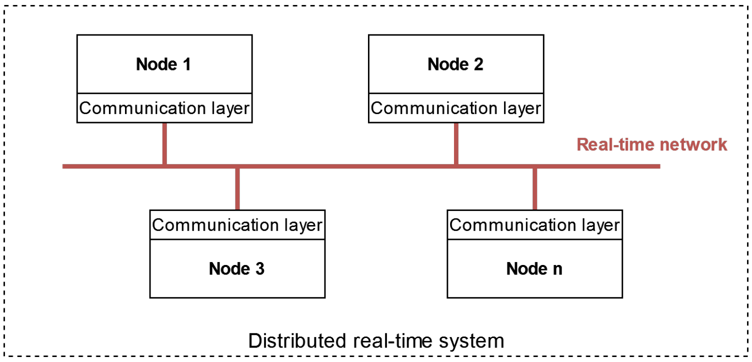


Figure 1.5: A real-time network connecting nodes within a distributed real-time system.

1.3 EtherCAT

EtherCAT is an Ethernet-based real-time industrial communication protocol initially developed by Beckhoff Automation and later standardized in IEC61158 as a suitable communication protocol for hard and soft real-time requirements in automation technologies. EtherCAT uses the physical layer and standardized frames of the Ethernet standard (IEEE 802.3), but it addresses additional concerns and requirements for industrial automation, such as rapid response times, synchronization, minimal device requirements, reduced overhead, and low cost.

Apart from sharing the physical and data layers, the EtherCAT protocol significantly deviates from Ethernet. The operating principle of EtherCAT is based on a master/slave[1] configuration. The EtherCAT master is the sole initiator of communication, which will actively send EtherCAT frames to the rest of the network. The EtherCAT slaves (nodes) are only forwarding the frame down- or upstream. Instead of sending separate frames to each EtherCAT node, only one EtherCAT frame is sent to the whole network simultaneously, containing data for each addressed node. The nodes are only operating on their respective parts of the frame (reading or writing the appropriate values) 'on the fly', which means that the local processing is decoupled from the communication to avoid potential delays and guarantee real-time capabilities. The decoupling is achieved through dedicated low-cost EtherCAT slave hardware. A simplified abstract version of the 'on the fly' communication method is shown in Figure 1.6. A common analogy to describe the EtherCAT operating principle is a moving train, where the EtherCAT slaves serve as 'stations' to load and unload passengers (data). The propagation delay introduced at each slave is typically less than a microsecond.

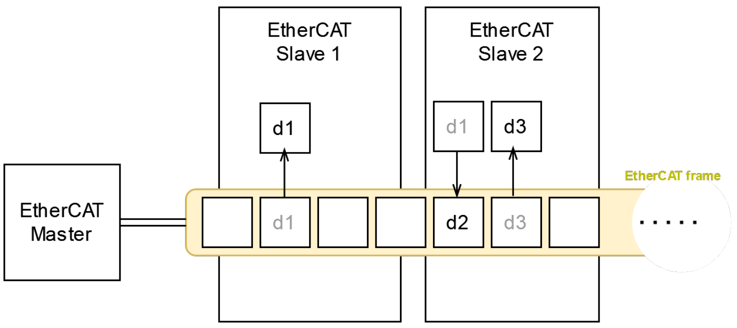


Figure 1.6: EtherCAT 'on the fly' processing (simplified).

The EtherCAT protocol embeds its data into the standardized Ethernet frames by conserving the Ethernet headers and inserting its own EtherCAT frame into the Ethernet data portion of

[1]This unfortunate terminology is pervasively used in EtherCAT documentation and general computing but shall be replaced with *system/module* respectively, where possible in later chapters.

the frame, as shown in Figure 1.7. The EtherCAT master configures each device's payload during startup, which allows variable data exchange with each EtherCAT slave, ranging from 1 bit up to a almost 2 kilobytes of data.

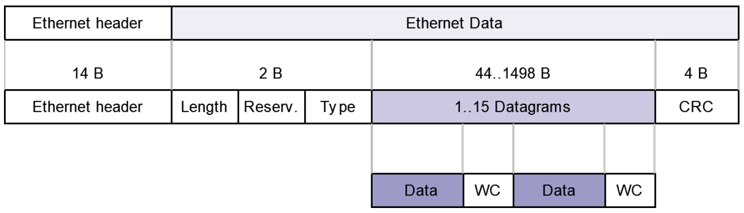


Figure 1.7: Structure of an Ethernet frame with EtherCAT protocol data.

One of EtherCAT's advantages is its flexibility regarding the network topology. It supports a range of typical topologies such as line, tree, star, ring, mesh, and hybrid combinations. Depending on the chosen topology, additional hardware such as I/O switches might be required. An additional advantage is an ability for uninterrupted operation even if part of the network, or a particular node, is disconnected. EtherCAT supports up to 65535 devices connected to one segment, meaning that a network expansion is almost unlimited.

EtherCAT includes a high-precision synchronization mechanism known as distributed clocks (DC). High-precision synchronization might be required in applications where a simultaneous action is required, such as coordinating servo axes movements for motion control. The principle is based on using multiple local clocks within the EtherCAT slaves and a chosen reference clock (typically belonging to the first node in the segment). The time of the reference clock is transmitted to all other nodes using the EtherCAT frame so that the EtherCAT slaves can adjust their local clocks according to the reference time. The EtherCAT master measures the propagation delay for each node and embeds the information into the frame, indicating how the EtherCAT slave should adjust their local clock. The synchronization frames are sent periodically to compensate for jitter.

The discussed EtherCAT features are supported by dedicated hardware for EtherCAT slaves, known as EtherCAT Slave Controllers (ESCs). The hardware is interfaced with a host microcontroller which indirectly participates in the EtherCAT network. The host microcontroller periodically accesses the EtherCAT data from the ESC's memory, usually locally synchronized through interrupts. Apart from an 'incoming data' interrupt, the ESC includes two more interrupt lines to support the DC mechanism. The EtherCAT slaves can be configured to several modes of synchronization using a combination of the available interrupts generated by the ESC, such as free-run, synchronous with SM-events (receipt of frame), synchronous with SYNC-events (distributed clocks).

EtherCAT's performance is considered exceptional compared to other industrial real-time network protocols [9]. For typical applications of up to 50 nodes, cycle times can be lower than 250 μs, with bounded jitter in the range of few hundreds of nanoseconds. A common industry general-principle for the maximum EtherCAT (100 Mbit) frequency is 8 KHz, or 125 μs cycle time.

Due to its performance, flexibility, and robustness, EtherCAT is widely used in the automation industry for various products ranging from robotics, assembly systems, offshore applications, automated guided vehicles, power plants, and many others.

Chapter 2

Research question definition

2.1 State-of-the-art analysis

Solid-state transformers are extensively investigated and researched as a promising technology expected to be essential in future smart grids, distributed generation sources, and modern traction systems. In addition to their primary role as transformers offering galvanic isolation between two AC systems, SSTs offer many advantages over conventional transformers: voltage stabilization and regulation, reactive power compensation, active harmonic filtering, controllability, isolation, and fault protection, and increased power quality [10]. Many SST topologies have been proposed in the literature [11][12], commonly composed of input, isolation, and output stages. A common characteristic found in many proposed variations is the modularity [13], as currently available IGBT or SiC switches are not distribution voltage rated. Additionally, modularity is desired as it increases the reliability of SSTs towards the desired 100% uptime within the grid, and allows expansion of the SST to higher power ratings. The most popular input rectifier stage topologies used in SSTs are cascaded H-bridges (CHBs) and multilevel modular converters (MMCs), both inherently modular. A comparative study of CHB and MMC topologies has been conducted in [14].

However, a proper control strategy must be employed to prevent unbalanced modular voltages, overvoltages, and overcurrents which can stress the power electronic switches and cause a break down of the SST system. In [15], a systematic control strategy for a three-phase modular cascaded SST is proposed and verified, showing that the problems introduced by modularity can be avoided using a suitable control strategy. Similarly, in [16], a current control strategy for a star-connected cascaded H-bridge (CHB) based SST is proposed, proving that voltage and power imbalances between phases or modules can be mitigated using control strategies.

Both centralized and distributed control schemes have been applied in CHB and MMC-based MV applications. In [17], the authors argue that distributed control is superior to centralized control for CHB or MMC-based applications such as SSTs due to higher computing and fast real-time network demands. Additionally, the use of local processors in submodules allows for full modularity and less wiring. Moreover, in [18] the control design of an MMC-based SST is presented, showing the advantages of distributed as opposed to centralized control. In a complex application such as the SST, a centralized control would mean that a single controller must acquire and act upon all power electronic components using numerous AD converters while dealing with various switching and acquisition frequencies in real-time. Therefore, the literature suggests that distributed control is more suitable for SST applications. However, the control schemes in SSTs may benefit further from combining the two approaches into a hybrid control scheme, where part of the control is realized by distributed module controllers, and part is done on a system level by a central controller. The control architecture is typically in a master-slave configuration, but the amount of control delegation may vary between schemes. For example, [17] proposes a coordinate control system organized as a minimum data exchange master-slave configuration, which would allow higher control frequencies without the need for complex network protocols and higher hardware costs.

A hybrid control system can only be realized by a real-time communication network that provides sufficient bandwidth, speed, and synchronization. Many modular HVDC applications use a form of hybrid control which requires real-time communication due to the time-sensitive control tasks [19, 20]. In [21], a dedicated hybrid communication topology and protocol is proposed for such applications, which would allow the use of low-speed real-time communication. However, the most popular high-speed real-time communication protocol used in HVDC applications is the high-speed industrial protocol EtherCAT, which is successfully used for control in [22, 23, 24, 25]. EtherCAT is based on a master-slave communication model. A thorough comparison of several high-speed communication networks for power electronics converters is conducted in [26], showing that EtherCAT has significant advantages for power electronics control applications. In addition, EtherCAT is equipped with a synchronization protocol known as the Distributed Clocks mechanism [27] and communication cable fault tolerance [20], suitable for the demands of modular distributed control.

The existing SST literature is mainly concerned with electrical topologies and their implementation using control strategies wherever necessary. However, little research investigates the requirements and aspects of real-time networks used to enable communication and control. In [28], the authors have highlighted the lack of discussion and research in communication, specifically for (hybrid) distributed control of MMCs. They discuss the transmission media, synchronization accuracy, and network topology when designing a distributed control architecture for MMCs and verify their importance using simulations. While the concept investigated in the paper is similar for SSTs, no specific and thorough investigation of real-time network requirements for control and communication in SSTs exists, to the best of our knowledge.

Therefore, this work contributes to the gap in existing research concerning requirements of real-time networks in solid-state transformers, recognizing the importance of SSTs in the future of electric grids.

2.2 Problem description

While a good amount of related work focuses on control and power balance between modules in SSTs or similar power electronic converters, there is only a limited discussion on the communication aspects. Therefore, the lack of clearly defined requirements for real-time networks in solid-state transformers should be investigated, which leads to the following research question.

How to design, implement, and verify real-time networks for solid-state transformers?

Furthermore, the TU/e SST prototype aims to enable and facilitate future SST research through a uniform commercial development environment, Simulink. Therefore, communication through the real-time network should also be controllable through Simulink. The SST prototype, which will become fully operational through a real-time network, can be used to verify and validate the real-time network and its requirements.

The research problem's methodology is as follows: the real-time network requirements are first investigated mainly concerning the predefined SST requirements such as control strategy, communication protocol, and Simulink integration. Next, a communication model is defined to abstract and facilitate design decisions for each SST controller type (module, system, and system sensor board). Finally, tests are introduced and executed to experimentally verify quantifiable requirements on a down-scaled SST prototype (three instead of the original 18-module design). Finally, the results of the tests are analyzed to validate whether the proposed design satisfies the defined requirements.

Chapter 3

Determining requirements

3.1 Introduction

This chapter investigates the requirements for real-time networks used in modular Solid-state transformer applications. Specifically, the requirements will focus on the SST developed at the TU/e; however, they can be generalized.

3.2 Functional requirements

The functional requirements of the real-time network for the SST should specify criteria that enable the correct function of the SST. The real-time network cannot be used in the SST without satisfying the functional requirements, as it would not enable proper operation. This section identifies and discusses the following requirements: performance, data frames, synchronization, transmission medium, and network topology.

3.2.1 Performance

The real-time network performance requirements for Solid-state transformers can be derived from the control architecture and the higher harmonic compensation requirements. SST control architectures can generally be centralized, distributed, or hybrid. In a centralized architecture, a system controller is responsible for all sensing, control, and actuation, which would require significant bandwidth and communication frequency as the size and complexity of the system increase. On the other hand, a distributed control architecture omits a central system controller in favour of autonomous local controllers distributed throughout the system. As mentioned in section 2.1, many modular SSTs utilize a hybrid control architecture, in which the system has a semi-autonomous controller for each module and a system controller for high-level control and synchronization.

The TU/e SST too uses a hybrid control architecture. To derive the real-time network performance requirements, the local (module-level) and central (system-level) control responsibilities were investigated and separated, as discussed in section 1.1.2. The module-level and system-level control diagrams are presented in Figures 1.3 and 1.4, respectively.

The real-time network studied in this work connects the system controller with its system sensor board controller and all module controllers. Since the module controller directly interfaces the sensing of electrical signals and the actuation of switches, its performance is considered independent of the real-time network's performance. However, the module controller must handle the communication rate of the system-level real-time network, meaning that the local sensing and actuation should not block the receipt or delivery of central messages, i.e., not impose unbounded jitter on the system-level communication.

The system controller is responsible for sensing the grid characteristics and determining the required voltage for each module. The Dutch Grid code describes harmonic distortion up to the 40th harmonic (NEN-EN50160 standard). However, compensating for distortion up to the 50th harmonic introduces greater applicability. With the fundamental frequency being 50Hz, the limits for the amplitudes of higher harmonics can be described up to 2500 Hz. According to the sampling theorem [29], the MV grid voltage must be sampled and actuated upon at a minimum of 5000Hz to theoretically compensate for the 50th harmonic. However, a factor of 10 is recommended as a rule of thumb to sufficiently sample a sine wave, meaning that the sampling rate should be $25kHz$. The proposed switching frequency of the cascaded H-bridge in the TU/e SST is $1.6kHz$. However, by combining six modules per phase using interleaved switching, the actual frequency would become six times $1.6kHz$, or $9.6kHz$. Additionally, through unipolar switching instead of bipolar, the frequency is doubled to $19.2kHz$, closer to the pragmatic $25kHz$ for compensating the 50th harmonic.

Therefore, the minimum communication frequency should be at least twice the frequency of the 50th harmonic, i.e., $5kHz$. *In other words, the cycle time for the fully connected real-time network should be less than* $250\mu s$

Figure 3.1 shows the 50th harmonic with respect to the fundamental frequency. It can be seen that the base CHB frequency of $1.6kHz$ is not enough to compensate for the 50th harmonic alone (without unipolar interleaved switching), but that the high harmonic could be theoretically sampled and communicated within a frequency of $8kHz$.

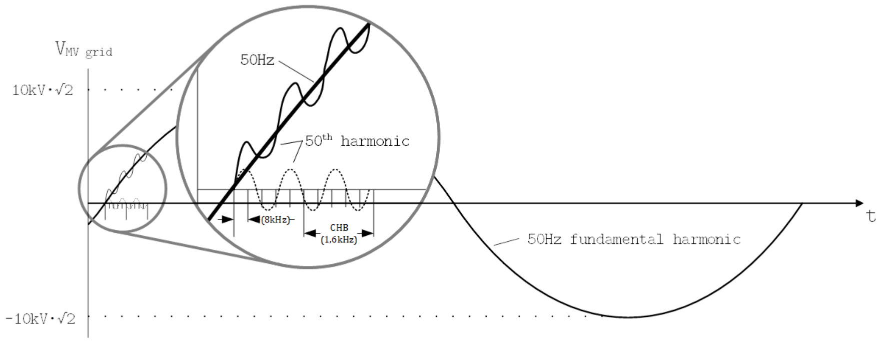


Figure 3.1: Compensating for higher harmonics

As mentioned in section 1.1.4, EtherCAT is the preselected choice of communication protocol and is treated as a research constraint in this work. Furthermore, EtherCAT is known to be capable of cycle times as low as $100\mu s$, which means it could satisfy the communication performance requirements of the TU/e SST.

3.2.2 Data frame

The control data exchange through the real-time network must be considered to define the communication data frame for the TU/e SST. Additionally, the data frame should fit spare auxiliary data to be transferred if necessary and as a form of readiness for future development or control loop expansion.

As discussed in section 1.1.2, the SST system controller sends and receives control data from two system-level control loops, with the possibility of further expansion since the SST is an ongoing project. All system-level communication is to be transmitted through the real-time network. The identified control data is presented in Table 3.1, including its origin, destination, representation and quantity.

The minimum required digital representation is 16-bit signed integers (INT16), as no value may theoretically exceed the $[-32768, 32767]$ range. It can be argued that floating point numbers would improve the precision and performance of the control loops. However, depending on the precision, floating point numbers require 32 or 64 bits, making the potential frame two to four times larger. Moreover, floating point calculations are generally slower than integer calculations. Using an INT16 representation for the given data in Table 3.1 would require a data frame size of 128 bytes. The 16-bit integers could be used to represent voltage directly, e.g., a decimal value of 256 represents 256 V, or the range of values could be used as quantization levels. For a 16-bit integer, where the MSB is used as a sign, there are $2^{16} - 1 = 65535$ quantization levels, with a quantization step of $Q_{step} = (v_{max} - v_{min})/2^{16}$, where v_{max} and v_{min} are the largest and smallest possible value, respectively. Representing analog values as digitized integer requires rounding off operations, which introduces a rounding error of up to:

$$Maximum rounding error = \frac{v_{max} - v_{min}}{2(2^n)} \tag{3.1}$$

For example, using the full possible decimal range representable by a 16-bit integer, where the LSB represents the quantization step of $Q_{step} = 1V$, the maximum introduced rounding error is $0.5V$. However, the error could be reduced if a smaller range of values is represented instead, i.e., the LSB represents a smaller value increment than 1. For example, by using the maximum theoretical voltage of $v_{max} = 10.5kV$, the maximum rounding error would be $0.16V$.

Given that the ADCs used in the SST have a 16-bit resolution, an inherent quantization error is not improved by using larger data representations such as floating point numbers or larger integers.

Table 3.1: The communicated control data through the real-time network.

Data	Origin	Destination	Representation	Quantity	Description
V_{MV}	System sensor board	System controller	INT16	1	The medium voltage on the grid side of the filter
$V_{MV_{converter}}$	System sensor board	System controller	INT16	1	The medium voltage on the converter side of the filter
I_{MV}	System sensor board	System controller	INT16	1	The current of the medium voltage grid
$V_{DC_{bus}}$	System sensor board	System controller	INT16	1	The voltage of the DC-bus
V_{LV_A}	System sensor board	System controller	INT16	1	The voltage of phase A on the LV-side
V_{LV_B}	System sensor board	System controller	INT16	1	The voltage of phase B on the LV-side
V_{LV_C}	System sensor board	System controller	INT16	1	The voltage of phase C on the LV-side
I_{LV_A}	System sensor board	System controller	INT16	1	The current of phase A on the LV-side
I_{LV_B}	System sensor board	System controller	INT16	1	The current of phase B on the LV-side
I_{LV_C}	System sensor board	System controller	INT16	1	The current of phase C on the LV-side
$V_{cap_{sp}n}$	System controller	Modules	INT16	18	The capacitor set point for module n
V_{cap_n}	Modules	System controller	INT16	18	The actual capacitor voltage of module n
PWM_{level_n}	System controller	Modules	INT16	18	The PWM level for the CHB of module n

Therefore, for the proposed SST design at the time of writing, the data frame size must be a minimum of 128 bytes.

EtherCAT's maximum data frame size of 1486 bytes far exceeds this requirement. However, increasing the data has a negative effect on EtherCAT's cycle time, i.e., the performance.

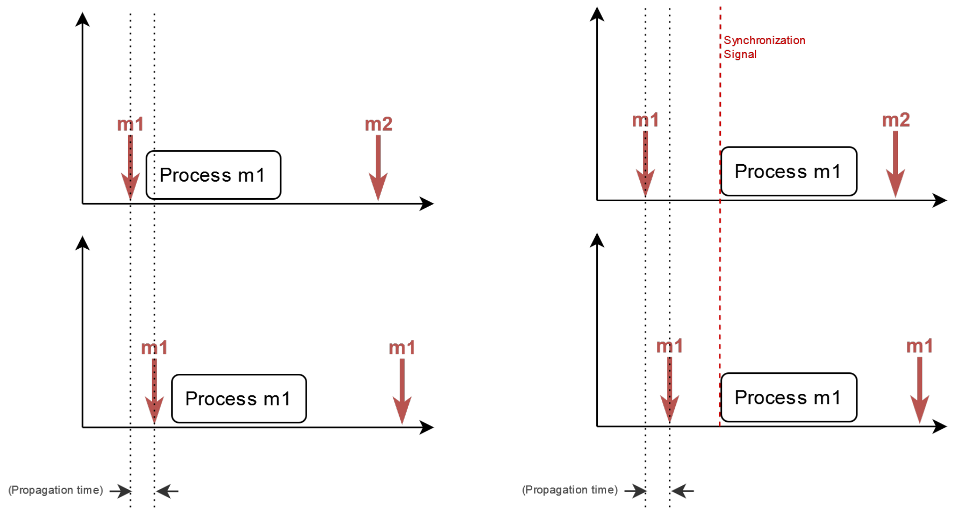
3.2.3 Synchronization

The system-level control within the SST includes MV voltage control and module MV capacitor balancing, which require a level of synchronization between the received message and the module response.

Locally, the modules control the input current, capacitor voltage levels, and power transfer. However, the system controller supplements the local control loops by ensuring a sinusoidal input current and correct capacitor voltage set points, as discussed in section 1.1.2. The control data transmitted from the system controller consists of the PWM level for the module's CHB and a voltage and/or power set point. Therefore, the module should timely (within the same communication cycle) provide the required control data such as actual capacitor voltage back to the system controller.

As the system controller regulates and corrects the module-level control loops, synchronization should ensure that each module has used the incoming data and sent its data back within the same communication cycle. However, since the SST operates on a 50Hz fundamental frequency and up to 2500Hz harmonics, sub-microsecond synchronization might be unnecessary.

The inherent propagation time of data with communication protocols such as EtherCAT means that to ensure synchronization, the communication initiator (i.e., the EtherCAT master for EtherCAT) should simultaneously broadcast a signal that all nodes have received the messages. Upon the receipt of the synchronization signal, the nodes could simultaneously, i.e., with insignificant jitter, start processing the data. Figure 3.2 shows how the propagation time affects the data processing start times without synchronization (left image) and how this can be mitigated using a synchronization signal (right image). For example, the receipt of the message 'm1' cannot be simultaneous due to inherent propagation time, but the data processing task 'Process m1' can be synchronized using an external signal.



(a) Without process synchronization. (b) With process synchronization signal.

Figure 3.2: Comparison of synchronization modes for message data processing.

In the case of the SST, the data processing task is associated with the local module control loops. So at a module level, the control loop start must be synchronized to the receipt of the control data. **As long as the control loops' execution times fit within the communication cycle, the SST can be considered appropriately synchronized in both scenarios presented in Figure 3.2** since all modules would process the data from the same communication cycle and consequently would be ready for the next communication cycle. Therefore, the proposed synchronization requirement in this work is as follows:

The receipt of the control data should be synchronized with the control loop start on a module level, whereas on a system level, abstracting the propagation time through synchronization is unnecessary.

EtherCAT's distributed clocks mechanism supports the scenario presented in Figure 3.2b, as discussed in section 1.3, but at the cost of increased cycle times.

3.2.4 Transmission medium

The TU/e SST is designed to comply with the IEC60076-3 standard, which governs the requirements for conventional power transformers in electricity grids. One of the tests necessitated by the standard is the AV - Applied voltage test, which verifies whether a power transformer can withstand alternating voltage or fundamentally verify the galvanic isolation between the primary and secondary sides of a transformer. From the perspective of the SST, the AV test would verify the galvanic isolation between the power electronics on the MV and the LV sides. To achieve isolation, the SST architecture envisions fibre optics as a communication bridge between the LV and the MV sides. The isolation requirement also translates to the real-time network, as each SST module has a $50kV$ isolation barrier.

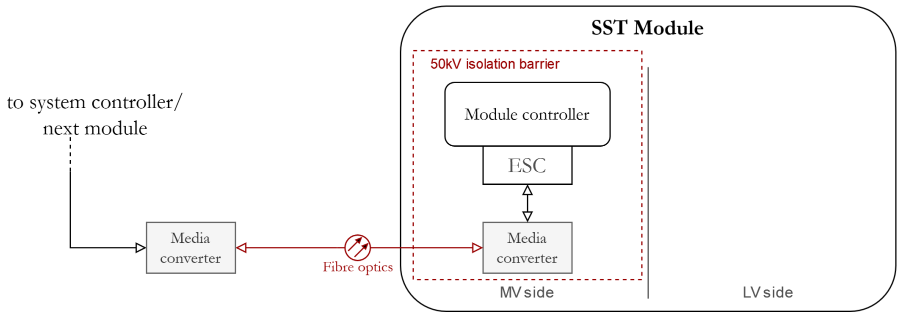


Figure 3.3: Media converters enable communication through the 50kV isolation barrier.

Therefore, the real-time network must use optical fiber as a physical medium for communication.

3.2.5 Network topology

The choice of network topology is dependant on multiple factors, such as propagation delay, redundancy, hardware requirement, etc. It can have a significant impact on the scalability and fault tolerance of the system, which is why this requirement is interdependent. EtherCAT networks support flexible topologies and configurations such as ring, line, and star. However, despite the physical connection type, EtherCAT logically uses a ring topology. For example, to create a star connection that is traditionally a popular Ethernet topology, specialized EtherCAT junctions are required to reroute the traveling packet to the next node in the 'logical ring'. This logical transformation is depicted in Figure 3.4.

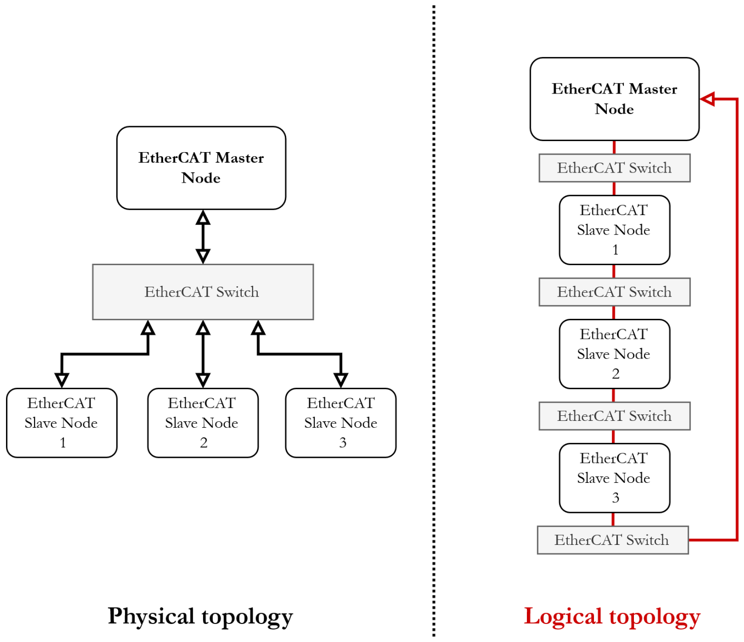


Figure 3.4: Physical versus logical topology of EtherCAT

Therefore, when choosing a physical topology, the logical connection must be considered to calculate the expected propagation delays introduced by the topology.

Considering EtherCAT's logical communication, an analysis is conducted for three popular topologies: star, ring, and line. The topologies are compared by three criteria: propagation delay, fault tolerance, and cabling and hardware requirements. Propagation delay is calculated with the general formula:

$$\tau_{prop} = t_{fw_M} + \sum_{network} t_{cable} + \sum_{network} t_{d_{MC}} + \sum_{network} t_{fw_S} \tag{3.2}$$

The terms in Equation 3.2 are summarized in Table 3.2. If an EtherCAT switch (junction) is necessary for the topology, an additional switch propagation delay $t_{fw_{SW}}$ is introduced. Furthermore, if a node is the last in a physical connection, it has to close its Tx port and reroute the packet to its predecessor, which introduces a marginally larger delay than a regular node, denoted as $t_{fw_S end}$.

Table 3.2: Summary of equation terms for propagation delay calculation

Equation term	Description
t_{fw_M}	Forwarding propagation delay introduced by the EtherCAT master node
t_{fw_S}	Forwarding propagation delay introduced by the EtherCAT slave nodes
t_{cable}	Cable propagation delay
$t_{d_{MC}}$	Media converter (fibre optics/copper) forwarding delay
$t_{fw_{SW}}$	EtherCAT switch forwarding delay
$t_{fw_S end}$	Forwarding propagation delay of the last EtherCAT slave node in the network

Fault tolerance in EtherCAT is possible as long as there exists a (redundant) physical connection to the rest of the nodes. A topology is single-fault tolerant if the failure of one node does not disrupt communication with other nodes. Cabling and hardware requirements suggest the necessity for additional equipment to enable the topology, introducing additional costs and points of failure. For the SST specific fault tolerance requirement, see section 3.3.2. A comparison of the three topologies is presented in Table 3.3, where n represents the number of EtherCAT slave nodes. From the comparison in Table 3.3, it can be concluded that the ring topology introduces the slightest propagation delay to the overall network performance, it requires less hardware and cabling than a star topology, and it offers a single-fault tolerance. On the other hand, the star topology is exclusively superior if higher redundancy and therefore fault tolerance is required.

Table 3.3: Comparison of EtherCAT topologies including media converters

		Propagation delay	**Fault tolerance**	**Cabling and hardware**
Topology	**Line**	$\tau_{line} = t_{fw_M} + 2n \cdot t_{d_{MC}} + 2 \cdot 3n \cdot t_{cable} + (n-1) \cdot t_{fw_S} + t_{fw_S end}$	no guarantees	minimal
	Ring	$\tau_{ring} = t_{fw_M} + \frac{(n+1)}{2} \cdot t_{d_{MC}} + (3n+3) \cdot t_{cable} + n \cdot t_{fw_S}$	single fault tolerant	+ 2 media converters, 3 cables
	Star	$\tau_{star} = t_{fw_M} + 2n \cdot t_{d_{MC}} + 2 \cdot (3n+1) \cdot t_{cable} + n \cdot t_{fw_S end} + \tau_{fw_S}$	n-fault tolerant	+ dedicated EtherCAT switch(es), cable(s)

The requirement for the network topology significantly depends on the envisioned fault tolerance of the system and the network performance, as each topology has potential strengths and weaknesses.

Therefore, no topology requirement could be defined or generalized for SST RTNs without looking at performance, hardware, and fault tolerance requirements.

3.3 Non-functional requirements

The non-functional requirements, as opposed to the functional requirements, specify criteria regarding the quality of operation of the SST system. While there are many possibilities for judging

Table 3.4: Typical approximate EtherCAT times and bandwidth.

EtherCAT parameter	Typical value
t_{p_M}	8 μs
t_{fw_M}	9 μs
t_{fw_S}	0.3 μs
$t_{d_{MC}}$	1 μs
BW_{EC}	100 $Mbit/s$

the operation of the SST, only the seemingly most important non-functional requirements are investigated in this section.

3.3.1 Scalability

The modular design of the SST allows for future-proofing the SST for higher power ratings. The functional requirements so far have been investigated and defined for the current SST design consisting of 18 modules. However, the potential and likely addition of modules to increase power rating should be analysed from the aspect of the real-time network, i.e., the impact on performance, bandwidth, and synchronization. Moreover, the investigation of scalability should also encompass the increase of bandwidth in the network due to new data for improved or additional control loops and various safety information required by the system controller.

As discussed in section 3.2.2, the system-level control loops dictate the amount of data that needs to be transmitted through the real-time network each cycle. However, as the TU/e SST is an ongoing project, new data might be introduced into the system to extend the control loops or introduce new ones. As seen in Table 3.1, while most of the distinct variables originate from the system sensor board, 75% of the data frame is dedicated to the communication between the system controller and the modules. Using a data frame containing the variables listed in Table 3.1, the introduction of each new module would increase the data frame size by 4.68%, i.e., by 6 bytes. However, the amount of data in Table 3.1 is conservative, as it is only the minimum requirement. Therefore, auxiliary data should be considered within the communication frame between the system controller and the modules to enable scalability and future-proofing, especially for enabling future research which might require additional data to be transmitted. Introducing auxiliary data means that for every additional variable (considering 18 modules), the frame size would increase by 28.125% from the original.

Two limitations must be considered for scalability: the maximum EtherCAT frame size and the cycle time. The maximum EtherCAT frame size is 1486 bytes. It is still possible to send larger amounts of data, but EtherCAT will split it into multiple frames, which must be sent in different cycles. The EtherCAT cycle time depends on the bandwidth, frame size, the number and type of devices in the network, the frame processing times, and the physical propagation time. Additionally, as discussed in section 3.2.4, the real-time network's required use of fibre optics and, consequently, media converters will increase the propagation delay due to the latter.

An approximation of the EtherCAT cycle time T_{EC_n} for n devices, i.e., modules, using a frame with size S_{frame} can be calculated using the following equation:

$$T_{EC_n} = t_{p_M} + t_{fw_M} + 2n(t_{fw_S} + t_{d_{MC}}) + \frac{S_{frame}}{BW_{EC}} \tag{3.3}$$

where t_{p_M} is the EtherCAT master application processing time, t_{fw_M} is the forward delay from the EtherCAT master controller to the EtherCAT hardware, t_{fw_S} is the forwarding delay of the EtherCAT slave devices (assuming they are identical), $t_{d_{MC}}$ is the media converters delay, and BW_{EC} is the bandwidth of the EtherCAT network expressed in $Mbit/s$.

With 18 modules and the typical approximate values presented in Table 3.4, the calculated EtherCAT cycle time is presented in Figure 3.5 as a function of the frame size. As can be seen on the graph, for a pragmatic $8kHz$ frequency, which is equivalent to 125 μs cycle time, the frame size should not exceed 765 bytes. A graph which shows the frame size growth by scaling the number of system-to-module (or vice versa) variables is depicted in Figure 3.6. It shows that for 18 modules, the data frame presented in Table 3.1 can be extended with up to 17 system-to-module 16-bit integer variables, 8 single precision floating point variables, or 4 double precision floating point variables.

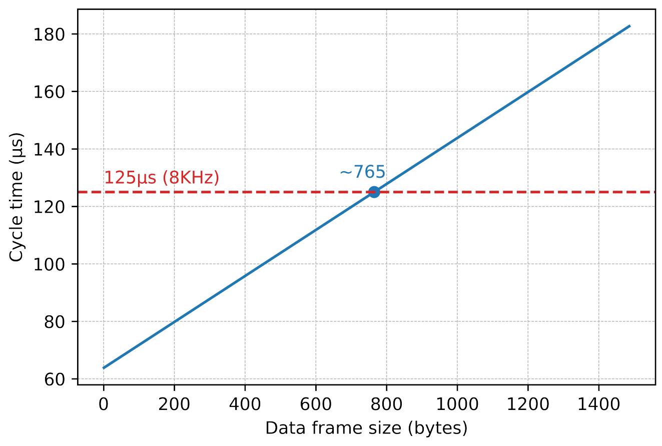


Figure 3.5: EtherCAT's cycle time with respect to the data frame size.

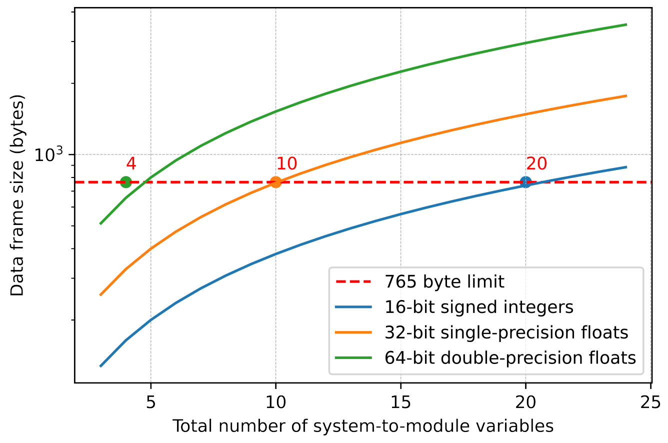


Figure 3.6: EtherCAT's data frame growth with respect to the increase of variables (18 modules).

With the minimum required frame size of 128 bytes, according to section 3.2.2, Figure 3.7 shows the effect of module scaling on the cycle time. An EtherCAT real-time network that complies with the functional requirements in section 3.2 can scale up to 37 modules, or slightly more than 100%.

The scalability requirement for the SST real-time network can be defined as follows: the real-time network should support at least 100% data increase and at least 20% increase in the number of modules.

3.3.2 Fault tolerance

The complexity of the TU/e SST and its numerous electronic components introduce many possibilities for failure, which could be mitigated through proper fault management. The modular

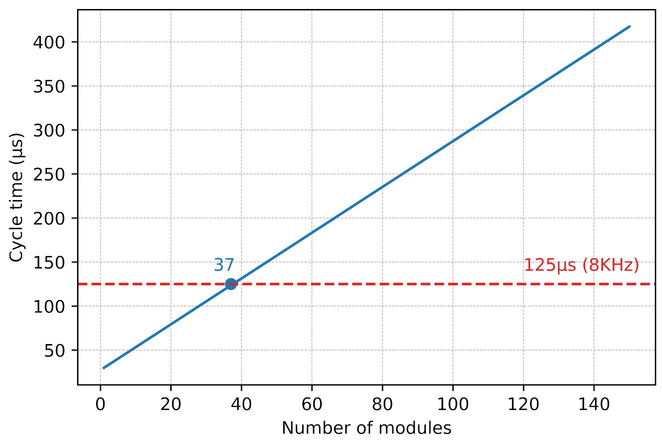


Figure 3.7: EtherCAT's cycle time growth with respect to the increase of modules (128 byte frame).

design of the SST allows for the MV grid voltage and power to be divided among the participating modules. Therefore, a failure of a module within a phase would mean that the remaining modules would incur both larger voltages and power. Since the SST is over-dimensioned by a factor of two to comply with the IEC60076-3 standard, the voltage incurred by limited module failure can be tolerated. However, the power must be redistributed over the remaining modules in a module failure scenario. The power redistribution could bring the SST to a thermal criticality depending on the number of failed modules and the power output of the SST. While conventional transformers are typically oversized in practical applications for both overloading and future-proofing (a combined factor of 4), the current cost and lifetime of SSTs render such over-dimensioning infeasible. Therefore, the failure should be limited to a few modules simultaneously.

A pragmatic and safe solution would be to design the system as a single-fault tolerant, such that a failure would lead to a safe system shut down to prevent further damage. However, considering that conventional transformers in electrical grids are very reliable with minimal downtime, the much more complex SST should also be able to deliver power even during partial failure to increase competitiveness. Therefore, the real-time network should offer n-node failure tolerance. The main factor in RTN fault tolerance is redundancy, i.e., how the nodes are connected. Therefore, choosing a suitable network topology and protocol is crucial to fault tolerance.

As discussed in the section 3.2.5, EtherCAT supports various network topologies, but only the star-topology is n-fault tolerant. While due to the logical operation of EtherCAT, a star-topology would accrue the highest propagation delay, it exclusively offers the amount of redundancy and tolerance required by the SST.

Therefore, the real-time network for the SST must be n-fault tolerant of potential failure of several modules.

3.3.3 Ease of use

The TU/e SST project is both a prototype and a research platform for further SST-related research. Even though it does not directly influence functionality, ease of use is an essential requirement for this project. Specifically, the SST project requires complete integration with Simulink, a model-based graphical programming and simulation environment. While it might seem unrelated to the real-time network, the Simulink integration has significant consequences on the design choices and implementation of the network.

The integration with Simulink would allow the system to be reprogrammed graphically through Simulink code generation available for the specific hardware. In addition, the Simulink integration allows researchers to focus on the design of algorithms rather than their low-level implementation. The real-time network would require an interface to send and receive messages in Simulink, which is not a typical use of such networks. In general, real-time protocols such as EtherCAT are not intuitive to be used without low-level development.

The real-time network should fully support the integration with Simulink as a development environment for the SST.

3.4 Summary

This chapter identifies and describes the real-time network's functional and non-functional requirements to comply with the TU/e SST design. The functional requirements relate to the criteria necessary for the proper network function and, consequently, the SST. The functional requirements include performance, data frame definition, synchronisation, transmission medium, and network topology, for which it has been determined:

- The performance requirement for the SST real-time network is determined to be a minimum communication rate of $5kHz$, which could be satisfied using EtherCAT's typical $8kHz$.
- The minimum data frame size requirement to satisfy the control data communication using 16-bit signed integer representation is 128 bytes.
- The receipt of control data must be synchronized with the control loops on a module level, whereas on a system level, each module must receive and process the data from an incoming frame within the same communication cycle.
- The SST real-time network must use fibre optics as a communication transmission medium to satisfy SST isolation requirements.
- The choice of network topology depends on the desired fault tolerance, hardware requirements, and propagation overhead. Therefore, no strict topology requirement has been defined.

On the other hand, the non-functional requirements describe the criteria for the system's quality of operation. The identified and described requirements relate to the scalability, fault tolerance, and ease of use of the SST w.r.t. the real-time network, for which the following conclusions have been made:

- The SST real-time network must support the scaling of both data and the number of modules by at least 100% and 20%, respectively. The network can support up to 37 modules using the current control design and data frame requirement. Alternatively, using the proposed 18 modules, the maximum data frame size can be 765 bytes.
- The SST, and consequently the SST real-time network, must be n-fault tolerant of potential failure of several modules. This requirement can be satisfied by using an EtherCAT star topology.
- The real-time network should fully integrate within the Simulink development environment for the SST.

Chapter 4

Design and implementation

4.1 Introduction

The real-time network is an essential part of the overall SST system, so its design must be incorporated into the overall SST system design. It is impractical to completely isolate the network as a distinct entity without considering the communication model of the SST. Moreover, as determined in the requirements in chapter 3, the real-time network must support and enable Simulink integration.

This chapter defines the communication model and explains how the real-time network is integrated into the overall system design according to the defined requirements, including implementing the communication layer which supports the communication protocol, EtherCAT.

4.2 Communication model

To correctly analyse whether the EtherCAT-based real-time network can satisfy the requirements, the communication model of the system must be defined.

Three levels of communication can be defined within the SST: module-level, SSB-level, and system-level communication. The module-level communication is between a module controller and local sensors and actuators on the SST module. Similarly, the SSB-level communication is between the system sensor board controller and the sensors on the SSB through external ADCs. Finally, the system-level communication is the highest level of communication between the system controller, the SSB controller, and all of the module controllers. The communication model described in this section is based on system-level communication as the main focus of this work.

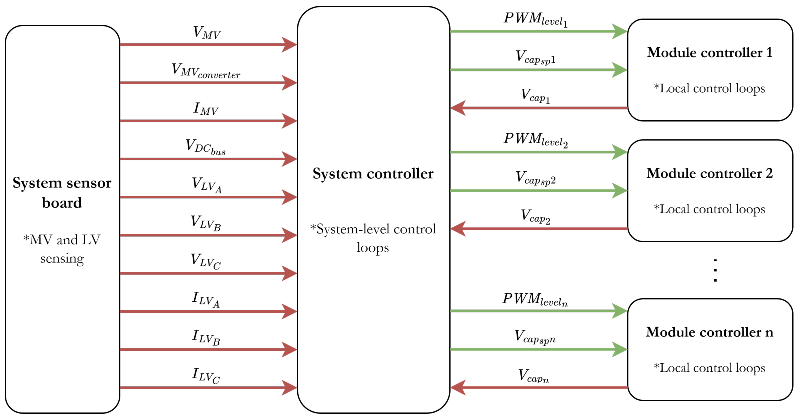


Figure 4.1: Communication data exchange block diagram.

Figure 4.1 depicts a block diagram of the communication data exchange on a system level, as defined in sections 1.1.2 and 3.2.2. As can be seen, apart from the system-level communication, each controller executes one or several tasks, e.g., the system-level control loops on the system controller.

For each controller, we can define a periodic communication task $Task_{comm}$ which processes the system-level communication, i.e., reads and writes data to the communication bus. The communication $Task_{comm}$ has a period of T_{comm}, and an execution time e_{comm}. The period T_{comm} must be equal to the system-level communication period $T_{network}$, i.e., $T_{comm} = T_{network} = 125\mu s$ for an 8KHz communication frequency, on all controllers. The execution times e_{comm} may vary from controller to controller, but it must satisfy the following:

$$e_{comm} \leq T_{comm} - \sum_{i=0}^{k} e_i \tag{4.1}$$

where $\sum_{i=0}^{k} e_i$ is the sum of the execution times of all other local tasks on the controller which have a higher priority than $Task_{comm}$. This strict requirement ensures that the communication task will finish execution before its next iteration, i.e., the task will meet its implicit deadline $D_{comm} = T_{comm}$.

To analyse timing behavior and scheduling at a system level, the relative start and end times of the total execution cycle for each module n are defined as $t_{n_{start}}$, and $t_{n_{end}}$, respectively. Specifically, $t_{n_{start}}$ represents the time the module n receives a message from the RTN (relative to the start of the frame), and consequently its $Task_{comm}$ begins to execute, which globally can be denoted as $Task_{comm_n}$. The end time of all remaining processing tasks within a module is denoted with $t_{n_{end}}$. The following condition must be true to ensure that each module will receive and process the message from the same RTN communication cycle:

$$t_{n_{end}} - t_{1_{start}} \leq T_{network} \tag{4.2}$$

where n is the last module in the network.

4.3 Simulink integration

The software for all components of the SST, including the module controllers, the SSB controller and the system controller, is developed using the graphical simulation environment and programming language Simulink. The Simulink blocks are translated into the target-specific firmware with code generation with target-specific libraries and compilers.

As the real-time network is based on EtherCAT, to fully enable programming the controllers using Simulink and code generation, Simulink requires a dedicated interface for EtherCAT transmitting and receiving. A proposed interface is depicted in red in Figure 4.2. However, neither Simulink nor Texas Instruments Simulink libraries include EtherCAT slave interfaces, as the only available EtherCAT blocks are intended for the EtherCAT master.

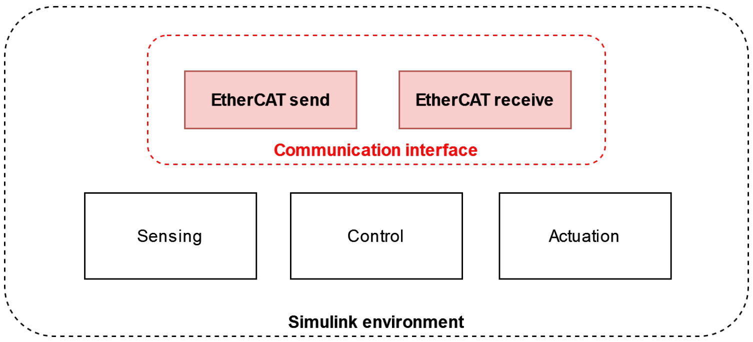


Figure 4.2: Proposed EtherCAT Simulink interface

An additional complication is the EtherCAT slave stack software for the TI-based controllers. The host microcontroller requires special firmware enabling the connection between itself and the dedicated EtherCAT slave hardware chip (ESC) to participate in the EtherCAT communication. Moreover, the EtherCAT protocol includes a state machine whose transitions must be dictated by the host microcontroller, as well as various protocol handlers. The complexity of creating EtherCAT slave stack software is recognized by the EtherCAT Technology Group (ETG), which is why they provide a code generation tool to facilitate the process called EtherCAT Slave Stack Code (SSC) [1].

The generated software stack using SSC is generic, except for a hardware abstraction layer and the user application. The software architecture of the EtherCAT slave stack is presented in Figure 4.3. As previously discussed in chapter 1, EtherCAT supports on-the-fly communication, i.e., the ESC read/writes data from the incoming EtherCAT frame. However, the EtherCAT slave, i.e., the host controller, further processes the data and executes calculations. Therefore, to enable data transfer between the ESC and the host microcontroller, a hardware abstraction layer (HAL) is required, which enables correct reading and writing of the ESC's registers, as well as supporting synchronization through several interrupts. Texas Instruments supply the HAL for the controllers used in the SST project.

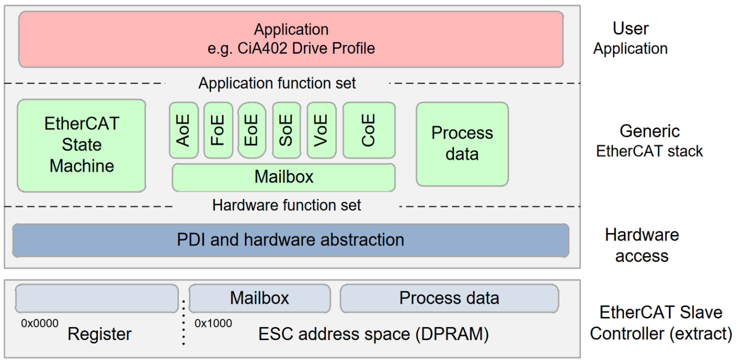


Figure 4.3: EtherCAT slave stack software architecture [1]

The user application is split into three parts: output mapping (data originating from the EtherCAT master, copied from the ESC to the host microcontroller), input mapping, and EtherCAT application. The three parts are executed in the order presented on Figure 4.4. In typical use-cases, the EtherCAT application would contain the control algorithm. However, given the requirement for Simulink integration, the use of the generated EtherCAT slave stack code is very limited in this work.

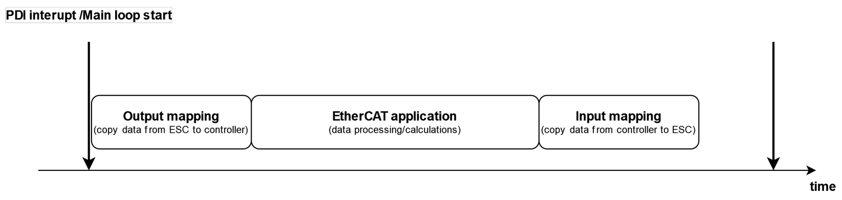

PDI interupt /Main loop start
Output mapping
(copy data from ESC to controller)
EtherCAT application
(data processing/calculations)
Input mapping
(copy data from controller to ESC)
time

Figure 4.4: User application execution order in the EtherCAT slave stack

The finalized EtherCAT software stack, including the user application, is then compiled and loaded into the host microcontroller. However, the problem becomes apparent as the control firmware must go through the same code generation, compilation, and loading process onto the microcontroller. From a design perspective, the ideal scenario would be automatically placing the generated Simulink code into the EtherCAT application. However, this is practically infeasible without manual intervention, given the complexity of the generated Simulink code.

The design question is how to integrate the EtherCAT communication and its stack with Simulink-generated firmware without requiring manual intervention (adhering to the 'ease of use' requirement).

4.4 System design overview

The proposed system design, including the real-time network integration, is presented in Figure 4.5. The firmware for all controllers, including an EtherCAT interface, is to be developed in the Simulink environment and cross-compiled to the suitable controller using Simulink code generation and support libraries. In the hardware/software integration stage, the compiled firmware should be integrated to execute together with an appropriate EtherCAT software stack, which includes a controller-appropriate hardware-abstraction layer to communicate with EtherCAT hardware. In the RTN hardware stage, the proposed physical connection can be seen. The system controller utilizes an EtherCAT switch to form a star topology. Media converters are used to ensure isolated communication through fibre optics.

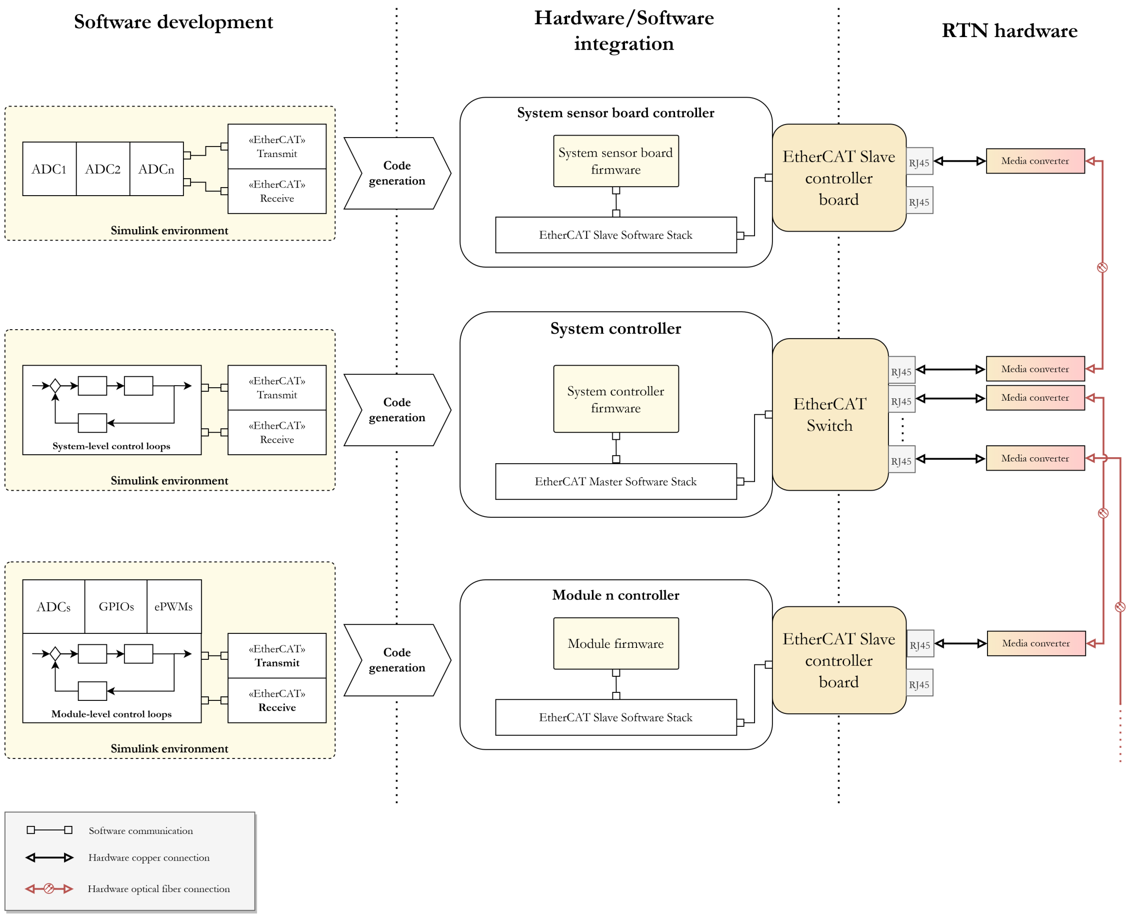

Software development
Hardware/Software integration
RTN hardware
ADC1
ADC2
ADCn
«EtherCAT» Transmit
«EtherCAT» Receive
Simulink environment
Code generation
System sensor board controller
System sensor board firmware
EtherCAT Slave Software Stack
EtherCAT Slave controller board
RJ45
Media converter
System-level control loops
System controller
System controller firmware
EtherCAT Master Software Stack
EtherCAT Switch
ADCs
GPIOs
ePWMs
Module-level control loops
Module n controller
Module firmware
Software communication
Hardware copper connection
Hardware optical fiber connection

Figure 4.5: A system design diagram for RTN Simulink integration

The system sensor board can be abstracted as part of the system controller and decoupled from the real-time network of the 18 modules, as it would technically become the 19th node otherwise. A suggestion is to use a dedicated EtherCAT network between the SSB controller and the system controller. This network division would enable potentially higher cycle frequencies without affecting the main real-time network. A further discussion is included in section 4.4.2.

The proposed design completely abstracts the real-time communication with Simulink interfaces. However, the diagram does not show the inner controller integration and scheduling of the Simulink-generated firmware and the EtherCAT software stack. These concepts are explained in the following subsections.

4.4.1 Module controller

The proposed system design in section 4.4 specifies two different software applications within the Module controller. The first application is the entire SST-related firmware which contains the module-level control loops, ADC and GPIO configurations, and ePWM peripheral configurations. The second application contains the entire software stack to support the EtherCAT protocol and communication with the dedicated EtherCAT ESC board (attached to the module controller) through an asynchronous external memory interface. Both applications are generated as distinct codebases with appropriate dependencies through Simulink code generation (supported by TI libraries) and through SSC (as discussed in 4.3). The diagram in Figure 4.5 also shows a software connection between the two applications, realised through the so-called 'EtherCAT Transmit' and 'EtherCAT Receive' blocks in Simulink.

There are two significant challenges to solve to enable this design. The first challenge is integrating the two separate codebases, and the second is to realise the Simulink EtherCAT blocks since no such blocks are intended for EtherCAT slave devices.

An apparent solution for the first challenge of codebase integration would be to merge the two codebases into a larger codebase, such that the various application functionalities are separated into distinct tasks, as discussed in 4.2. However, the problem with this approach is the noncompliance with the 'ease of use' requirement since any change in the SST firmware would require a significant amount of work outside of the Simulink environment. Furthermore, the complexity of merging two proprietary codebases and implementing a rudimentary operating system to schedule tasks is out of scope for this project.

The module controller is based on the TI TMS320F2837xD architecture (TI C2000 family of devices), which utilises a dual-core architecture. The microcontroller architecture [2] shows that the two CPUs are relatively independent with separate timers and interrupt lines, but share most peripherals such as ADCs, ePWMs, and GPIOs. As detailed in [2], *CPU1* acts as a master processor, which means that it configures all peripherals and grants ownership to *CPU2* explicitly. The full TMS320F2837xD architecture block diagram is included in Appendix A.

Since both the Simulink-generated SST firmware and the EtherCAT software stack are single-core applications, the proposed design is to deploy each application on its CPU. The Interprocessor Communication module (IPC) can be used to bridge the communication between the two processors. Moreover, the TI C2000 Simulink library includes 'IPC receive' and 'IPC transmit' Simulink blocks, which can be repurposed to serve as the 'EtherCAT receive' and 'EtherCAT transmit' blocks proposed in section 4.4. The proposed software architecture is presented in Figure 4.6. The orange-coloured blocks on *CPU2* represent the application generated in Simulink. However, since the SST module firmware utilises peripherals owned by *CPU1* (such as ePWM modules and GPIOs), some configuration code needs be present on *CPU1*, as well as remote peripheral call functions, which are also communicated through IPC.

The operational idea of the proposed architecture is to dedicate CPU1 to EtherCAT communication and remote peripheral calls, whereas *CPU2* will only execute the Simulink-generated module-level control loops and the rest of the SST firmware functionalities. The 'ease-of-use' requirement will be satisfied since any changes to the SST firmware will only affect *CPU2*; Simulink-generated code can be deployed immediately.

The IPC module can be configured to use interrupts for message signalling, which can be used to synchronise the two CPUs. As defined in section 4.2, the communication task $Task_{comm}$ will include ESC data processing within the EtherCAT software stack and the IPC communication between the two CPUs. $Task_{comm}$ is scheduled on *CPU1* and is to be triggered by incoming EtherCAT frames at the communication frequency, i.e., 8KHz. Apart from $Task_{comm}$, no other periodic task is scheduled on *CPU1*.

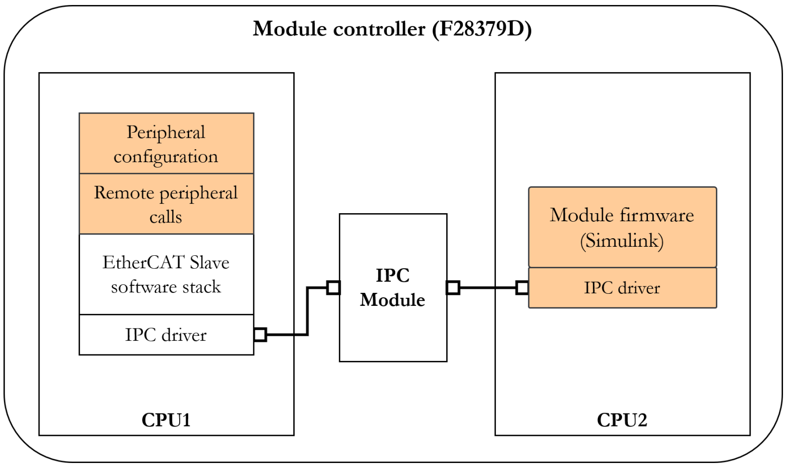


Figure 4.6: The proposed module controller software architecture.

The module firmware which includes the control loops can be abstracted as a single task to be executed periodically at the same 8KHz frequency, while satisfying the performance requirements discussed in section 3.2.1. Therefore, we define the task $Task_{module}$, with a period $T_{module} = T_{comm} =$ 8KHz, and execution time e_{module}. As discussed in section 4.2, Equation 4.1 must be satisfied in order to ensure that all tasks can be scheduled and meet their deadlines.

Since both CPUs are utilised to complete the communication-control cycle, a timing diagram including both CPUs and their interaction represents the schedule, shown in Figure 4.7. This means that Equation 4.1 must be further defined such that $\sum_{i=0}^{k} e_i$ (where e_{module} is included) is the sum of the execution times of all tasks dependent on $Task_{comm}$ within the controller.

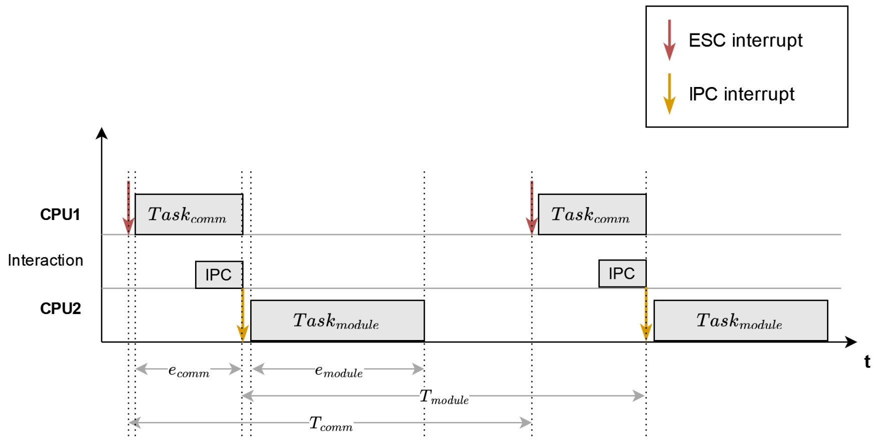


Figure 4.7: The proposed module controller task schedule.

The $Task_{comm}$ must be implemented on *CPU1* outside of the Simulink environment but is considered as a one-time setup; thus, it will not impact the 'ease-of-use' criteria. As discussed in section 4.3 the EtherCAT software stack requires implementation of the user application, i.e., the functions shown in Figure 4.4. The *Output mapping* function reads the ESC's registers containing the latest incoming frame data and stores the data in memory. The *EtherCAT application* will pack the data and send it to *CPU2* through the IPC driver. Additionally, it will read the IPC buffer to check for data originating in *CPU2*, i.e., data that must be sent back to the system controller. The *Input mapping* function will store any outbound data into the ESC registers, which will be collected with the next EtherCAT frame.

4.4.2 System sensor board controller

The system sensor board controller controls the external ADCs which sample voltages and currents from both the MV and the LV sides of the SST. The SSB's primary role is to send its sampled and processed data to the system controller, as shown in Figure 4.1. Furthermore, the SSB implements a PLL loop which takes the MV grid voltage and generates a stable fundamental representation of it, i.e., it removes harmonic distortion from the grid voltage. The SSB is designed to sample and process the data periodically, with a period equal to the real-time network communication frequency, i.e., 8KHz.

However, the SSB can be viewed as part of the system controller, which happens to also use EtherCAT for communication, which is a common industry practice. The real-time network discussed so far in this work excludes the SSB from the requirements definition in chapter 3. Furthermore, it is possible that the ADC sampling frequency might be increased in the future, i.e., oversampling. Therefore, it is proposed to abstract the SSB as part of the system controller, which uses a separate EtherCAT real-time network for communication. The proposed abstraction is depicted in Figure 4.8.

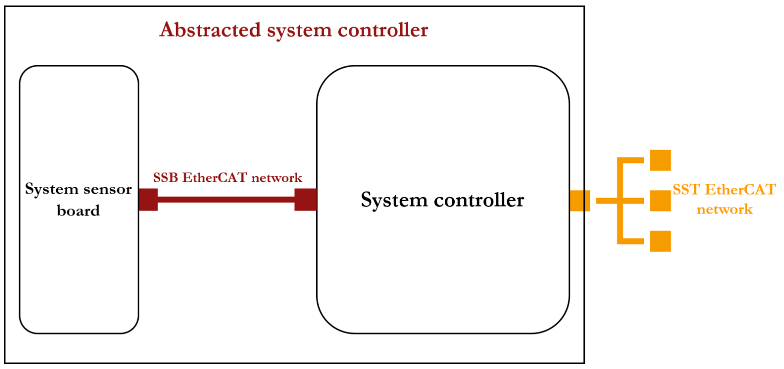


Figure 4.8: Abstracting the SSB as part of the system controller (separate EtherCAT network).

The SSB controller is based on the same hardware as the module controllers, i.e., the Texas Instruments TMS320F28379D microcontroller. To enable both EtherCAT communication and programmability in the Simulink environment, the same approach presented in section 4.4.1 is used.

Similarly, a communication task $Task_{comm}$ can be defined which handles EtherCAT and sequentially IPC communication, with a period T_{comm} which may be equal or greater to the system-level communication frequency of 8KHz, i.e., $T_{comm} \geq T_{network} = $ 8KHz, and an execution time e_{comm}. The two other SSB functionalities, i.e., the ADC sampling and the PLL loop, can be abstracted as one task, since they sequential and synchronized to the communication task through IPC interrupts. Therefore, the task $Task_{ssb}$ is defined with a period $T_{ssb} = T_{comm}$, and execution time e_{ssb}. The proposed schedule is presented in Figure 4.9.

4.4.3 System controller

The system controller's responsibility includes initiating real-time communication and implementing the system-level control loops described in section 1.1.2. The system control has the role of the EtherCAT master in the real-time network. Since the EtherCAT protocol enlists dedicated ESC hardware to process and correctly sort the data within the frame, the EtherCAT master is straightforward and uninvolved in frame processing. Because of the simplicity, an EtherCAT master only requires an Ethernet interface, usually implemented using DMA, which means that there is little overhead for the EtherCAT interface so that the CPU can be almost entirely dedicated to other tasks.

Unlike all other controllers in the SST system, the SST system controller is based on the

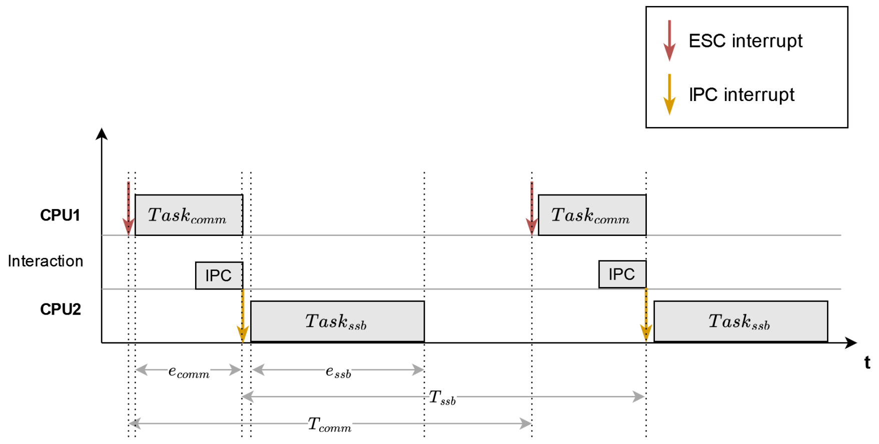


Figure 4.9: The proposed system sensor board controller task schedule.

Beckhoff CX2000 architecture, which is a multi-CPU Embedded PC. Using TwinCAT [30], a software system developed by Beckhoff, any general-purpose PC can be transformed into a real-time computer for automation applications such as PLCs or any runtime system. The Beckhoff CX2040 system controller contains two Ethernet interfaces, which can be used for EtherCAT. Moreover, the I/O interface can be extended using EtherCAT switches.

The system controller hardware inherently supports Simulink integration through I/O blocks available in Simulink. However, instead of generating native code through cross-compilation, Simulink generates code for a so-called *TwinCAT target for MATLAB®/Simulink®*. The TwinCAT target uses the generated C++ code to create a TwinCAT Object Model, i.e., a TcCOM object, which exhibits the same I/O behaviour as the Simulink model. Finally, the TcCOM object can be loaded into the TwinCAT software, configured to run as a periodic task, and executed in real-time on a certain resource. The entire Simulink-to-execution process is shown in Figure 4.10, where the generated TcCOM object is contextualised as a task and scheduled to execute periodically on *CPU1*. Since the TcCOM object contains both the Simulink firmware and the EtherCAT data mapping, the two tasks can not be decoupled. Therefore, a single task $Task_{sys}$ is defined, with a period of $T_{sys} = T_{network}$.

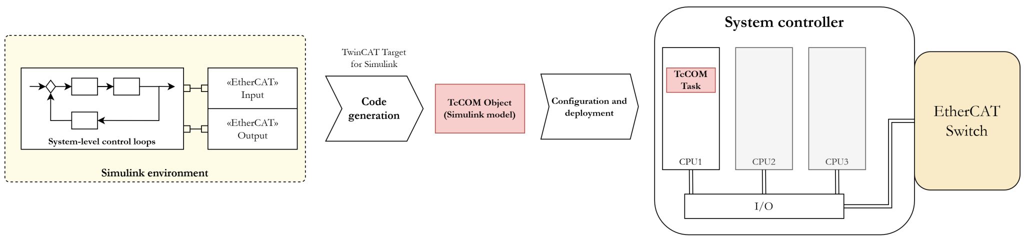


Figure 4.10: Simulink integration and firmware deployment for the system controller

4.4.4 Integration

The previous sections discussed and proposed how to adapt all controllers individually for communication using the real-time network integrated with Simulink. All the internal processing has been modelled as tasks within each controller, typically split into a communication task and an SST processing task. The communication task abstracts all EtherCAT-related processing and IPC communication when necessary. The remaining processing related to SST specific applications has been abstracted into one task, since all processing is sequentially synchronized to the system level communication.

The complete proposed RTN architecture is presented in Figure 4.11. The system controller participates in two EtherCAT real-time networks: the main system RTN, and the auxiliary system sensor board RTN. The rationale behind the network division, discussed in section 4.4.2, is the system controller abstraction and potential for higher frequency sampling using the SSB. The system RTN uses a star topology with fibre optics as a transmission medium to comply with both the fault tolerance and transmission medium requirements.

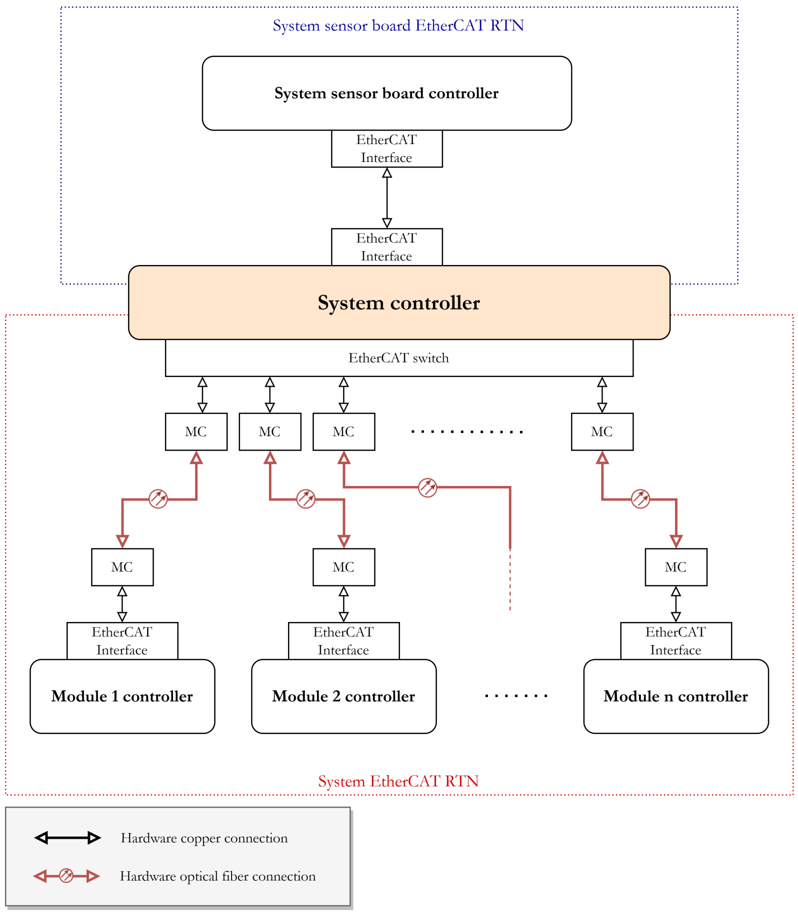


Figure 4.11: The SST RTN architecture.

The proposed individual designs have been accompanied by local controller schedules presented in Figure 4.7 for the module controllers and Figure 4.9 for the SSB controller. However, at a system level, a global schedule is needed to reflect the integrated operation of all controllers. Figure 4.12 depicts a proposed schedule for the main system network at a global level. Individual controller schedules are positioned relative to the system controller schedule and inherent RTN propagation time. The RTN frame propagates after $Task_{sys}$ completes execution on the system controller. The first module's ESC interrupts its *CPU1* to start the new cycle's execution, i.e., the $Task_{comm_1}$. Since the ESC hardware is decoupled from the module controllers, the frame continues to propagate while Module 1 is still processing $Task_{comm_1}$, and later $Task_{module_1}$. After an inherent propagation delay t_{prop}, the frame reaches Module 2 and triggers the same behaviour as the first module. The message propagates and reaches the final module after a total propagation time of $n \cdot t_{prop}$. As discussed in section 4.2, the proposed design must satisfy Equation 4.2, i.e., the end time $t_{n_{end}}$ relative to the start of the frame must be reached before the start of the next frame. As defined in section 3.2.3, the synchronization requirement is satisfied with this schedule since all modules operate on the data from the same cycle.

EtherCAT's distributed clocks mechanism was not utilized in this design due to hardware

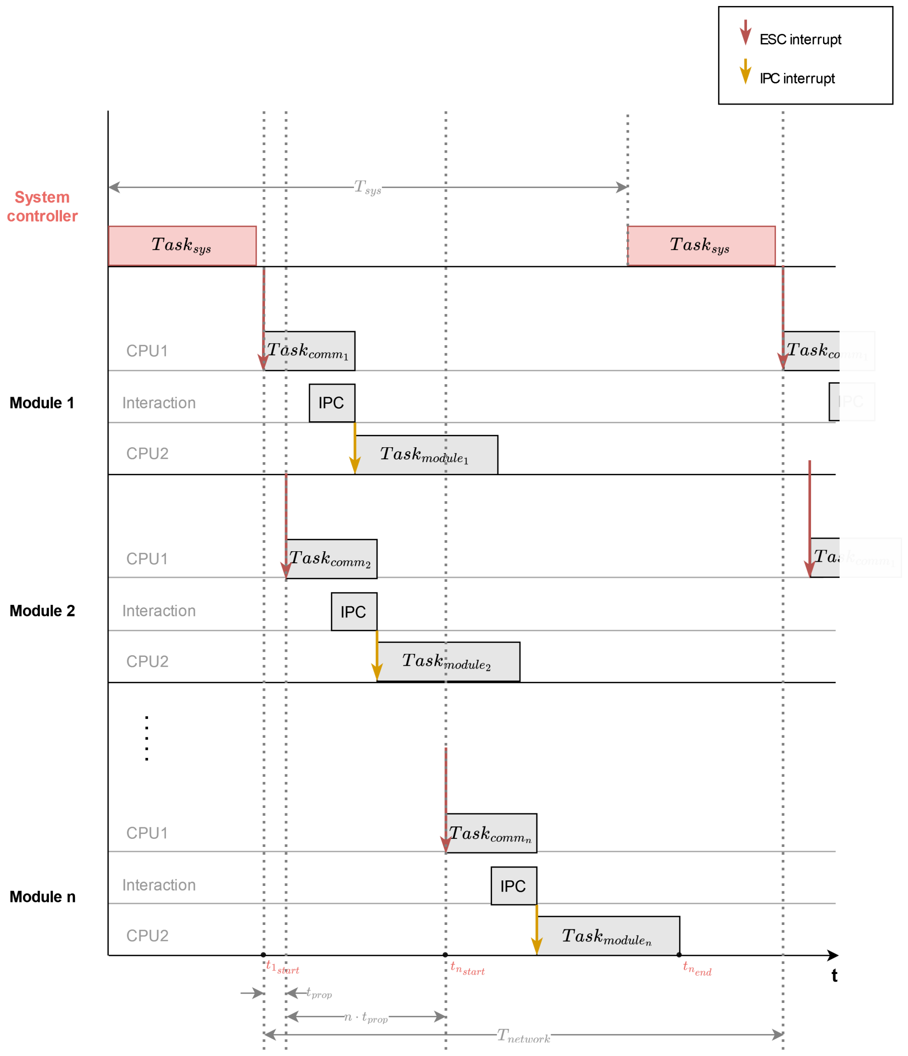


Figure 4.12: The proposed system level schedule

limitations with the chosen communication frequency of 8KHz. However, if distributed clocks is
845 used, tasks $Task_{module_k}, k \in [1, n]$, would start at $t_sync = t_{n_{start}} + e_{comm_n} + d_{DC}$, where d_{DC} is
a small necessary delay.

4.5 Summary

This chapter presents the design decisions for integrating and implementing the SST real-time network based on a defined communication model. Furthermore, the section includes the design
850 implications of the Simulink integration requirement on the overall system design. Each controller is designed to enable participation in the RTN communication and an interface to the network through Simulink. The RTN Simulink interface can connect incoming and outgoing data to the SST control loops on all communication. The seamless Simulink integration has been achieved by utilizing the dual-core architecture of the Module and SSB controllers, i.e., by dedicating *CPU1*
855 for EtherCAT, and *CPU2* for Simulink-generated firmware, bridged through IPC communication.

Chapter 5

Analysis and experimental verification

5.1 Introduction

This chapter describes the validation of the real-time network for the TU/e SST through analysis and experimental verification of relevant requirements defined in chapter 3. The included experimental tests are related to measurable and quantifiable requirements such as network performance (communication delay, control delay, jitter, and cycle time analysis). In addition, the synchronization and scalability requirements are verified analytically using the network performance measurements. Finally, requirements which cannot be experimentally verified, or are satisfied by design, are discussed qualitatively.

5.2 Experimental setup

The TU/e SST prototype has been down-sized and de-scoped from the original three-phase 18-module design, which was the system of interest for this study. However, almost all requirements have been generalized for an arbitrary number of modules. The current TU/e prototype is composed of a single-phase three-module design, but all design principles should apply to the down-sized system.

A significant change in the experimental setup compared to the original design is the change of topology for the real-time network. Instead of using a star topology to comply with the fault tolerance requirement described in section 3.3.2, a ring topology is used. The change was due to hardware and timing constraints, specifically the lack of a sufficiently large EtherCAT switch for the system controller. However, the difference in propagation time should be negligible for the small number of connected modules. Using a ring topology instead of a star topology will reduce the system's fault tolerance from n-fault to 1-fault tolerant. For the experiments conducted in this study, a 1-fault tolerant network is sufficient.

The system architecture used for the experiments presented in this chapter is shown in Figure 5.1; it is mainly similar to the design architecture proposed in section 4.4.4 except for the number of modules and the system RTN topology.

Another consequence of the down-scaled system is a smaller EtherCAT data frame, which should result in a shorter cycle time, as explained in section 3.3.1. However, the data frame was appended with ten auxiliary system-to-module variables to keep propagation times in line with calculations for 18 modules and to future-proof the research-oriented SST prototype.

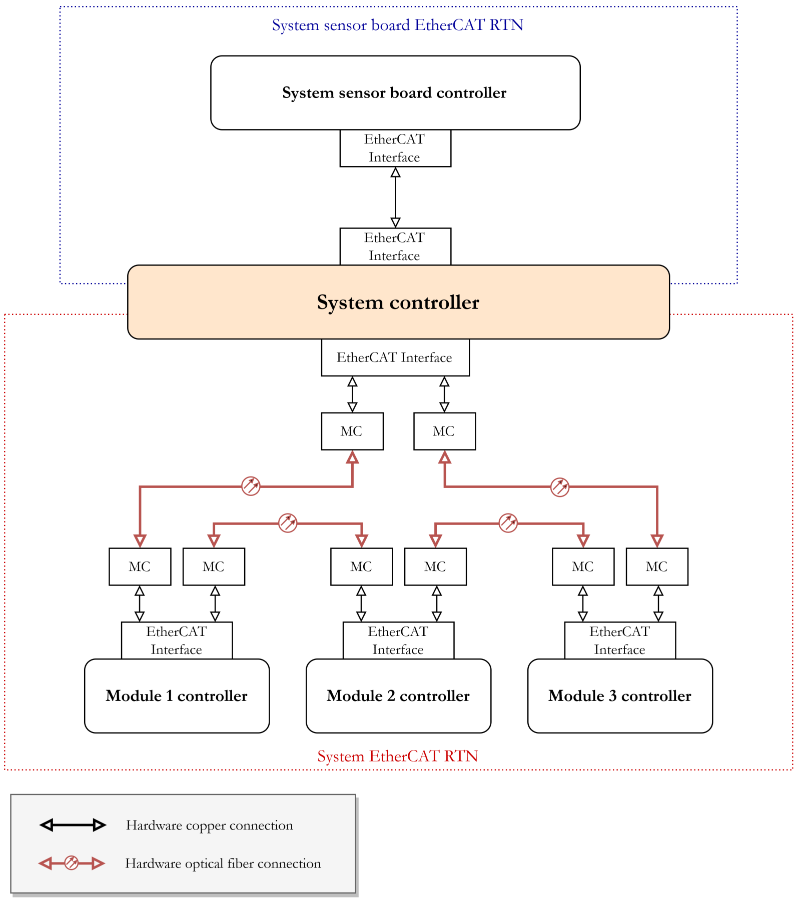


Figure 5.1: The prototype SST RTN architecture.

5.3 Network performance

Network performance in the context of this work has been defined as the achievable communication cycle time, i.e., communication frequency. The performance requirement defined in section 3.2.1 is based on sufficient sinusoidal sampling to allow the SST higher harmonic compensation, and was determined to be a minimum of 250μs cycle time. The selected frequency during design and implementation, as described in chapter 4, was 125μs, i.e. 8KHz frequency, which exceeds the minimum required cycle time. However, it has to be verified whether the cycle time of 125 μs is sufficient to allow all SST-related processing to be completed before the next cycle. Furthermore, this section defines the network communication delay, control delay, and jitter to aid in the analysis.

5.3.1 Communication delay

The communication delay τ_{cd} is defined as the amount of time it takes for a data packet to hop from one node to the next, or in this case, from one module to the next. If the data packet i reaches module controller n at time t^n_{cd}, and then module controller $n+1$ at time t^{n+1}_{cd}, then the communication delay is measured as:

$$\tau_{cd} = t^{n+1}_{cd} - t^n_{cd} \tag{5.1}$$

The measurement is performed as shown in Figure 5.2, i.e., when the data packet i reaches the ESC of module controller n, a PDI interrupt is generated, which is detected using an oscilloscope. The difference between the arrival of the PDI interrupts in the two modules is then calculated to result in the communication delay.

Figure 5.3 shows an oscilloscope measurement based on Figure 5.2 using the three modules of the SST prototype. The receipt of the interrupts shown in Figure 5.2 toggle a specific GPIO on the module controller which is measured using an oscilloscope. The average communication delay was measured to be $\tau_{cd} = 1.66\mu s$.

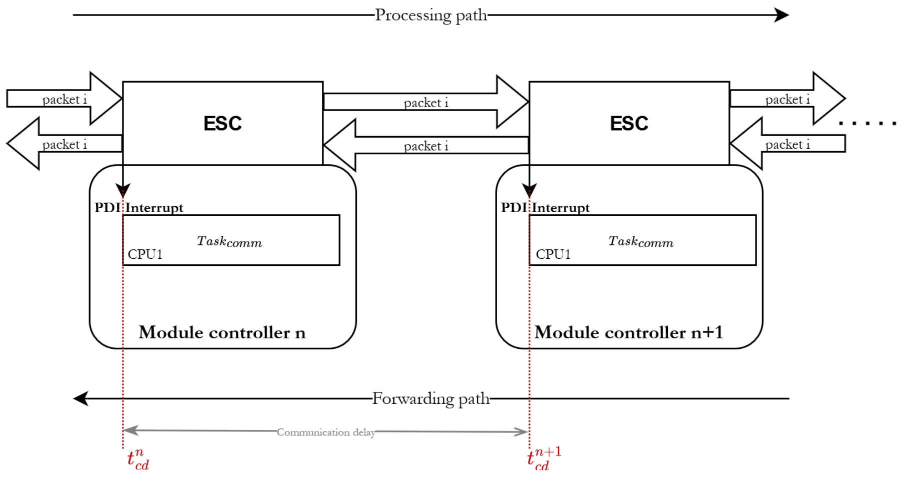


Figure 5.2: Communication delay measurement diagram.

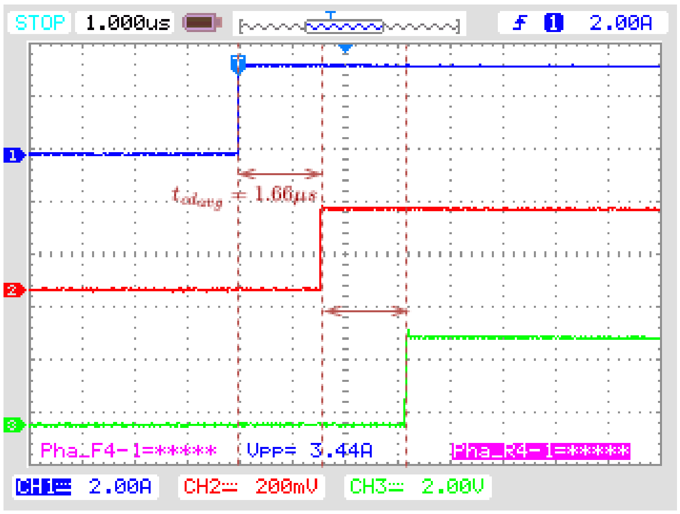


Figure 5.3: Average communication delay measurement (oscilloscope)

5.3.2 Control delay

Another type of delay can be defined as the control delay τ_{ctr}, or the amount of time from the arrival of a data packet in the module until the start of the control loop. Measuring the control delay is important since the task $Task_{module}$ containing the Simulink-generated SST control loops is triggered by an IPC interrupts during the execution of the communication task $Task_{comm}$. The control delay measurement is performed as shown in Figure 5.4. When the EtherCAT message in packet i reaches the ESC of module controller n, a PDI interrupt is generated at time t^{i}_{m} which triggers $Task_{comm}$. During the execution of $Task_{comm}$, the message data is transferred from *CPU1* to *CPU2* using IPC, so an IPC interrupt is generated at time t^{i}_{c} which triggers $Task_{module}$.

The control delay for packet i is calculated with the following equation:

$$\tau_{ctr} = t_c^i - t_m^i \tag{5.2}$$

Figure 5.5 depicts the results of the average control delay measurement using an oscilloscope on the SST prototype, i.e., $\tau_{ctr} = 14.2\mu s$. The control delay is measured on the second module where the orange falling-edge represents the receipt of the EtherCAT packet on *CPU1*, and the red falling-edge represents the start of the control loop after the data has been received through IPC on CPU2. Unrelated, the blue and green pulses represent the duration of the communication tasks for the first and third modules, respectively.

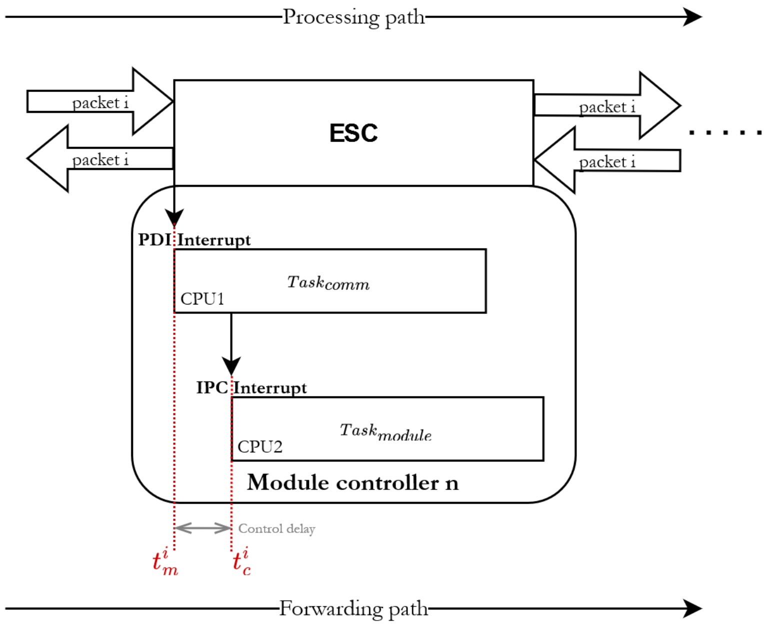


Figure 5.4: Control delay measurement diagram.

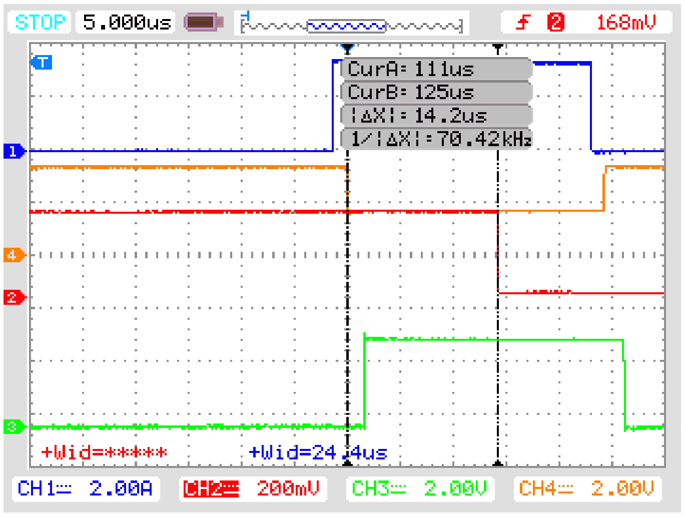


Figure 5.5: Message to control delay.

5.3.3 Jitter

The communication delay defined in section 5.3.1 is susceptible to jitter for multiple reasons including media converters and interrupt latencies. Jitter represents the time variation of the communication delay. The jitter can be represented as a probability distribution of the communication delay τ_{cd}.

Jitter was observed during the measurements for the communication delay, as shown in Figure 5.6, which appears to be bounded, as expected. The measurement was repeated with the distributed clocks mechanism (at 16KHz communication frequency instead of 8KHz) to verify that the jitter was not caused from lack of explicit synchronisation. The results from the oscilloscope shown in Figure 5.7 show that even with distributed clocks, jitter was present, but centered to the signal of the reference module.

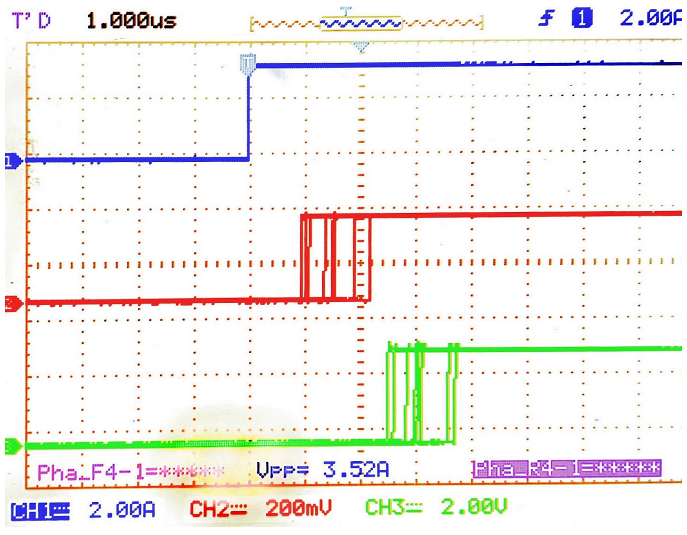


Figure 5.6: Communication jitter measurement.

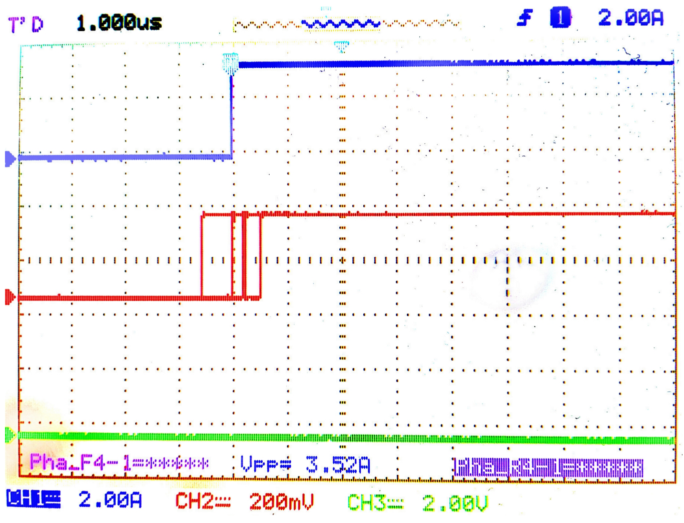


Figure 5.7: Communication jitter measurement with DC enabled.

The jitter of the communication delay τ_{cd} was measured using an oscilloscope which measures the delay between the rising edges of two signals (a rising edge represents the start of interrupt execution on the receipt of a EtherCAT message). The results were plotted in a series using a built-in oscilloscope history function, as presented in Figure 5.8.

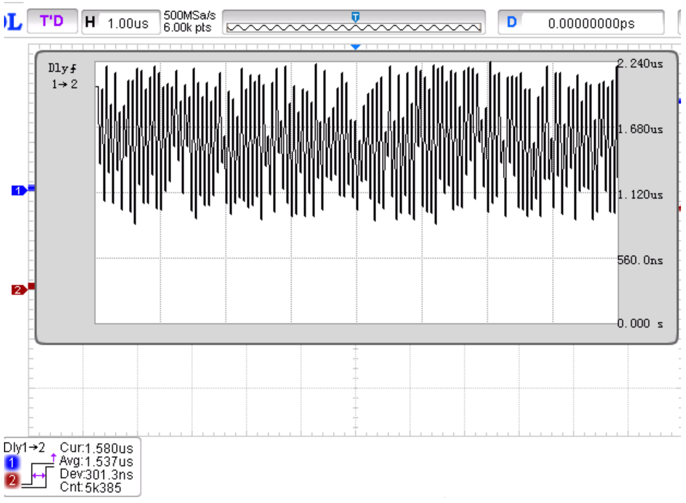


Figure 5.8: Jitter variation measurement.

The communication delay data was extracted and plotted as a probability distribution, shown in Figure 5.9. The probability distribution appears to be multimodal with a significant peak around the average of $1.537\mu s$. If the distribution is approximated as a standard distribution, the standard deviation is $\sigma = 0.301\mu s$. The worst-case observed communication delay value within 5385 measurements is $\tau_{cd_{max}} = 2.21\mu s$.

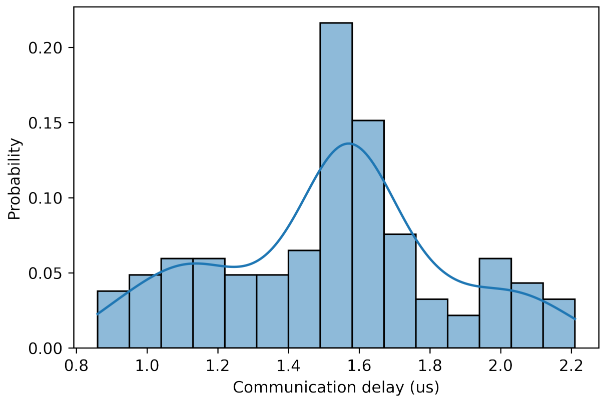


Figure 5.9: Communication delay jitter distribution.

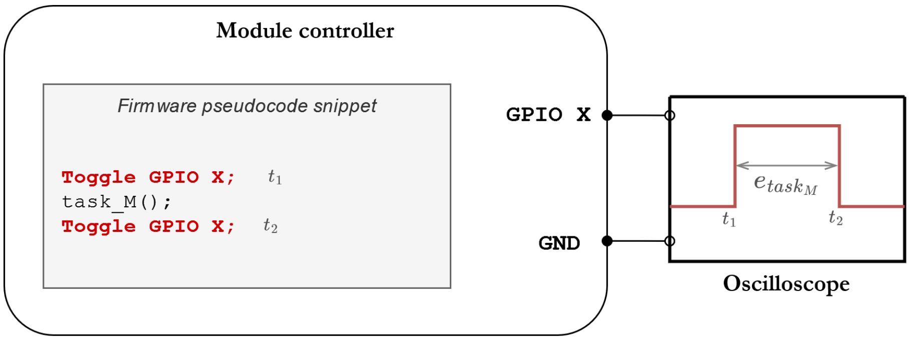
5.3.4 Cycle time analysis

The communication delay, control delay, and jitter can be used to calculate the network cycle time. Additionally, further measurements are required since the cycle time in this context includes the execution time of all SST-related processing. The proposed design of the integrated communication system was based on task-based modelling, as described in section 4.2, and depicted on Figure 4.12. However, the duration of the presented tasks is arbitrary and only driven by assumptions. All task execution times were measured using code profiling through GPIOs, to ensure the proposed design can support the target cycle time of 125 μs. Figure 5.10 shows an example of execution

Task	Average execution time	Task description
$Task_{comm}$	$25.6\mu s$	EtherCAT stack, ESC communication, and user application (IPC communication).
$Task_{module}$	$24\ \mu s$	Simulink-generated SST module firmware containing sensing, local control loops, actuation, and IPC communication (to EtherCAT)
$Task_{sys}$	8.8	EtherCAT master stack, and Simulink-generated SST system controller firmware containing data mapping
$Task_{ssb}$	23.6	Simulink-generated SST system sensor board firmware containing sensing, a PLL control loop, and IPC communication (to EtherCAT)

Table 5.1: Measured average task execution times.

time profiling for the arbitrary $Task_M$. To measure the the execution time of $Task_M$, a GPIO toggling command is issued before the task is executed, and a second toggling command directly after. The measurement-related GPIO is connected to an external oscilloscope, and a pulse formed by the two toggling moments t_1 and t_2 should be visible. The pulse width is equal to the execution time e_{task_M} of $Task_M$. It is important to note that higher priority interrupts must be disabled during measurements to measure the best-case execution time. The worst-case execution time can be measured if all periodic higher priority interrupts are triggered during the task execution time. However, since the SST and the RTN are considered firm real-time, measurements are related to the average execution time.



```
Toggle GPIO X;   t1
task_M();
Toggle GPIO X;   t2
```

Figure 5.10: Code profiling method used for measurements.

The results of the average execution times of the tasks defined in chapter 4, measured using the described method, are presented in Table 5.1. The measurement results presented in the table are subject to change during the SST research and development, as the firmware implementation, which is out of the scope of this work, may change. However, the real-time network limits the execution times of the tasks, such that the average cycle time $T_{cycle_{avg}}$ must be less than the communication period of $125\mu s$. Section 5.5 details these limits depending on the system scalability.

The average cycle time $T_{cycle_{avg}}$ can be calculated with the following equation:

$$T_{cycle_{avg}} = e_{sys} + 3 \cdot \tau_{cd} + \tau_{ctr} + e_{module} \tag{5.3}$$

where e_{sys}, and e_{module} are the average execution times (given in the second column of Table 5.1) of $Task_{sys}$ and $Task_{module}$, respectively. Since the processing tasks start last in the third module, only its execution times are calculated into the equation. The result of the cycle time for the SST prototype setup is 51.98 μs.

Equation 5.3 does not include the measured jitter into the cycle time calculation. By including the jitter in the equation, the cycle time T_{cycle} would become a probabilistic variable. However, the jitter (contained in the maximum of the communication delay), and the maximum of task execution times can be used to calculate the maximum cycle time by adapting the equation:

$$T_{cycle_{max}} = e_{sys_{max}} + 3 \cdot \tau_{cd_{max}} + \tau_{ctr_{max}} + e_{module_{max}} \tag{5.4}$$

The system sensor board task is not included in calculations since it operates on its own dedicated EtherCAT network, as explained in section 4.4.2.

5.4 Synchronization

As defined in section 3.2.3, the synchronisation requirement ensures that all modules process the data from the same incoming EtherCAT frame within a communication cycle. By design, the start of the module-level control loops is synchronized to the receipt of an EtherCAT message, as shown in Figure 4.7. Furthermore, measurements presented in section 5.3 verified that the module control loop completes execution before the end of the communication cycle, i.e.:

$$\tau_{ctr} + e_{module} = 38.2\mu s \leq T_{network} = 125\mu s. \tag{5.5}$$

therefore, according to the requirement, the modules are sufficiently synchronized.

While further synchronization using distributed clocks could theoretically synchronize the start of the control loops on each module within a few hundred nanoseconds, the measured jitter was in the range of $1\mu s$ (see Figure 5.7). Furthermore, distributed clocks at the chosen communication frequency of 8KHz was not permitted due to hardware constraints.

5.5 Scalability

The scalability of the SST prototype cannot be experimentally verified due to hardware constraints. However, using the performance measurements defined in section 5.3, scalability can be determined analytically. Scalability implies that the SST real-time network can support the addition of new modules without disrupting communication, i.e., the total communication cycle time T_{cycle} must remain less than the network communication period of $125\mu s$. Using the obtained measurements in section 5.3 and Equation 5.3, the average cycle time for 18 modules is $76.88\mu s$. However, when determining scalability, the worst-case values should be considered (including the jitter); therefore, using Equation 5.4, the maximum cycle time for 18 modules is $86.78\mu s$. The result proves that the experimental setup of three modules can be extended to the original 18 modules without exceeding the communication cycle time of $125\mu s$.

The maximum scalability of the SST real-time network, i.e., the maximum amount of modules which can be supported by the proposed SST RTN, can be analytically defined as a simple integer programming problem:

$$\begin{array}{ll} \text{maximize} & n \\ \text{subject to} & e_{sys_{max}} + n \cdot \tau_{cd_{max}} + \tau_{ctr_{max}} + e_{module_{max}} \leq 125\mu s \\ & n \in Z^{+} \end{array} \tag{5.6}$$

The solution to the integer problem defined in Equation 5.6 is 35, meaning that the real-time network can support up to 35 modules given the experimental measurement data. The solution closely aligns with the analytically predicted result in section 3.3.1. The task execution times in the constraint in Equation 5.6, i.e., $e_{sys_{max}}$ and $e_{module_{max}}$, are treated as constants obtained from the measurements presented in Table 5.1. While the execution times of these tasks are out of this work's scope, their limits concerning scalability can be determined by analyzing the integer problem in Equation 5.6. For example, the module task execution time $e_{module_{max}}$ to determine its limits for any number of modules in the range of $n \in [3, 35]$. The graphical solution to the integer problem is presented in Figure 5.11, where the possible execution time $e_{module_{max}}$ value can be found on the vertical lines for any (integer) number of modules.

For the original proposed design of 18 modules, the limits of the execution times for the system and module tasks, i.e., $e_{sys_{max}}$ and $e_{module_{max}}$ respectively, can be found in the solution space presented in Figure 5.12, which means that the growth of any execution time has an impact on the limits of the other task's execution time. The solution space can be extended if one of the tasks is simplified or optimized, so its execution time is lower than the currently measured one.

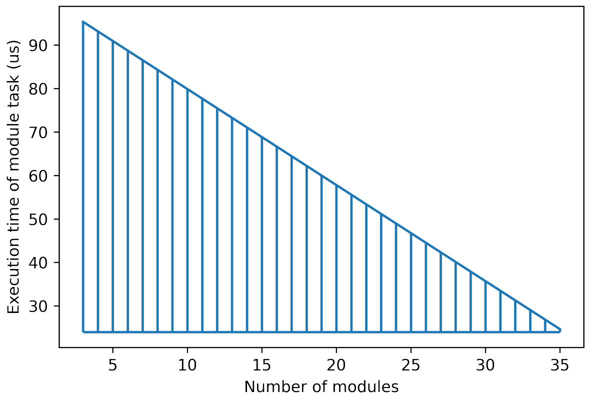


Figure 5.11: Limits of the module task execution time for module scalability.

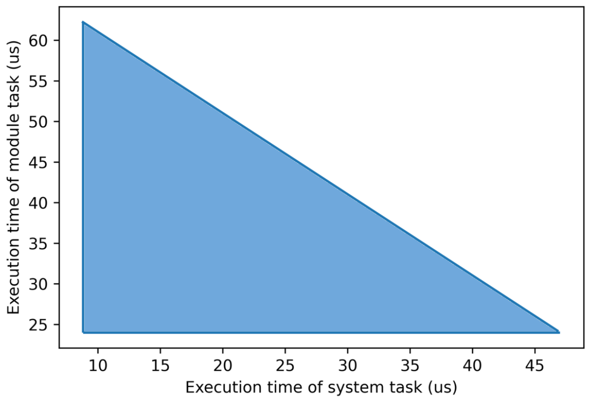


Figure 5.12: Trade-off of the system and module tasks' execution times (18 modules).

5.6 Further discussion

Some of the requirements defined in chapter 3 are inherently unquantifiable, already satisfied by design, or out of scope for this work.

5.6.1 Data frame

The data frame requirement is determined by the amount of data required by the SST control loops and algorithms, and is incorporated into the proposed design in chapter 4. The impact of the data frame size is reflected onto the network performance. Since the data frame presented in Table 3.1 is determined on the basis of 18 modules, the requirement can be scaled down to a minimum of *38 bytes*. However, the decision was to pad the minimal frame with 5 auxiliary system-to-module outputs, and 5 auxiliary system-to-module inputs, i.e., a total of 30 variables of size INT16. Therefore, the total data frame size is *98 bytes*, or approximately 2.5 times the minimum required size.

5.6.2 Transmission medium

The transmission medium is an inherited real-time network requirement from the SST isolation requirement. The choice of fibre optics as a transmission medium is out of this work's scope but impacts the network's performance. The transmission medium requirement is satisfied by design, but its effect on isolation can only be verified through complete prototype testing, which is out of this project's scope.

5.6.3 Fault tolerance

Due to the necessary topology change for the SST prototype described in section 5.2, the defined fault tolerance requirement cannot be satisfied. As a result, the ring-topology is single-fault tolerant rather than n-fault tolerant. However, a single-fault tolerance has been deemed satisfactory in the limited three-module prototype. Moreover, due to the physical separation of the module controllers and their ESC hardware, the prototype is n-fault tolerant to software faults since ESCs can still forward EtherCAT frames as long as they are powered on. If a complete power failure occurs in one of the modules, the remaining two can still be reached due to the ring topology's redundancy.

5.6.4 Ease of use

The ease of use requirement relates to Simulink integration, which would enable the development of the SST firmware within a relatively familiar graphical environment. The design decisions in chapter 4 were significantly impacted by this requirement. A real-time network Simulink interface for sending and receiving data was abstracted through IPC communication blocks. Furthermore, the communication layer on the controllers is not impacted by changes in the Simulink firmware.

From a researcher's point of view, they can experiment with the SST or further develop it without the necessity to get familiar with the real-time network implementation, which arguably satisfies the ease of use requirement.

5.7 Summary

This chapter presents the experimental validation of the proposed real-time network for the TU/e SST. The real-time network is implemented in a down-sized three-module SST prototype which is used as a verification platform for the requirements defined in chapter 3. The real-time network is instrumented to measure its performance in terms of communication delay, control delay, jitter, and cycle time. The measurements are used to analytically predict the network's scalability and compare it to earlier analyses and define the limits of the system and module tasks' execution times. The analysis shows that the real-time network can support up to 35 modules while satisfying its functional requirements. Furthermore, unquantifiable requirements are discussed and analyzed to complete the validation.

Chapter 6

Conclusions and future work

The work identifies the necessity for a real-time network in modular solid-state transformers and recognizes the gap in the literature concerning its requirements. Although some discussions for communication aspects in related power electronics applications exist, the research should be extended to the case of solid-state transformers as a potential solution for future grid challenges.

This graduation project investigates and defines the requirements for a real-time network in a modular solid-state transformer developed as a research platform at the Eindhoven University of Technology. The proposed requirements include network performance, synchronization, data frame, transmission medium, network topology, scalability, fault tolerance, and ease of use. Next, a real-time network design based on EtherCAT is proposed, which enables seamless integration with Simulink as a development environment for the SST firmware. The proposed design decouples the communication and control by utilizing the dual-core architecture of the module controllers. The Simulink integration facilitates research and further development of the SST by providing a communication interface and abstracting the low-level firmware development through graphical programming in a relatively familiar environment. The pragmatic design is implemented and integrated with an existing TU/e SST prototype, which is later used to verify and validate the proposed real-time network according to the defined requirements. Experimental measurements and analysis show that the proposed real-time network can support up to 35 modules without impacting its performance.

The contributions of this graduation project are towards the literature gap on real-time networks for solid-state transformers by introducing a requirement investigation and validation on a pragmatic SST prototype. Furthermore, the work proposes a design enabling full Simulink integration to facilitate the research and development of the SST.

This work can be extended and improved in several ways. First, the real-time network requirements discussion is limited to the EtherCAT protocol, a research constraint. However, their analysis and definition should be protocol agnostic to generalize the requirements for broader application. Second, EtherCAT's distributed clocks should be incorporated into the proposed design to improve synchronization and determinism. While the currently defined synchronization requirement is sufficient for the proper function of the TU/e SST's control loops, it could potentially limit control algorithms which inherently require synchronized actuation. Third, simulations could be used to verify further some requirements such as scalability and fault tolerance which are inherently bounded by hardware constraints. However, it can be argued that for this use case, it is infeasible to closely model all influences on the network's performance, such as propagation delays, interrupt latencies, and media converter delays. Moreover, the experimental tests could be extended to produce more detailed measurements, e.g. the communication delay could be decomposed into frame preparation delay, frame data transfer time, media converter delay, and interrupt latency. Consequently, by extracting the frame data transfer time, an analysis of the impact of data scaling can be conducted.

Bibliography

[1] William Mcmurray. Power converter circuits having a high frequency link, 1968. 1

[2] James L. Brooks, Roger I. Staab, James C. Bowers, and Harry A. Nienhaus. Solid state regulated power transformer with waveform conditioning capability, 1980. 1

[3] J. W. Van Der Merwe and H. Du. The solid-state transformer concept: A new era in power distribution. *IEEE AFRICON Conference*, 2009. 1

[4] Alex Q. Huang and Jay Baliga. FREEDM System: Role of power electronics and power semiconductors in developing an energy internet. *Proceedings of the International Symposium on Power Semiconductor Devices and ICs*, pages 9–12, 2009. 1

[5] Bram van Dam. *Medium Voltage Solid-State Transformer: An IEC60076-3 based design.* PhD thesis, Eindhoven University of Technology, 2022. Unpublished doctoral dissertation. 2

[6] M.A. Livani, J. Kaiser, and W.J. Jia. Scheduling hard and soft real-time communication in the controller area network (can). *IFAC Proceedings Volumes*, 31(14):13–18, 1998. 23rd IFAC/IFIP Workshop on Real Time Programming 1998 (WRTP '98)., Shantou, China, 23-25 June. 5

[7] Gunnar Prytz. A performance analysis of ethercat and profinet irt. In *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pages 408–415, 2008. 7

[8] Hamed Shadfar, Mehrdad Ghorbani Pashakolaei, and Asghar Akbari Foroud. Solid-state transformers: An overview of the concept, topology, and its applications in the smart grid. *International Transactions on Electrical Energy Systems*, 31(9):1–24, 2021. 8

[9] Ahmed Abu-siada, Jad Budiri, and Ahmed F Abdou. Solid State Transformers Topologies , Controllers , and Applications : State-of-the-Art Literature Review. 2018. 8

[10] Mahammad A. Hannan, Pin Jern Ker, Molla S.Hossain Lipu, Zhen Hang Choi, M. Safwan Abd Rahman, Kashem M. Muttaqi, and Frede Blaabjerg. State of the art of solid-state transformers: Advanced topologies, implementation issues, recent progress and improvements. *IEEE Access*, 8:19113–19132, 2020. 8

[11] Sixifo Falcones, Xiaolin Mao, and Raja Ayyanar. Topology comparison for solid state transformer implementation. *IEEE PES General Meeting, PES 2010*, pages 1–8, 2010. 8

[12] Fernando Briz, Mario López, Alberto Rodríguez, and Manuel Arias. Modular Power Electronic Transformers. *IEEE Industrial Electronics Magazine*, (december):6–19, 2016. 8

[13] Zhiyu Zhang, Hengyang Zhao, Shihang Fu, Jianjiang Shi, and Xiangning He. Voltage and power balance control strategy for three-phase modular cascaded solid stated transformer. *Conference Proceedings - IEEE Applied Power Electronics Conference and Exposition - APEC*, 2016-May(51277162):1475–1480, 2016. 8

[14] Sebastian Stynski, Marta Grzegorczyk, Cezary Sobol, and Radek Kot. Current control strategies for a star connected cascaded h-bridge converter operating as mv-ac to mv-dc stage of a solid state transformer. *Energies*, 14(15), 2021. 8

[15] Jintong Nie, Liqiang Yuan, Qing Gu, Jianning Sun, and Zhengming Zhao. A Coordinate and Distributed Control Scheme for Multilevel and Multi-Stage Medium Voltage Solid State Transformer. *2018 International Power Electronics Conference, IPEC-Niigata - ECCE Asia 2018*, (51577100):2963–2968, 2018. 8

[16] Mario López, Alberto Rodríguez, Enrique Blanco, Mariam Saeed, Ángel Martínez, and Fernando Briz. Design and implementation of the control of an MMC-based solid state transformer. *Proceeding - 2015 IEEE International Conference on Industrial Informatics, INDIN 2015*, pages 1583–1590, 2015. 8

[17] Tommi Laakkonen. *Distributed control architecture of power electronics building-block-based frequency converters.* PhD thesis, Lappeenranta University of Technology, 2010. 9

[18] Paul Dan Burlacu, Laszlo Mathe, Marcos Rejas, Heverton Pereira, Ariya Sangwongwanich, and Remus Teodorescu. Implementation of fault tolerant control for modular multilevel converter using EtherCAT communication. *Proceedings of the IEEE International Conference on Industrial Technology*, 2015-June(June):3064–3071, 2015. 9

[19] Hua Geng, Shuzhen Li, Chao Zhang, Geng Yang, Lei Dong, and Babak Nahid-Mobarakeh. Hybrid communication topology and protocol for distributed-controlled cascaded H-bridge multilevel STATCOM. *IEEE Transactions on Industry Applications*, 53(1):576–584, 1 2017. 9

[20] Raimarius Delgado, Byoung Wook Choi, and Hwachang Song. Application of EtherCAT in Microgrid Communication Network: A Case Study. *2018 International Conference on Platform Technology and Service, PlatCon 2018*, pages 1–6, 2018. 9

[21] Jan Henrik Fey, Frank Hinrichsen, Gyde Carstens, and Regine Mallwitz. Development of a modular multilevel converter demonstrator with EtherCAT communication. *Proceedings - 2019 IEEE 13th International Conference on Compatibility, Power Electronics and Power Engineering, CPE-POWERENG 2019*, pages 1–6, 2019. 9

[22] C. L. Toh and L. E. Norum. A high speed control network synchronization jitter evaluation for embedded monitoring and control in modular multilevel converter. *2013 IEEE Grenoble Conference PowerTech, POWERTECH 2013*, 2013. 9

[23] H. H.H. De Silva, D. K.J.S. Jayamaha, and N. W.A. Lidula. Review on design and control of solid state transformer based microgrids. *AIMS Energy*, 7(6):901–923, 2019. 9

[24] C. L. Toh and L. E. Norum. A performance analysis of three potential control network for monitoring and control in Power Electronics converter. *2012 IEEE International Conference on Industrial Technology, ICIT 2012, Proceedings*, pages 224–229, 2012. 9

[25] Paul Dan Burlacu, Laszlo Mathe, and Remus Teodorescu. Synchronization of the distributed PWM carrier waves for modular multilevel converters. *2014 International Conference on Optimization of Electrical and Electronic Equipment, OPTIM 2014*, pages 553–559, 2014. 9

[26] Tomas P. Correa, Luis Almeida, and Francisco J. Rodriguez. Communication aspects in the distributed control architecture of a modular multilevel converter. *Proceedings of the IEEE International Conference on Industrial Technology*, 2018-Febru(Section III):640–645, 2018. 9

[27] C.E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, jan 1949. 11

[28] EtherCAT Technology Group. Ethercat slave stack code (ssc) et9300. 22

[29] Kenneth W. Schachter. The tms320f2837xd architecture: Achieving a new level of high performance. Technical report, Texas Instruments, 02 2016. 24, 45

[30] Beckhoff Automation GmbH Co. KG. Twincat: Automation software. 27

Appendix A

The TMS320F2837xD architecture block diagram

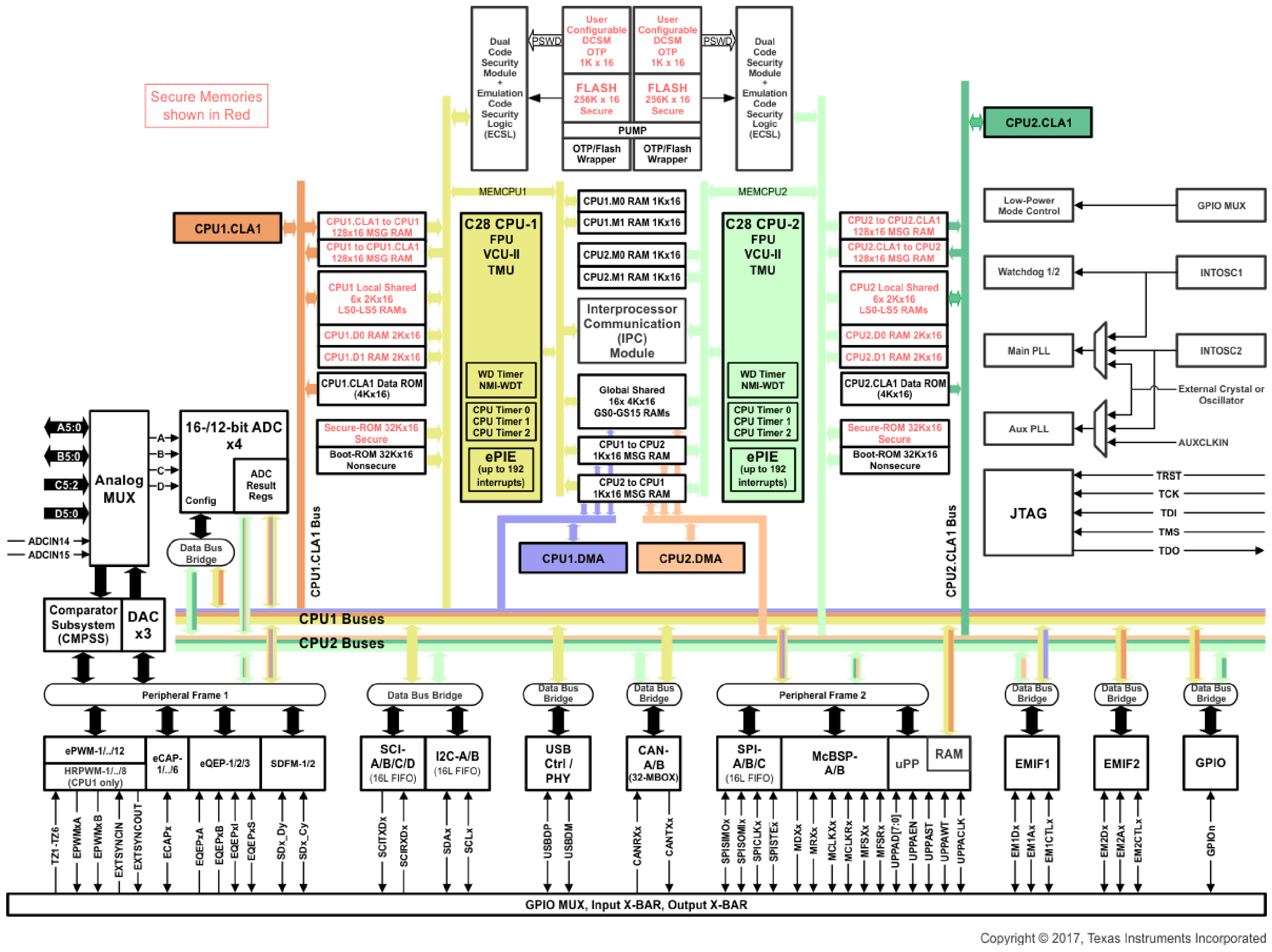


Figure A.1: The TMS320F2837xD dual-core architecture block diagram. [2]