Distributed Real-time Architecture for Mixed Criticality Systems

Integration and Support Report
D 3.4.1

Project Acronym	DREAMS	**Grant Agreement Number**		FP7-ICT-2013.3.4-610640	
Document Version	1.0	**Date**	30.06.2017	**Deliverable No.**	3.4.1
Contact Person	Andreas Eckel	**Organisation**		TTTech Computertechnik AG	
Phone	+43 1 585 34 34 16	**E-Mail**		andreas.eckel@tttech.com	

Contributors

Name	Partner
Thomas Koller	USIEGEN
Obaid Ur-Rehman	USIEGEN
Andreas Eckel	TTT
Cristina Zubia, Félix Casado	IKL
Anton Trapman	ALSTOM

Table of Contents

Contributors 2

Abstract 4

1 Introduction 5

1.1 Structure of the Deliverable 5

1.2 Process for Preparation of the Deliverable 5

1.3 Relationship to other DREAMS Deliverables 5

1.4 Consideration of Requirements 5

2 Cluster Communication Services 6

3 Global Resource Management Services 8

3.1 Introduction 8

3.2 Resource management in avionics demonstrator 9

3.3 Resource management communication 11

3.4 Platform configuration file (PCF) 11

3.5 Global reconfiguration graph 13

4 Security and Safety Services 15

4.1 Cluster-level Security Services 15

4.1.1 Security Services Based on MACsec Implementation 17

4.2 Application-Level Security Services 18

4.3 Security Services in the Avionic Demonstrator 19

4.4 Security Services in the Healthcare Demonstrator 20

5 Safety Communication Layer (SCL) 20

5.1.1 Integration of the SCL in the wind power demonstrator 21

5.1.2 EtherCAT Datalogger module for SCL 22

6 Bibliography 40

Abstract

This deliverable presents the activities related to the technical and operational assistance to the partners involved in the demonstrators regarding integration and maximized utility of the WP3 developments.

The scope of the document is to summarize the work of taskT3.4 which provides the means to support the use of the network services offered by the middleware and components in the aerospace and industrial control demonstrators in WP6 and WP7. It provides information on the technical and operational assistance to the partners involved in the demonstrator development regarding integration and maximized utility of the WP3 developments.

The document introduces and describes the added/improved services as elaborated on in DREAMS. This includes the following services:

- Cluster Communication Services
- Global Resource Management Services
- Security and Safety Services

as well as the Safety Communication Layer.

1 Introduction

This deliverable deals with the real-time architecture that was developed for the DREAMS platform which supports mixed criticality. The architectural approach is based on the service oriented architecture concepts that is already defined in GENESYS, INDEXYS, ACROSS and EMC2 projects. DREAMS now adds services dedicated to cluster communication, resource management and security & safety services in order to address the market demands in automotive-, off highway-, and industry 4.0 industrial domains.

1.1 Structure of the Deliverable

This document is structured by describing the topics addressed by the following main paragraphs:

1) Cluster Communication Services
2) Global Resource Management Services
3) Security & Safety Services

1.2 Process for Preparation of the Deliverable

This document was composed by proposing the initial structure to the team generating the subject inputs. Afterwards, it was filled chapter by chapter by individual partners. The input is strongly based on the information collected from potential customers in the above mentioned industrial domains. Thus it is guaranteed that the approach will be based on existing demand rather than on estimations that might lead to develop results that are not demanded by the markets.

1.3 Relationship to other DREAMS Deliverables

The document build on the results of the other tasks in WP 3, namely the Tasks T3.1, T3.2 and T3.3.

The WP produced the following deliverables, where D3.4.1 is referencing to:

Task 1:

- D3.1.1 High-level Design of Mixed-Criticality Cluster Communication Services
- D3.1.2 First Implementation of Mixed-Criticality Cluster Communication Services
- D3.1.3 Final Implementation of Mixed-Criticality Cluster Communication Services

Task 2

- D3.2.1 High-level Design of Global Resource Management Services
- D3.2.2 First Implementation of Global Resource Management Services
- D3.2.3 Final Implementation of Global Resource Management Services

Task 3

- D3.3.1 High-level Design of Cluster-level Safety and Security services
- D3.3.2 First Implementation of Cluster-level Safety and Security services
- D3.3.3 Final Implementation of Cluster-level Safety and Security services

The structure of the deliverables foresees three documents per Task, a general description of the subject service and two implementation documents, an early document (First Implementation) and a final version (Final Implementation).

1.4 Consideration of Requirements

Building on the dedicated documents of WP 3, the entire set of requirements as identified earlier in D1.1.1. These requirements are addressed and the solutions are implemented to an extent possible at this intermediate stage of the project. The implemented standards and protocols address all the identified requirements.

2 Cluster Communication Services

The DHP stands for DREAMS Harmonized Platform, which is a Xilinx Zynq ZC706 board consisting of an ARM based hard-core CPU and a Xilinx FPGA (together referred to as SoC - System on Chip). The software running on the ARM CPU consists of XtratuM hypervisor (provided by Fentiss) and a demo application provided by TTTech. The demo application is periodically creating and sending Time-Triggered (TT) data frames. The content (data payload) of the data frames consists of identical decimal digits that are being incremented by one for each frame (i.e. 1111111, 22222222, 3333333, then 44444, up to 9999999 and then again 0000000, 1111111, ...). Not only the digits are being incremented, but the amount of these digits too. This means that the first frame contains only 1 digit, the second frame 2 digits, third frame 3 digits, and so on... up to the maximum allowed size of a frame, when the number of digits drops again to just 1 digit. In case of too short frames, the frame is automatically extended by the TTE driver so that the size of the frame is always not smaller than the minimum allowed size.

The application is using the TTE driver for sending the created frames from the ARM CPU to the TTE IP core. The TTE IP core is connected to the ARM CPU by standard AMBA AXI interface via the Network-on-Chip (NoC). In the NoC, a time-triggered extension layer (also known as LRS at on-chip NI in other deliverables) serves as an interface between the ARM CPU, STNoC and other tiles, that can be implemented in FPGA fabric the same way as the TTE IP core and the NoC.

The TTE IP core is the Pegasus IP version 1.6. Since it is configured as clock synchronization master, it sends automatically PCF frames. Only after both synchronization masters are synchronized (i.e. the DHP and PCIe card), only then the TTE IP sends the TT frames further to the network (refer to Figure 1).

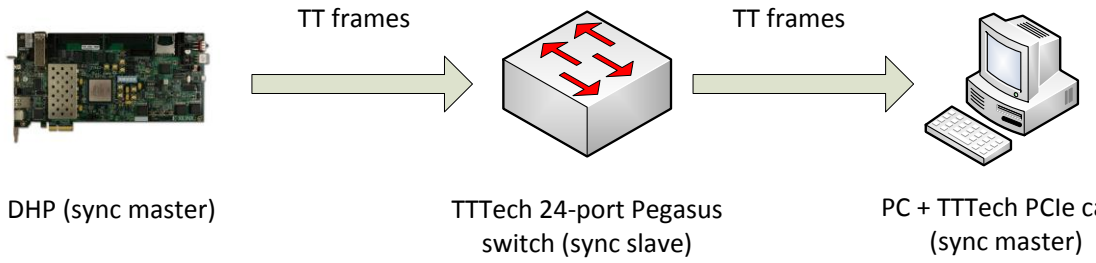


Figure 1: Demonstration set-up

The Zynq ZC706 board has 2 Ethernet ports - one normal port, which is not practically useful for our application because it is driven directly by the ARM CPU and cannot be used directly by FPGA (where the TTE IP core is located); the other port is the SFP port, which is connected to an SFP cage - copper or fiber (optical). Our demonstrator is using the optical fiber connection option.

As of Figure 1, the TT frames are being sent from the DHP to the TTTech 24-port switch, which is configured as synchronization slave. It is resending those frames to the PC equipped with a TTTech PCIe card. Of course, since these frames are TT frames, they must arrive at the switch according to the communication schedule, thus at specific time in order to not become filtered (dropped) by the switch.

The receiver (PC + PCIe card) is periodically checking whether a TT frame is received and it is printing out to the console the actual status. Thus whenever a TT frame is received, it is (including its data payload) printed to the console. If there is no frame received in any of the periods, the application prints out to the console that no frame was received. This way, it can be easily checked whether the TT frames are successfully transferred from the DHP (sender) to the PC (receiver).

Tests proved that all TT frames are successfully transferred over the network within the specified time.

The switch as well as the receiver consist of standard TTTech HW products, while the DHP is actually developed in the DREAMS project. The main idea (AFAIK) is that we are combining our TTEthernet (with PCF frames) + our security solution (MACsec) + XtratuM hypervisor from Fentiss + the TT extension at on-chip NI by University of Siegen into one complex solution. TTEthernet device for ARMv8 architecture

For the purpose of the healthcare demonstrator, a TTEthernet PCI card has been connected to the integrated PCI bus of the ARM Juno development platform in order to use the TTEthernet network. Such an implementation enables a reliable communication between the DHP and the Juno platform by using time-triggered traffic to exchange critical information related to the electrocardiogram data. The TTEthernet card has been mapped to the software partition, which contains Linux/KVM, since TTEthernet drivers are already provided by TTTech for Linux kernel Intel/x86. However, the Juno platform is based on the latest ARMv8 architecture (AArch64), which requires adapting the source code of drivers. Indeed, the TTEthernet drivers for Linux are composed by a kernel module, named 'tt-pci', as well as a set of user-space libraries, which are used to create an application for enabling the communication with the TTEthernet card through the PCI bus. The integration of this software in the Juno board has been achieved by porting the module and the libraries from the Intel/x86 to AArch64. In this context, the source code has been slightly adapted in order to cross-compile it for the ARMv8 architecture (ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-).

Finally, a “Packet Router” program has been developed by using the TTEthernet libraries in order to identify the network packet from the TTEthernet card. Since the critical and non-critical healthcare applications are running inside different virtual machines, it is necessary to route them to the correct operating environment. In this context, the “Packet Router” program is able to differentiate the network packets from the different traffic classes supported by the TTEthernet card, such as Time-Triggered (TT), Rate-Constrained (RC) and Best-Effort (BE), and transfer it to the right virtual machine, as shown in Figure 2.

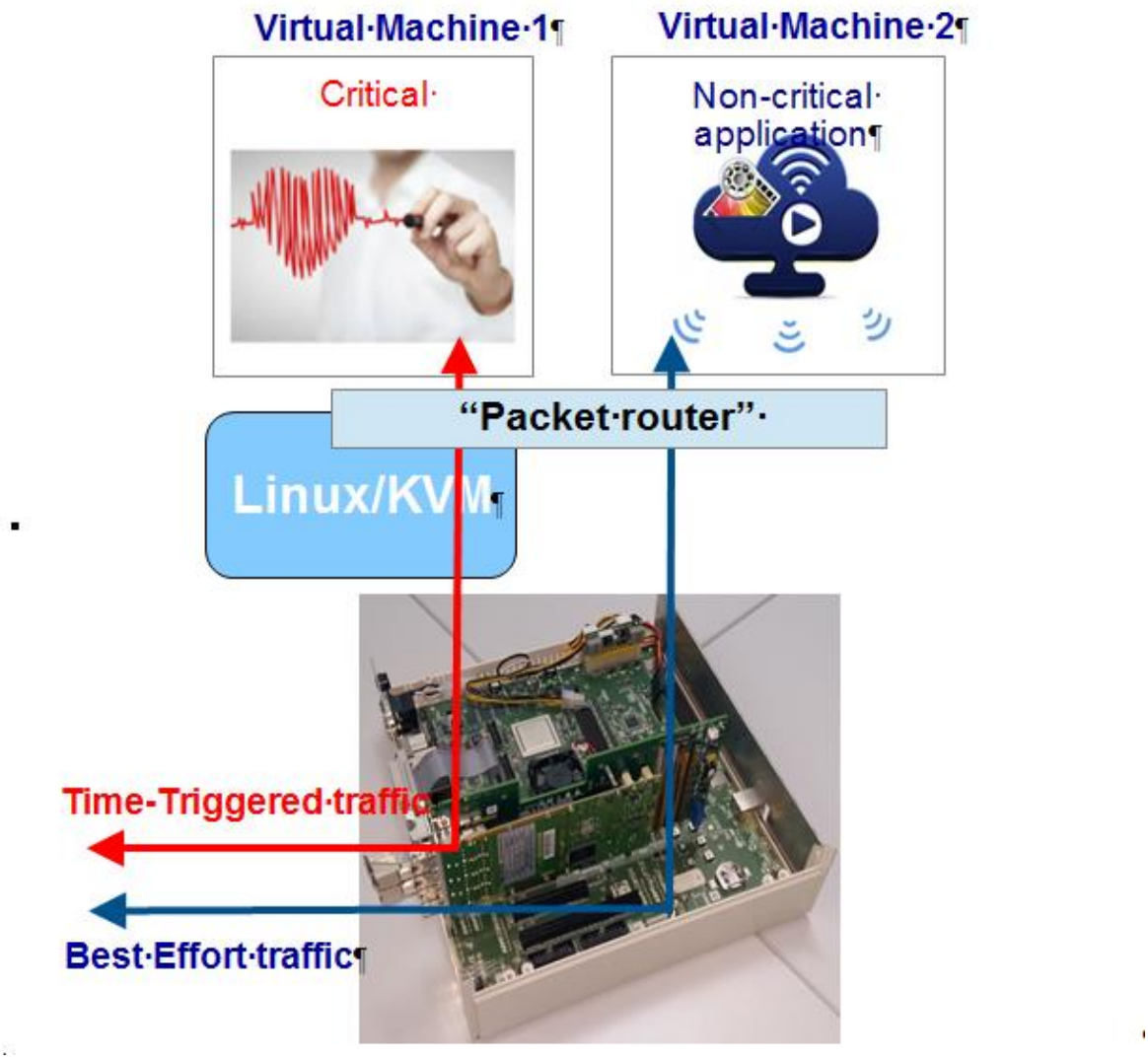


Figure 2: Juno platform and TTEthernet configuration

3 Global Resource Management Services

This section presents the integration and support activities of the DREAMS global resource management services with the avionics demonstrator.

3.1 Introduction

Fulfilling or recognizing system-wide constraints is not possible by viewing a single resource in isolation, but by enforcing a system-wide view, which may require system-wide decisions to be made. The resource management services in the DREAMS platform are realized by a Global Resource Manager (GRM) in combination with local building blocks for resource management. GRM provides services to manage resources globally based on given global offline configuration table. When the conditions, on which the offline configuration is based, change significantly at runtime (for example - core failure), then this information is communicated via the local resource manager to the GRM. In turn, the latter may select a different offline configuration from those provided by WP4.
Local resource management services consist of three major parts: Resource Monitors (MONs), Local Resource Schedulers (LRSs) and Local Resource Managers (LRMs). The MON monitors the resource availability and timing of components (e.g., detection of deadline violations). The LRS performs the

runtime scheduling of resource requests (e.g., execution of tasks on processor, I/O requests) based on the configuration set by the LRM. The LRM either adopts the configuration from the GRM to particular resources (e.g., processor core, memory, I/O) or selects a new configuration from the ones available and reports state of the resource (from MON) to the GRM.

The specification of the integrated resource management architecture, the high-level description of the resource management services and the design of the GRM are covered by deliverables of Task 3.2 (i.e. D3.2.1, D3.2.2 and D3.2.3). The design and implementation of the local building blocks for resource management correspond to Task 2.2 and Task 2.3 of WP2, and are covered in deliverable D2.2.2: "Report on monitoring, local resource scheduling and reconfiguration services for mixed criticality and security with implementation (source code) of low- and high-level monitors, scheduling, security and reconfiguration services supporting mixed criticality and adaptation" in M24, and D2.3.4: "Hypervisor adaptation and drivers for local resource managers" in M33. The specification of a security concept and implementation of necessary security mechanisms to provide trustworthy communication between the building blocks for resource management are realized in Task 3.3: "Cluster-Level Safety and Security" of WP3 and is presented in D3.3.3: "Final implementation of Cluster level safety and security services" in M34. Resource management services are integrated in the Avionics demonstrator developed in WP6.

3.2 Resource management in avionics demonstrator

The avionics demonstrator combines critical applications with non-critical applications using heterogeneous multi-core platforms, connected using a wired network. Five different applications will be deployed in the demonstrator, three critical and two non-critical. All the applications, but the panels, sit on top of the DREAMS middleware and the XtratuM hypervisor, and they are deployed in two different hardware platforms: the Freescale T4240QDS and the DREAMS Harmonized Platform. Communications between the different hardware platforms are ensured by a TTEthernet network.

The avionics demonstrator takes advantage of Resource Management (RM) services to improve the system reliability and the multi-cores performance usage:

- Core failure management, to improve system reliability requirements.
- Deadline overrun management, to satisfy time isolation requirements on multi-cores.
- QoS management, to improve multi-core performance usage.

Some design choices have been made for RM for use in avionics demonstrator as described below.

1. Overall RM design choices:
    - We consider that DREAMS middleware relies on time and space partitioning which is implemented at the chip-level by the XtratuM hypervisor, a technology involved in the project. Therefore, applications will be executed within a set of partitions. A partition is defined by one or multiple slots, each with a start time and a length. Inside a slot, several tasks can be executed.
    - In an embedded system software stack the DREAMS RM services lay on top of the system hypervisor and below the applications.
    - The RM services are executed at system level because they require access to system level hypervisor calls, like suspending partitions (XM_suspend_partition(partitionId)) and changing the hypervisor scheduler plan (XM_switch_sched_plan(newPlan, *currentPlan)).
    - The services are developed to limit the impact on application development, thus their applications interface is limited.
    - Security mechanisms are used to protect the systems resource management services.

2. GRM design choices:

- The GRM runs on the DREAMS Harmonized Platform.
- It is assumed that the core on which the GRM executes is fail-safe. (Justification: GRM only makes global reconfiguration decisions when necessary, but it is not required for the continuous operation of the system. Thus, in case of GRM failure, the overall system dependability is not compromised as the system will still keep on executing; just no new global reconfigurations will be possible. Thus, this failure is not considered)
- The GRM runs in its own XtratuM partition (no other applications or tasks can be executed in this partition).
- The GRM runs as a critical application.
- The GRM must be informed via an update message of the node's current configuration, as this information is used by GRM to select new global configuration if required.
- GRM stores the global reconfiguration graph.
- Interactions with the GRM is done only through configuration options without any direct interaction with the applications.

3. LRM design choices:
   - LRMs run on all nodes.
   - All LRMs must execute on each core at end of every MaC[1]. All LRMs on a node are synchronous, and only one amongst them is a master (per node). Only the master LRM can communicate directly with the GRM.
   - All master LRMs must send one update message to the GRM every MaC. The GRM will consider the node of the corresponding LRM dead (i.e., all cores have failed) in case of failure to receive an update message in a MaC.
   - There must be a MON partition on each core executing in each MaC to monitor the core health status.
   - LRS is dispatched at the beginning of each user-application slot. For critical applications, the tasks are called in sequence in each slot and there is no preemption.
   - Local adaptations are due to internal deadline overrun of critical applications. More precisely, maximal deadlines of each task executing in a partition slot may be specified by the user. In that situation, MON is in charge of monitoring those internal deadlines and if there is an overrun, the LRM immediately stops the best-effort applications.
   - LRM takes a local reconfiguration decision at the end of the MaC by collecting all failed cores. This entails that several failures may happen during a MaC and decision could consider multiple failures. To avoid non deterministic decisions, we impose reconfiguration graphs to be symmetric

4. Design choices related to reconfiguration strategies:
   - Reconfiguration changes, whether they are local or global, are computed offline and only occur in case of permanent core failures.
   - The reconfiguration strategy follows two rules in case not all applications could be locally hosted after some failure(s) on a node: critical applications are locally reconfigured in priority and complete applications must be moved, i.e. an application cannot run on two nodes at the same time.

Figure 3 shows the use of resource management in avionics demonstrator. Critical applications are shown in red color while non-critical applications are in green.

[1] Major Cycle (also known as MAF)

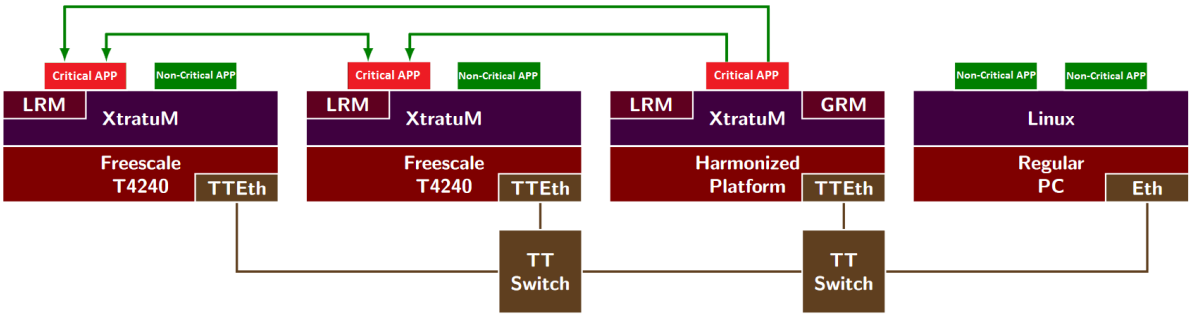


Figure 3. Resource management in avionics demonstrator

3.3 Resource management communication

As explained in the deliverables D2.2.2 and D3.2.3, we need two communication channels between GRM and each LRM:

- Orders channel – From GRM to LRM(s) for sending re-configuration orders.
- Updates channel – From LRM(s) to GRM for sending re-configuration requests and status updates.

More information about the channels is presented in Table 1.

Channel Name	Source	Destination	Time-Triggered	Type
Orders	GRM	LRM(s)	Yes	Sampling
Update	LRM(s)	GRM	Yes	Queuing

Table 1. RM Communication Channels

In the avionics demonstrator, 12 order channels and 12 update channels are present between the DHP and each T4240 (12 cores with one LRM each). At any point in time, information exchange occurs between master LRM(s) on each node and GRM. The channels between other (non-master) LRMs are not used. In case the master LRM fails, another LRM is assigned as master (in pre-decided order). The GRM will then use the communication channels that correspond to the new master LRM.

To support RM communication, corresponding virtual links must be added to TTEthernet configuration files.

3.4 Platform configuration file (PCF)

During the course of the project, it was decided to use a new configuration file based on YAML to make configuration of the platforms manageable. More details and an example PCF can be found in deliverable D6.3.1. Resource management configuration define in the PCF is used to automatically generate the LRM/GRM/MON partitions and extend the configurations of the user partitions (as they are built using the LRS partitions). Since the initial definition, there have been changed to the PCF for declaration of RM services, especially GRM as mentioned below.

```
lrm_desc:
  # Memory space that will be used to perform the communication between
the different resource management partitions
  shared_mem:
    # Start address
    start: 0x10000000
    # Size of the resource management partitions communication area
    # This value is currently user defined, but a good heuristic to
    set it up might be something like: ceil(number of partitions in
    the system including resource management partitions) * 2
    megabytes), where ceil is the smallest power of 2 bigger or
    equal than the given value
    size: 32
  # Start address of the memory area that will be used to map the
  resource management partitions (lrm/mon/grm).
  #*Note*: the memory area size required for all the resource
  management partitions is not defined by the user directly, but can
  be computed as follows:
  (lrm.memory_area_size + mon.memory_area_size) * hw_desc.num_cores
  + grm.memory_area_size
  physical_memory_addr: 0x12000000
  # Security configuration for the communication between the lrms
  and the grm
  security:
    # Security level of the update, order and membership channels
    # level -1: skip security library
    # level 0-3: use security library with appropriate level
    #Fixed value 3 for Avionics use case
    update_ch: 3
    order_ch: 3
    #Note: Membership channel doesn't exist anymore
  # Local resource management (lrm) partitions configuration
  lrm:
    # Memory area size of each lrm partition
    memory_area_size: 2
    # XtratuM partition description list of flags to apply to all
    the lrm partitions
    part_desc_flags: [ fp, system, boot ]
    # Name of the console device to use for all the lrm partitions
    part_desc_console: 'Uart'
    # RM Communication configuration parameter - max queue depth for
    update channels
    max_no_messages: 32
    # LRM NODE ID = 0 for Node with GRM on it. LRM NODE ID = 1 for
    LRM on T4240_1 and LRM NODE ID = 2 for  LRM on T4240_2
    LRM_Node_ID: 0
    #Node ID and TTE_ports of LRMs not on the same node as GRM
    should correspond to
    #values in grm.TTE_ports.order_ports and
    grm.TTE_ports.update_ports
    #If GRM is not on the same node as LRM, then LRM needs TTE ports
    to be defined as below. TTE ES port IDs are in Master LRM order.
    #TTE_ports:
```

```
  # PortIDs can be obtained from TTE tools.

        # order_ports:
                # - {tte_ap_id: 1, ports:
                                [8,11,12,13,14,15,16,17,18,19,9,10]}
        #update_ports:
                #- {tte_ap_id: 2, ports:
                                [8,16,24,32,40,48,56,64,72,80,88,96]}
# Monitor (mon or mon_cf) partitions configuration
mon:
  # Memory area size of each mon (mon_cf) partition
  memory_area_size: 2
  # XtratuM partition description list of flags to apply to all
  the mon partitions
  part_desc_flags: [ fp, system, boot ]
  # Name of the console device to use for all the mon partitions
  part_desc_console: 'Uart'
# [Optional] Global resource management (grm) partition
configuration (only one system in a distributed system will
contain this configuration)
grm:
  # Memory area size of the grm partition
  memory_area_size: 4
  # XtratuM partition description list of flags to apply to the
  grm partition
  part_desc_flags: [ fp, system, boot ]
  # Name of the console device to use for the grm partition
  part_desc_console: 'Uart'
  TTE_ports:
  # PortIDs can be obtained from TTE tools.
  # TTE ES port IDs are in Master LRM order.
       order_ports:
       #1st T4240
                - {tte_ap_id: 1, ports:
                                [8,11,12,13,14,15,16,17,18,19,9,10]}
       #2nd T4240
                - {tte_ap_id: 3, ports:
                                [8,11,12,13,14,15,16,17,18,19,9,10]}
        update_ports:
        #1st T4240
                - {tte_ap_id: 2, ports:
                                [8,16,24,32,40,48,56,64,72,80,88,96]}

        #2nd T4240
                - {tte_ap_id: 4, ports:
                                [8,16,24,32,40,48,56,64,72,80,88,96]}
```

3.5 Global reconfiguration graph

The global reconfiguration graph has been defined in D3.2.3. Some changes have been made since then.

When the last core in a node fails, the GRM receives no new update / reconfiguration request message from the last LRM of that node, and thus knows that the complete node has failed. However, without a message, the GRM cannot look up a new configuration for this case. To resolve this problem, we

reserve some message values which correspond to such events. If ‘n’ nodes are present, then the messages 0,1,2,..,n-1 message values are reserved for the events “failure of the last core of node”.

Example: Consider a case where there are three nodes. Except one core on the second node, all other cores have failed on this node. The LRM on the functional core of the second node sends periodic update messages to the GRM. If this core fails as well, the GRM will not receive the update message anymore. In such an event, the GRM will look for a new configuration with message value ‘1’, corresponding to the failure of second node.

4 Security and Safety Services

This section describes the support for the integration of the security services and the safety services. The security services for cluster level communication are separated into three parts: the security services for off-chip communication, the secure time synchronization and the security services for application-level communication. The security services for off-chip communication and the secure time synchronization are provided on cluster-level to secure the off-chip communication. The security services for application-level communication secures the communication between the applications on end-to-end basis.

The description of the different security services were described in [13] [14] [15]. In this deliverable, the actual implementation for the demonstrators is performed.

4.1 Cluster-level Security Services

The cluster-level security services are implemented in the TTEthernet gateways and in the TTEthernet switches (“off-chip routers”) as shown in Figure 4.

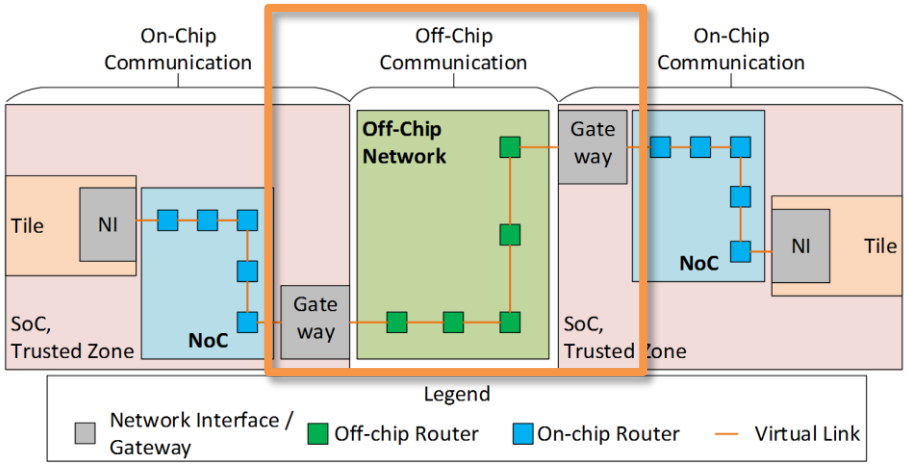


Figure 4: Components of the cluster-level security services

To provide these services, TTEthernet is extended by MACsec [13] [14] [15]. MACsec provides confidentiality, integrity and authenticity of data origin and guarantees a connectionless data integrity and is robust against replay attacks as well as denial-of-service attacks up to some extent. It has to be implemented in every gateway and switch, as it is an OSI layer 2 service and secures only the link between the components (Figure 5).

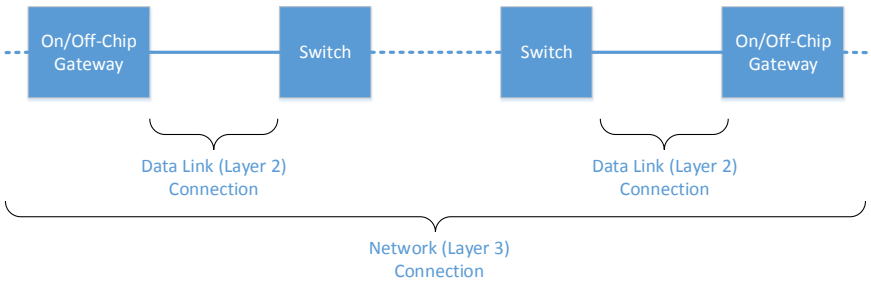


Figure 5: Layer 2 connection

On every link, two Secure Channels (SC) are implemented for MACsec. The each SC is a unidirectional channel that is identified by a Secure Channel Identifier (SCI). This SCI is used to distinguish the different links, and to calculate and select the right keys.

For each channel, there is a set of different keys. A Secure Connectivity Association Key (CAK) is pre-shared between the two components being connected by the link. This CAK is used to authenticate each other by exchanging a message that is verifiable when possessing the CAK. Additionally, the CAK is used to derive sub keys, i.e., an Integrity Check Value Key (ICK) and a Key Encrypting Key (KEK). The ICK allows the generation of the authentication message to check the possession of the CAK. The KEK is used to encrypt the Secure Association Key (SAK) that is used as the session key for the actual MACsec frames. The used key hierarchy is shown in Figure 6. The calculation of the keys is performed on each gateway or switch for each link.

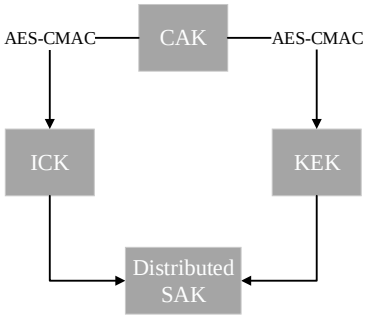


Figure 6: MACsec key hierarchy

Figure 7 shows the usage of the different SAKs for sending and receiving frames in the gateway. The realization in the switches is similar.

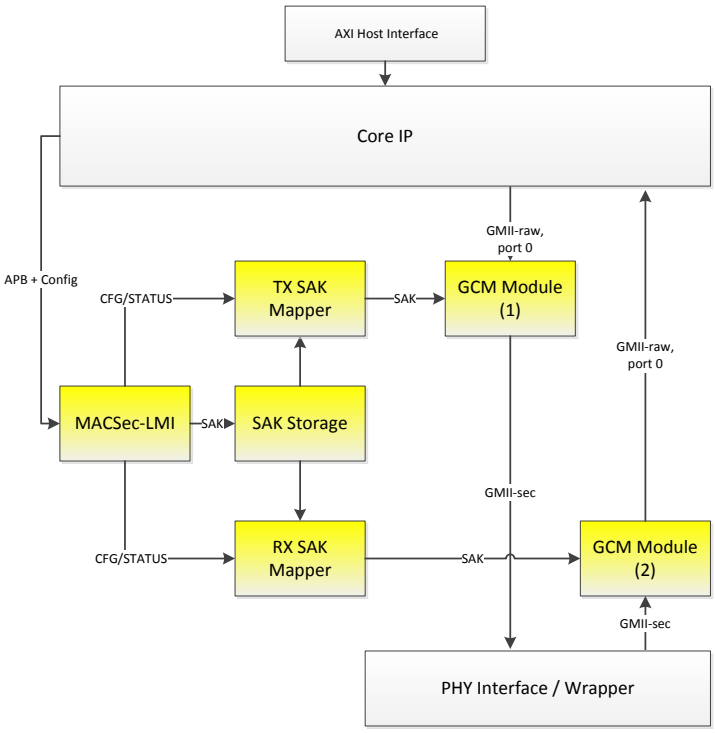


Figure 7: MACsec block diagram at the off-chip gateway

The SAE AS6802 synchronization protocol [17] is used for the off-chip time synchronization. SAE AS6802 uses Protocol Control Frames (PCFs) to exchange synchronization information between the network participants. To secure these PCFs using MACsec, the jitter of the encryption/decryption process and the Integrity Check Vector (ICV) generation and check process has to be low. MACsec provides confidentiality, integrity and authenticity for the time synchronization service. As shown in Figure 7, the blocks that affects the jitter in a gateway (and in a switch) are the GCM-Modules. As the SAKs can be pre-calculated, their calculation is not relevant for the jitter.

4.1.1 Security Services Based on MACsec Implementation

The final solution of the implementation will be the same as described above, except for the fact that all frames (including PCF frames) will be protected by the MACsec. MACsec is the IEEE 802.1AE standard for authentication and encryption of Ethernet data packets transmitted between units capable of handling MACsec authenticated and encrypted data packets..

In order to provide security properties for TTEthernet, TTEthernet (or more generally Deterministic Ethernet) had to be extended with a security solution. For this purpose, IEEE 802.1AE (MACsec) has been selected as the preferred means to provide these message authentication and message encryption services. The principle of MACsec operation is depicted in Figure 8. SecTag the EtherType is set to MACsec 0x88e5. The Ethernet's frame user data including the optional VLAN tag are encrypted. The ICV (integrity check value) ensures the integrity of the MAC Destination Address, MAC Source Address, SecTAG, and User Data. MACsec allows to use different encryption techniques, but defines as default AES 128. Thus, our MACsec implements AES 128[2] (Advanced Encryption Standard). The actual encryption process and process to generate the ICV is depicted in Figure 9.

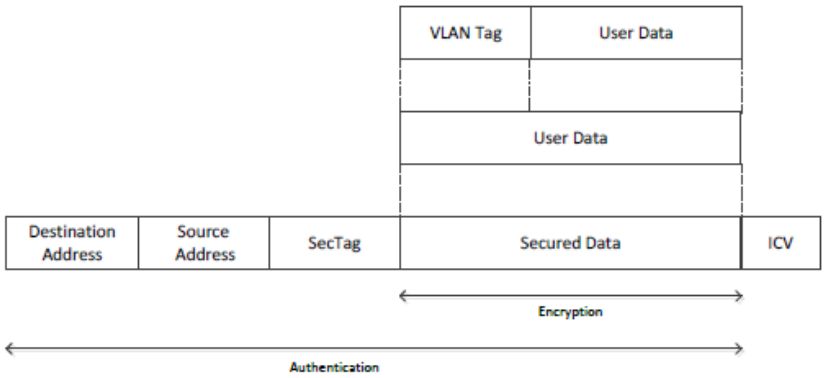
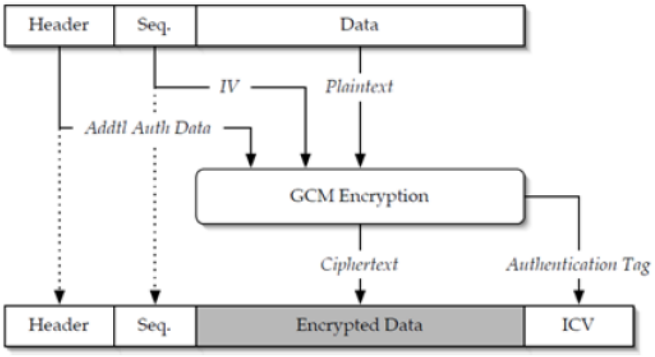


Figure 8: Principle of MACsec operation



[2] See https://de.wikipedia.org/wiki/Advanced_Encryption_Standard

Figure 9: Principle of the Encryption

Thus the frames will be encrypted and tagged with ICV (Integrity Check Value) for authentication in the DHP End System, then these frames will be sent to the switch, the switch will decrypt the frames and check integrity (authenticity). The switch will then process the frames as usual (i.e. TT frames are evaluated and eventually filtered, PCF frames are used for clock sync). The frames are encrypted and sent to the receiver ES (the PC with PCIe card), which will decrypt the frames and check their authenticity.

Consequently the test scenario is the same as without MACsec implementation – just encryption & authentication tag insertion is performed whenever any frame is sent from any device to wire + decryption & authentication (ICV check) is performed whenever any frame is received.

The security approach based on MACsec is working “hop-by-hop”, not “end-to-end”.

We can also perform a second test scenario, which is exactly the same as the first one, except for the fact that the frames would not be encrypted, rather only tagged with ICV and authenticated. Thus the data would be accessible to a potential attacker for reading but the attacker still could not modify the frame because if the attacker modifies content of the frame, ICV check will notice that the frame was modified. It is similar to CRC failure, however CRC is used for safety reasons, not for security because the attacker can change content of the frame, calculate the new and correct CRC and then the received frame looks OK, but attackers cannot do this with ICV tag because they are not able to recreate a new ICV tag with correct value, since for generation of ICV tag one must know and use the correct key value. In our code, we have a simple binary parameter that is either set to 1 (true) or 0 (false). We can easily turn ON and OFF the encryption/decryption feature, while the remaining parts of the MACsec protocol are still used (i.e. MACsec header and ICV tag).

The status of the current work implementing the MACsec is that the MACsec is finished, tested and working in End Systems, which is probably more important since the main scope of the DREAMS project is the DHP platform, which is End System.

The encryption is symmetric, which means that there is only one common key that is used for all encryptions/decryptions within the whole network. Since MACsec does not include any solution for key distribution, this feature is not implemented at all. Instead of that, the key is “hardcoded” in devices, which is sufficient for prototypes that are supposed to prove that the MACsec works with.

4.2 Application-Level Security Services

The security library described in [14] and [15] extends the end-to-end communication services of the hypervisor providing a secure communication. The security library is implemented as a sublayer between the applications and the hypervisor XtratuM. Figure 10 shows the integration of the security library.

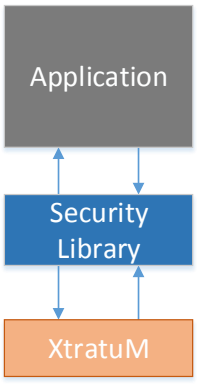


Figure 10: Security library

Originally, it was designed for secure resource management communication. The extension of the security library to support applications of the avionics demonstrator required modifications in the data handling of the payload and the checksums.

The different endianness of the ARM-based DREAMS Harmonized Platform and the PowerPC-based T4240 used in the avionics demonstrator requires a conversion of the protocol data. Converting only the endianness before transmitting the frame would result in an incorrect checksum as both header and payload are protected. Additionally, a correct decryption has to be ensured. Figure 11 shows the affected field in the header.

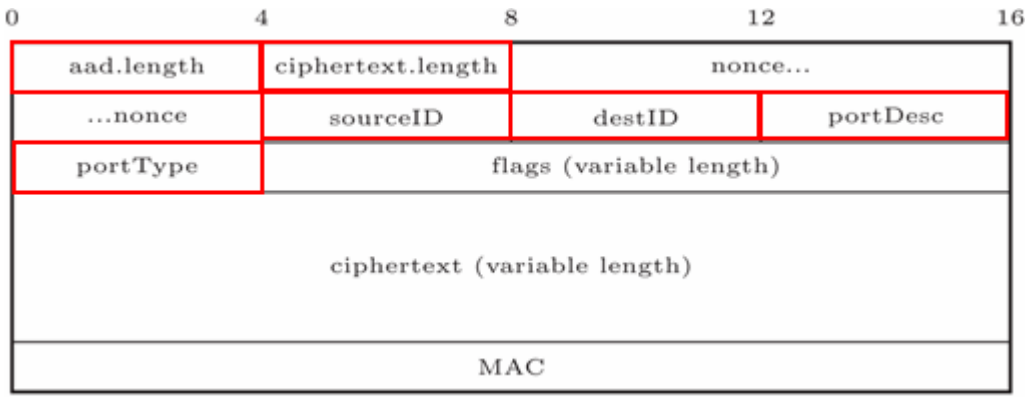


Figure 11: Security level 3 frame

4.3 Security Services in the Avionic Demonstrator

The avionic demonstrator uses all of the cluster-level and application level security services. Figure 12 shows the avionics setup. There are two Freescale T4240, the DREAMS Harmonized Platform and a regular PC, connected by two TTEthernet switches.

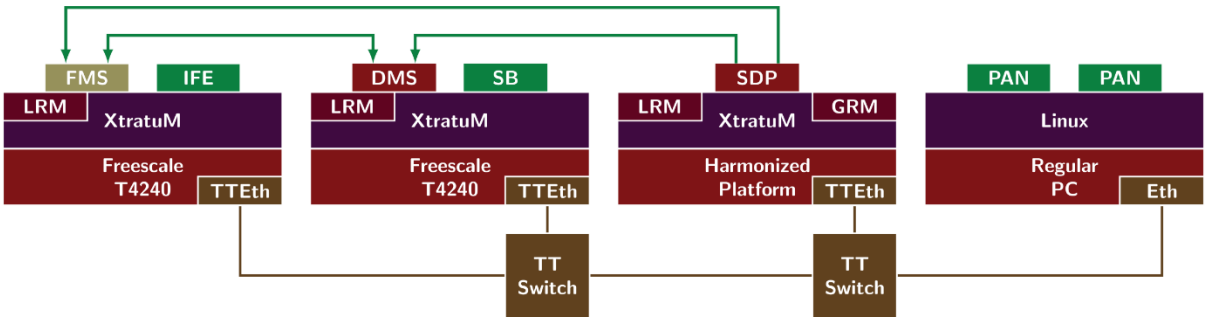


Figure 12: Avionic use case

The cluster-level security services, i.e., the secure communication service and the secure time synchronization service are used on the TTEthernet connections between the T4240s and the first TTEthernet switch, between the DREAMS Harmonized Platform and the second TTEthernet switch, and between the two switches. The connection between the second TTEthernent switch and the regular PC is a normal Ethernet link. Hence, on this connection, no security services are used.

Fife different applications implemented in the demonstrator. The critical applications are a Flight Management System (FMS), a Display Management System (DMS) and a Sensors Data Provider (SDP). The non-critical applications are an In-Flight Entertainment system (IFE) and panels. There are two

panels, one displays information from the IFE and the other one displays information from the DMS. In addition, there are some stressing benchmarks (SB) which are also non-critical applications.

The application-level security services are used to secure the communication between these applications. The security library provides different security levels so that the required security services can be selected. The different security levels for the communication channels are defined in the following:

- FMS -> DMS: Security Level 3. Integrity and authenticity is required. The communication might contain sensitive information. Hence, confidentiality also is required.
- DMS -> FMS: Security Level 3. Integrity and authenticity is required. The communication might contain sensitive information. Hence, confidentiality also is required.
- DMS -> SDP: Security Level 3. Integrity and authenticity is required. The communication might contain sensitive information. Hence, confidentiality also is required.
- SDP -> FMS: Security Level 3. Integrity and authenticity is required. The communication might contain sensitive information. Hence, confidentiality also is required.
- IMS: Security Level 2. Integrity and authenticity are required. Confidentiality is not required.
- SB: Security Level 3. No security service is required for the stressing benchmarks, but to utilize the highest amount of computation time, the highest security level is used.

4.4 Security Services in the Healthcare Demonstrator

The heath-care demonstrator uses the cluster-level security services. TTEthernet is used between the DREAMS Harmonized Platform (DHP) and the Juno board as shown in Figure 13. So, MACsec secures the connection between these two components (red arrows).

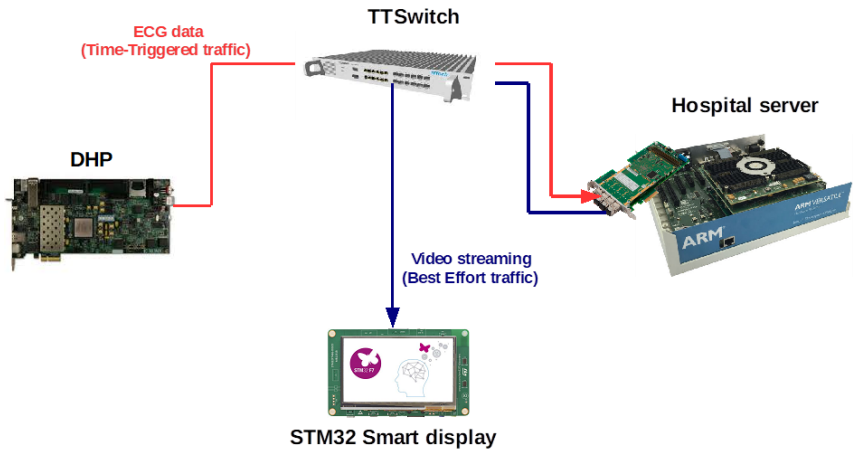


Figure 13: Communication in the health care demonstrator

In addition to the cluster-level security services, a secure monitor firmware is used. The adaptions for the health care demonstrator are described in the demonstrator deliverable D8.2.1 [16].

5 Safety Communication Layer (SCL)

The Safety Communication Layer (SCL) developed in T3.3 is used in the wind-power demonstrator to transport safety related input/output data between EtherCAT slaves and the safety protection system deployed in the DHP. The primary goal of the SCL integration is to enable safety communications. The

integration of the SCL on an existing fieldbus protocol such as EtherCAT has been carried out, providing evaluation of safety features of the developed communication layer, which will be stressed in the evaluation plan by means of the real time fault injection framework developed in T5.2 and described in D5.2.3.

5.1.1 Integration of the SCL in the wind power demonstrator

In order to guarantee data integrity in the data exchange between the EtherCAT node and the DHP, a layer which defines a safety communication protocol has been implemented over the EtherCAT-DHP channel. Figure 14 shows the wind-power demonstrator that has been set up in the scope of DREAMS project.

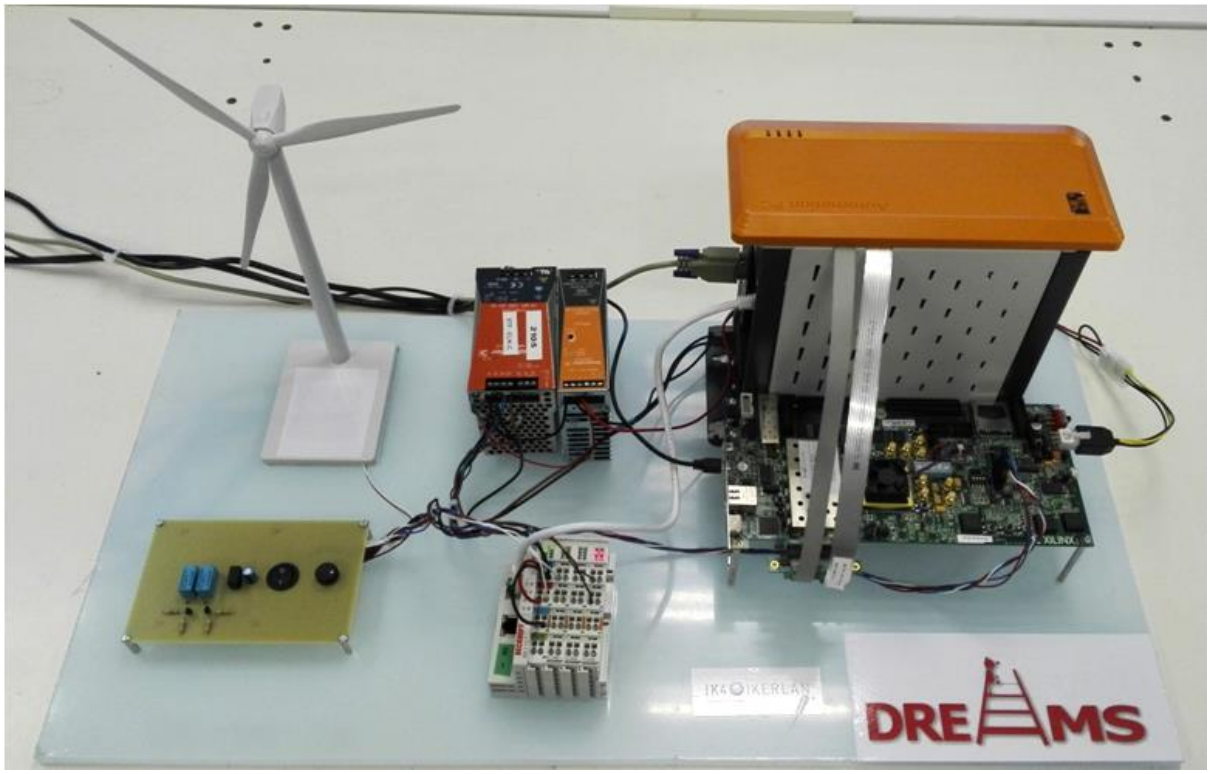


Figure 14: DREAMS wind-power demonstrator.

Beckhoff EtherCAT nodes are used in the demonstrator shown in Figure 14. When SCL is used, these EtherCAT nodes have to be replaced by a PC running Windows with an EtherCAT board attached.

In the scope of wind-power demonstrator, the Galileo control system not only performs real time supervision and control tasks, but it is also a bridge for SCL messages between the EtherCAT ring and the DHP.

The safety data enclosed in SCL messages are processed in the control partition. All EtherCAT variables described in the EtherCAT Network Information (ENI) file are processed by the control application. Each variable has a unique identifier and this way SCL messages can be filtered from the rest of variables exchanged in the EtherCAT ring. First, input variables are read and once a SCL message coming from the EtherCAT ring is identified, it is sent to the DHP by means of a specific driver developed for this purpose (DHP driver). Then, output variables are set. An SCL frame coming from the DHP and targeting the EtherCAT node would be sent as such a variable. To that end, DHP driver is checked in order to know whether any new SCL frame has been sent by the DHP. In case the result to that enquiry is positive, data is read from the DHP and output variable is set accordingly. This process is repeated in a loop every 12 ms, which is the base time for the EtherCAT ring.

In the FPGA design developed for DREAMS wind-power demonstrator, “AXI Memory Mapped to PCI Express” Xilinx IP has been integrated. This acts as the PCIe interface in the DHP and is mapped to the AXI bus. This bus has been later accessed by safety cores in order to exchange safety data by means of SCL frames.

The application logic implemented in the safety cores handles data exchange with Galileo control system. In addition, safety cores are responsible of managing GPIOs connected to the safety relays. SCL logic makes sure the received frame is a valid frame. In case it is a legitimate frame, it is processed in order to check the safety data values sent by the EtherCAT node. If these values exceed the threshold that is required to open the safety line, GPIO output will be set accordingly. Thus, the safety line will be open. Safety line will be also opened when received SCL frame is not valid. In case, the values received in the SCL frame do not exceed the threshold that is required to open the safety line, GPIO output will be set such that the safety line will remain closed.

After GPIO output is set, if there is any SCL frame that needs to be sent to the EtherCAT node, the given frame will be written in the corresponding port.

More details can be found at deliverable D7.2.1.

5.1.2 EtherCAT Datalogger module for SCL

A data-acquisition system is crucial in order to analyse the safety frames interchanged among the EtherCAT slaves and the master. This system is the so called Datalogger module for SCL, and has been developed within T3.4 to support the integration of the SCL in the wind-power demonstrator. Figure 15 shows a scheme of the wind-power demonstrator where all the different actors related to SCL, included the Datalogger, are depicted.

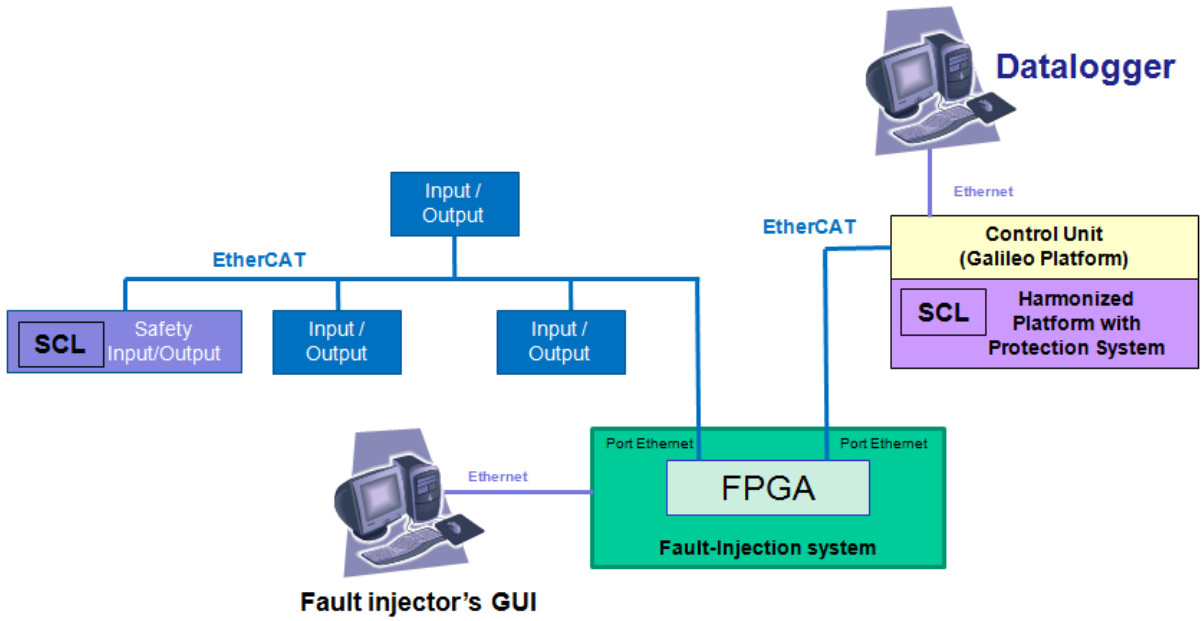


Figure 15: SCL assessment in the wind-power demonstrator

The EtherCAT Datalogger module logs the information of the EtherCAT I/O variables related to SCL. The acquisition system captures the EtherCAT frames from the bus, extracts the value of the variables related to SCL and stores the information in a binary file format.

The input of the system is an ENI (EtherCAT Network Information) file. This file contains information about the network configuration and the variable distribution in the process memory of the EtherCAT

master. The acquisition system captures the EtherCAT frames and extracts the value of the variables using the ENI file for further processing.

The following sections detail the design, development and evaluation of the Datalogger module. In order to better understand the features of the SCL module of the Datalogger, some background information on the SCL API and the EtherCAT network is given first.

5.1.2.1 Background information

5.1.2.1.1 EtherCAT network

A basic EtherCAT network is mainly formed by a master and one or more slaves that exchange information cyclically between them. Thus, master sends outputs and receives inputs from slaves. These outputs and inputs variables are allocated in datagrams, and these datagrams shape the frames that will travel around the EtherCAT network. Usually, EtherCAT networks are deployed where Real-Time features are required, which mean demanding scenarios where the variables exchanged between master and slaves are of great relevance. Therefore, it would be of interest to have access to these network frames so users can be completely aware of what is happening. Thus, by monitoring the exchanged frames, users are able to track when and where takes place an unexpected behaviour. The Datalogger tool allows users to store network frames and extract the variables that travel within them. In this particular case, the Datalogger functionalities have been upgraded in order make the tool capable of monitoring EtherCAT networks. Therefore, by placing the Datalogger between the master and slaves of an EtherCAT network all the variables exchanged between them are extracted from the frames and stored. In Figure 16 a sketch of the explained scenario is shown.

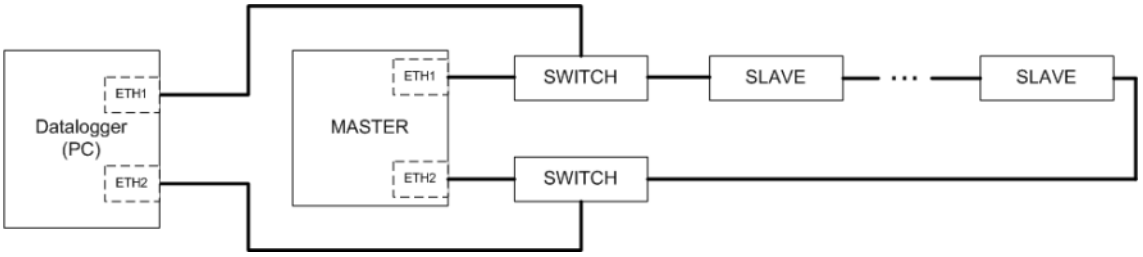


Figure 16: Sketch of the Datalogger deployment scenario

5.1.2.1.2 SCL API (Application Programming Interface)

In this section it is given a brief description of the SCL and the main features that have had impact in the design of the Datalogger. Detail information on SCL can be found in T3.3 deliverables' series.

SCL variables are provided by the SCL, which is located immediately below the Application layer. The main purpose of the SCL is to report the status of the communication to the application, that is, to make the application aware of whether the received data is reliable or not, so that the required actions can be taken. To that purpose, and based on the specifications of the international standard IEC 61784-3-3 [1], the SCL receives the application data to be send and produces a Process Data Unit (PDU); so, the SCL in destination will unpack it in order to check the reliability of the data and to deliver it to the application. As the communication is Host (Master)-Device (Slave) based, the PDU will have a slightly different format depending on which of them operates the SCL. In Figure 17 sketch of a PDU is shown.

According to the standard, the message PDU (Protocol Data Unit) has the following format:

F-Input/Output Data	Status/Control Byte	CRC2
Max. 123 bytes	1 byte	3 or 4 bytes

Figure 17: Generic PDU format

A SCL variable (or a PDU) is therefore formed by the application data (F-Input/Output Data), the information related to the state of the communication (Status/Control Byte) and the CRC (CRC2).

The main difference between the PDU of a F-Host and a F-Device is found in the Status/Control Byte, which are detailed in Annex A. The Status/Control Byte is used by the F-Device/F-Host to provide information about the communication state to its counterpart. These two bytes are different between them; basically, the F-Host uses the Control Byte to send safety-related commands to the F-Device and the Status Byte is used by the F-Device to answer F-Host commands and to report any detected failure that may lead to a safety state. The format of both bytes is detailed below.

Regarding the Datalogger, it might not be of interest to log all the information contained in the SCL variable; therefore, rather than logging the whole variable, just the relevant information is stored, which are as follows:

- **Input Variable:** Data (optional, it will be indicated in the name of the variable), Status Byte (FV_activated, WD_timeout, CE_CRC and Device_Fault), and CRC2.
- **Output Variable:** Data (optional, it will be indicated in the name of the variable), Control Byte (activate_FV), CRC2.

5.1.2.1.2.1 SCL within the wind-power demonstrator

The wind-power scenario consists of at least one EtherCAT slave that uses SCL to encapsulate the application data, forming a PDU. As stated previously, the PDU contains several variables; thus, the Datalogger receives a PDU of 9 bytes and has to be able of splitting it into the desired variables. To this purpose, especial nomenclature is used for SCL variables. Therefore, the name of this kind of variables always starts with *"SCL_D…"* or *"SCL_WD…",* where *D* and *WD* stand for *Data* and *Without Data*, respectively; thus, SCL_D indicates that the 'Data' field of the PDU has to be logged while a SCL_WD indicates the contrary. This way, the Datalogger is able to identify SCL variables and perform the required actions.

5.1.2.2 Datalogger development

Essentially, the Datalogger receives EtherCAT frames, through direct traffic (winpcap library) or from a wireshark capture, and, according to a previous configuration, stores the datagrams and extracts the variables. In Figure 18, a diagram with the inputs and outputs of the system (Datalogger) it is shown.

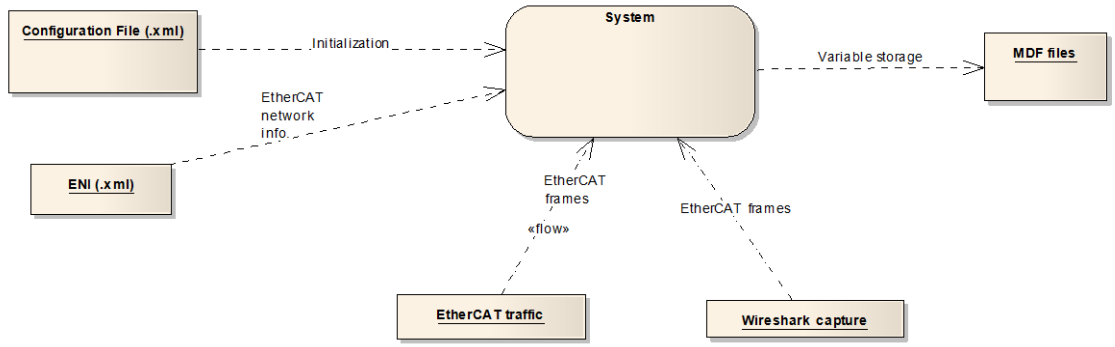


Figure 18: Diagram of the inputs/outputs of the system (Datalogger)

5.1.2.2.1 Datalogger configuration

Before any other action, the configuration file has to be created. This is done using an additional application that generates an xml file (will be detailed later) that the Datalogger uses to configure some parameters before starting with any logging related action. In Figure 19, the Graphical User Interface (GUI) of this application is shown. The fields that users have to fill are detailed below.

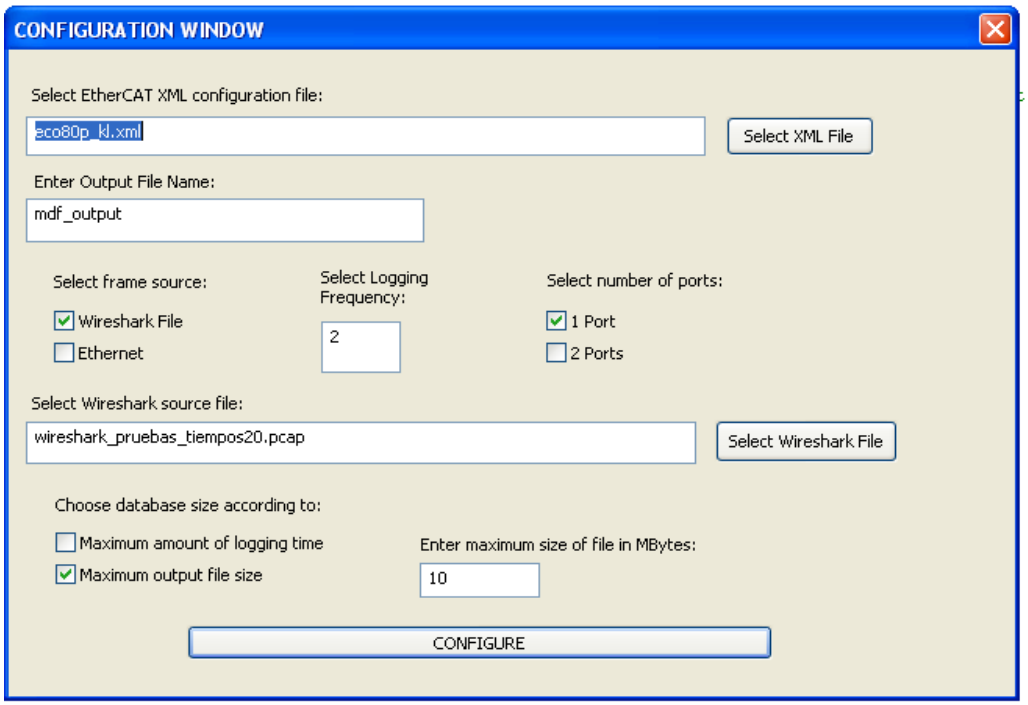


Figure 19: Graphic User Interface (GUI) of the application used to generate the configuration file

- *Select EtherCAT XML configuration file:* the EtherCAT Network Information (ENI) file has to be selected. This file gathers all the information regarding the EtherCAT network that the Datalogger requires to set up the objects that manage the logging actions.
- *Enter Output File Name:* users must introduce the name of the output files generated by the Datalogger.
- *Select frame source:* users are allowed to choose between two sources of frames. On one hand (on "Ethernet" option), it is possible to operate capturing frames through a real bus using winpcap library; and, on the other hand, it is also possible to use, as a source, frames captured by Wireshark.
- *Select Logging Frequency:* the sampling rate is defined. For instance, though the cycle time of the EtherCAT cyclic frames is 2 ms, users may prefer to extract variables with a period of 10 ms.
- *Select number of ports:* if "Ethernet" is chosen as frame source, here users have to indicate the number of network ports. EtherCAT networks are capable of using redundancy; so, by indicating the number of ports, it is determined whether both main and redundancy frames can be processed or only the main frames.
- *Select Wireshark source file:* if "Wireshark File" is chosen as frame source, here users have to select the Wireshark file with the captured frames.

- *Choose database size according to:* users are allowed to choose whether they prefer to limit the output generated files by time or by size.
- *Enter maximum size of file in Mbytes:* here users must indicate the maximum size of the files. If the database size has been chosen according to time, the value introduced here is seconds, otherwise is Mbytes.

In Figure 20, it is shown an example of a configuration file. The meaning of its main fields is explained afterwards.

```
<?EDConfig version='1.0'?>
<EtherCATDataloggerConf>
  <ConfigFile>
    <FileName>EtherCATDataloggerConf</FileName>
    <Description>...</Description>
  </ConfigFile>
  <Autostart>
    <Value>no</Value>
    <!--...-->
  </Autostart>
  <EthercatENIFile>
    <FileName>...</FileName>
    <!-- ENI file with EtherCAT bus configuration -->
  </EthercatENIFile>
  <OutputFile>
    <FileName>mdf_output</FileName>
    <!-- mdf main output file root name -->
  </OutputFile>
  <FrameSource>
    <Type>wireshark</Type>
    <!--...-->
    <WiresharkFile>wireshark_pruebas_tiempos20.pcap</WiresharkFile>
    <NumInterfaces>0</NumInterfaces>
    <Ports>
      <Port>...</Port>
      <Port>...</Port>
    </Ports>
  </FrameSource>
  <DefaultLoggingFreq>
    <DefaultFreq>2</DefaultFreq>
    <!--...-->
  </DefaultLoggingFreq>
  <DatabaseSize>
    <LimitType>size</LimitType>
    <!--...-->
    <TimeLimitValue>0</TimeLimitValue>
    <SizeLimitValue>10</SizeLimitValue>
  </DatabaseSize>
  <VariableFilter>
    <Activated>no</Activated>
    <Type>white</Type>
    <!--...-->
    <DefaultBlackFilter>black_filter.xml</DefaultBlackFilter>
    <DefaultWhiteFilter>white_filter.xml</DefaultWhiteFilter>
  </VariableFilter>
  <Variables>...</Variables>
  <Triggers>
    <ExternalTrigger>
      <SegBefore>1</SegBefore>
      <SegAfter>1</SegAfter>
    </ExternalTrigger>
    <TriggersForVar/>
  </Triggers>
</EtherCATDataloggerConf>
```

Figure 20: Example of a configuration file

- **ConfigFile:**
  - *FileName:* Current configuration file name.

  - *Description:* Brief description related to the configuration file.
- **Autostart:**
  - *Value:* This field indicates whether the Datalogger must start automatically, once the configuration is done, or wait till the user commands it through the start button.
- **EthercatENIFile:**
  - *FileName: P*ath where the ENI file is allocated.
- **OutputFile:**
  - *FileName:* Name used for the output files.
- **FrameSource:**
  - *Type:* indicate whether the frame source is Winpcap or Wireshark
  - *WiresharkFile:* Wireshark file name.
  - *NumInterfaces:* number of network interfaces to be used (used when winpcap is the frame source, otherwise the number is 0).
  - *Ports:* Ethernet interface id of the ports used (used when winpcap is the frame source, otherwise this field is omitted).
- **DefaultLoggingFreq:** frequency at which each variable is logged.
- **DatabaseSize:**
  - *LimitType:* determine which criteria must be followed to create a new output file. Two options are possible: limited by time or size.
  - *TimeLimitValue:* in case the limit type is time, here it is specified the amount of time.
  - *SizeLimitValue:* in case the limit type is size, here it is specified the limit size.

Once the configuration file (named “EtherCATDataloggerConf.xml”) has been generated, it is copied to the directory where the system executable file is allocated, so the configuration file can be used by the Datalogger.

5.1.2.2.2 Datalogger GUI (Graphical User Interface)

The Datalogger provides users with an intuitive GUI, so it can be easily commanded. In Figure 21, a capture of this GUI is shown.

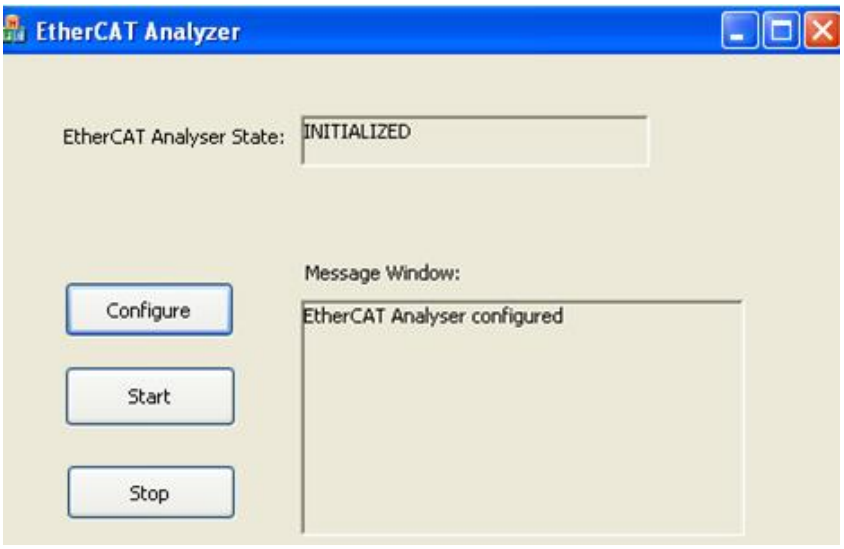


Figure 21: Graphical User Interface to command the Datalogger

As shown in Figure 21, there are two fields that provide information to the user: the *EtherCAT Analyser State* gives information about the current state of the Datalogger, and the *Message Window*, in addition to the tool state, gives any informative message, warning or error in the configuration or performance. Besides, there are three buttons that (i) allow users to generate a new configuration file, (ii) to start logging and (iii) stop if the Datalogger is logging.

5.1.2.2.3 Datalogger engine

The Datalogger engine consists basically of three subsystems: the Cataliser Manager, the Extraction and the Storage subsystems. In Figure 22, a sketch of the Datalogger subsystems is shown.

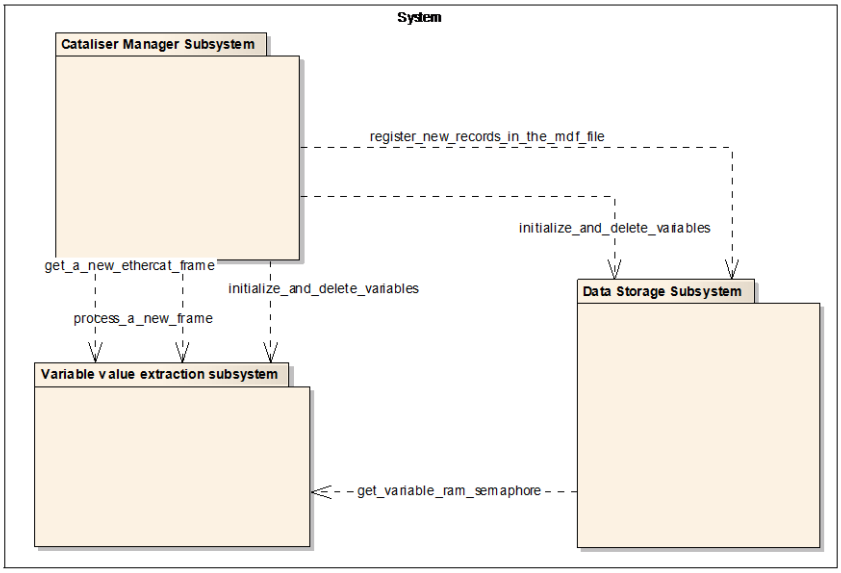


Figure 22: Datalogger subsystems

5.1.2.2.3.1 Cataliser Manager Subsystem

The main task of the Cataliser Manager is to perform the actions related to the commands coming from the GUI (see Figure 21).

When it is commanded to *Configure*, the Datalogger takes the ENI file's name, the output file's name, the frame source and the logging frequency from the "EtherCATDataloggerConf.xml" configuration file. Regarding the frame source, the name of Wireshark file and the ports' number are also taken. Afterwards, the Datalogger configures the structures where the variables and the cyclic commands are stored. These two last tasks are designed specifically for an EtherCAT network and will be explained in detail later.

When it is commanded to *Start*, the Cataliser Manager commands the Storage Subsystem to create the Measurement Data Format (MDF) files configuration, where the extracted variables are written. Next, all the buffers used for storing purposes are reset. Eventually, the Cataliser Manager triggers the storing frames, extracting and storing variables processes by calling to the Extraction and Storage Subsystems.

In case that the button pushed is *Stop*, the Datalogger puts an end to all the logging related actions.

Figure 22 shows the Cataliser Manager Subsystem state machine, where the previous description is put in a schematic and more intuitive way. As it can be observed, the first step, before beginning with any logging related action, is to configure the Datalogger. Afterwards, the tool is ready to start logging. Once the logging activity is finished, either by reaching the end of the file (in case of using Wireshark source) or by users action (pushing the *Stop* button), the Datalogger keeps an idle state waiting for

the next command. Additionally, and providing that there is no logging activity, the tool can be reconfigured by pushing again the *Configure* button.

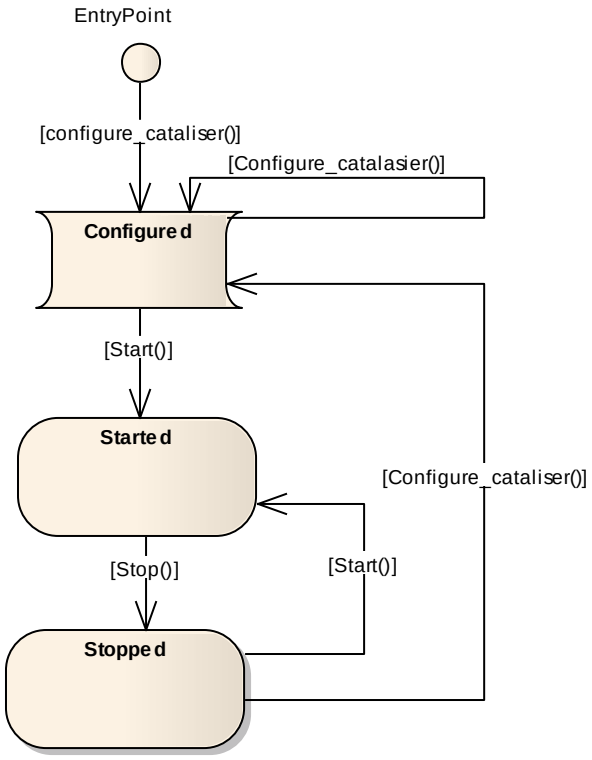


Figure 23: Cataliser Manager Subsystem state machine

5.1.2.2.3.2 Extraction Subsystem

The Extraction subsystem is in charge of obtaining the Ethernet frames and extracts the variables of interest from them. As it has been explained before, depending on the configuration, the frames may come through a network interface or from a Wireshark file, where the frames have been previously captured. In order to meet the particular requirements of EtherCAT, this subsystem is in charge of:

- Storing the datagrams of EtherCAT frames in a circular array.
- Detecting when, by any failure in the network, these datagrams are not complete.
- Finding the complementary datagrams from the redundant frames sent by the EtherCAT Master and combining them with the main datagrams in order to complete them.
- Extracting the variables from the resulting datagrams. As this task is specific for EtherCAT networks, it will be explained in detail later.

In Figure 24, Extraction Subsystem state machine is shown. When the Datalogger receives a new frame, it is stored in the circular array. After a given time (and just once), it is checked whether all the expected cyclic commands have arrived in order to confirm that the used ENI file is the proper one. In addition, it is verified that the frame has not been already processed (when a frame coming from the main port is complete, its redundant frame is discarded). Next, while there are new frames to process, the Datalogger continues storing the datagrams and extracting the variables of interest. When the logging activity has finished, the variable buffer is reset.

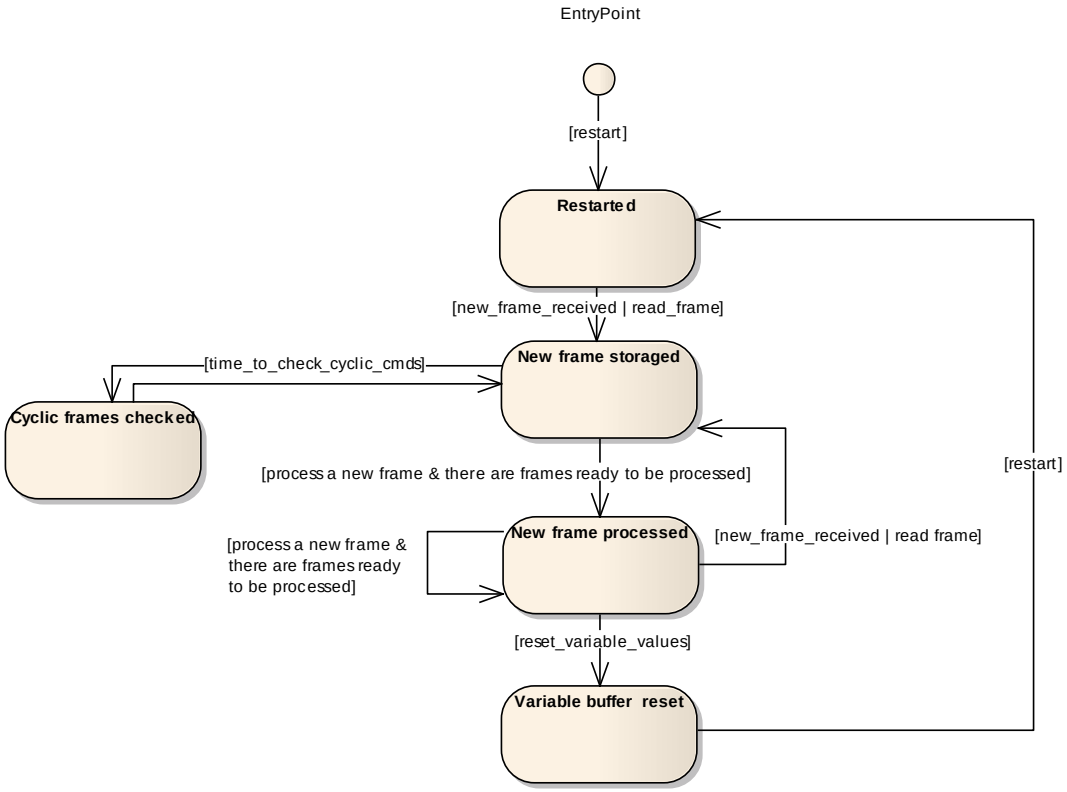


Figure 24: Extraction Subsystem state machine.

5.1.2.2.3.3 Storage Subsystem

The Storage Subsystem is in charge of allocating the extracted variables into a user readable file. To that purpose, this subsystem creates an MDF configuration and writes this configuration and the data extracted from the EtherCAT frames into an MDF file.

MDF is a binary file format that can be used for recording, exchanging and post-measurement analysis of measurement data. The MDF file is composed of a series of blocks. Each block consists of a number of contiguous Bytes and can be seen as a record or as a structure of data fields. There are different types of blocks. Blocks can include pointers to other blocks that are stored in a data field of type LINK. A link is an absolute Byte position within the file, starting at the beginning of the file. Thus, a normally tree-like hierarchy of blocks is formed. So, the specific configuration for the Datalogger is fashioned as follows: one Data Group[3] formed by several Channel Groups[4]; and, each Channel Group shaped by different Channels[5]. There are three main types of Channels, one regarding the time, a second referring the reliability of the data (that is, indicating whether the recorded data is fully reliable or not), and others regarding the data of all the variables that go within the same datagram.

Additionally, this subsystem manages the limit of the MDF files (depending on the configuration this limit is reached by time or by size). Therefore, when the limit is reached, a new MDF file with the nomenclature “OutputFile_xxxx.mdf” (where “xxxx” is the number of the file: 0000, 0001, 0002, etc.) is created. In Figure 25, the Storage Subsystem state machine is shown. As it can be observed, before starting to record data, the MDF configuration is done. Afterwards, while there are variables to log and the Datalogger is not commanded to stop, the Storage Subsystem continues writing the data into the MDF file.

[3] Data Group: description of data block that may refer to one or several channel groups.

[4] Channel Group: a set of signals which are always measured jointly.

[5] Channel: plain data regarding a certain variable (time, reliability, logged variables).

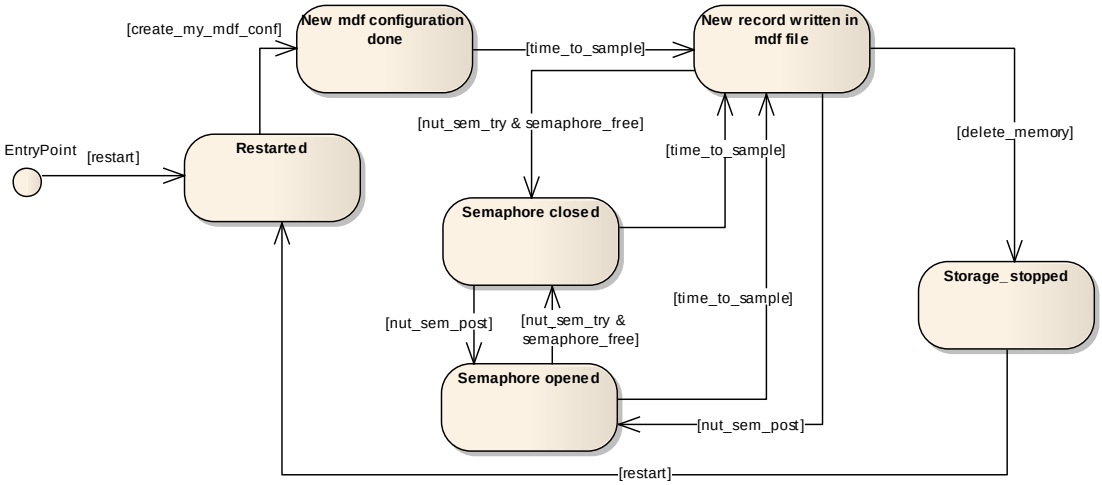


Figure 25: Storage Subsystem state machine.

5.1.2.2.4 Datalogger particular specifications for EtherCAT networks

At this point the basic structure of the Datalogger has been explained. In the subsystems description, two tasks have been highlighted as the most significant ones among all the tasks performed by the Datalogger, which are the configuration process and the variable extraction. Therefore, in the following subsections they are explained in detail.

5.1.2.2.4.1 Configuration Process

As it has been explained in subsection 5.1.2.2.3.1, when the Datalogger is commanded to *Configure* (through the GUI), it takes several parameters from the "EtherCATDataloggerConf.xml" configuration file and, in addition, it shapes the structures that allocate the variables and configures the information related to the cyclic commands. The relation between the variables and the cyclic commands is also determined, that is, the Datalogger tool is aware of in which datagram and position travels each variable.

EtherCAT technology uses a dual-port memory so the application delivers the data to one side of the memory, while the data received from the medium is allocated at the other side (see Figure 26). The cyclic commands, which are mainly a piece of this dual-port memory, contain a certain number of variables, each with its offset and length. Thus, once the offset and the length of the piece of the memory that forms the datagram are identified, the relation with the variables can be obtained.

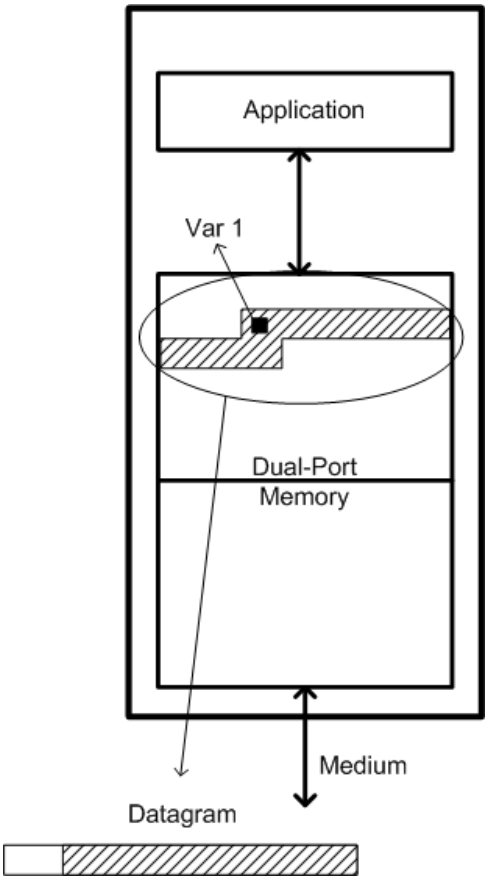


Figure 26: Sketch of the EtherCAT dual-port memory

The ENI file provides the required information to accomplish the task of linking the cyclic commands with the variables. On one hand, the relevant information regarding the cyclic commands is:

- Addr: logical address.
- Cmd: identifies the command type, which may be 10 for logical memory read (LRD) and 11 for logical memory write (LWR) and the logical address. As in this case the system is prepared to work with redundancy, the command type 12 (logical read write, LRW) is not used.
- DataLength: defines the length of the cyclic command data length.
- InputOffs: indicate the position where the input variables are allocated.
- OutputOffs: indicate the position where the output variables are allocated.

On the other hand, the significant information of variables is:

- BitSize: indicates the size of the variable.
- BitOffs: determine the position where the variable starts.
- Input/output: defines the variable as an input or an output.

The Datalogger tool is used in a scenario where, at least, one EtherCAT slave encapsulates the application data through a Safety Communication Layer (SCL) (refer to subsection 5.1.2.1 for further information). Hence, there are slaves with variables of 9 bytes which are managed as PDUs by the slave lower layers. Therefore, in the ENI file, appears a variable of 9 bytes which is recognized as an SCL variable by its nomenclature (“SCL_...”). May it be the case that not all the information from the PDU is of relevance; hence, the variable nomenclature indicates the Datalogger how many variables

have to generate to store the significant information from the PDU. Specifically, for input variables names beginning with “SCL_D…” the Datalogger generates six variables, while for names beginning with “SCL_WD…” five variables are generated. Similarly, for output variables, three and two variables are generated for names beginning with “SCL_D…” and “SCL_WD…”, respectively. In order to log all these variables, the Datalogger has to be aware of their offset and size. As the structure of the PDU is constant and well known, the offset and size of each variable is easily determined.

5.1.2.2.4.2 Variable extraction

The process of extracting the variables consists of different stages. In first place, the Datalogger starts receiving frames from winpcap or wireshark file, depending on the configuration. Previous to do any processing, the received frames are filtered so just EtherCAT response frames are taken into account. Afterwards, for each datagram within the frame, the command identifier and the logical address are checked against the values obtained from the configuration file. If these steps are successfully passed, the frame is then stored in a circular array. The size of this circular array is big enough to allow datagram processing and variable extraction before frames are overwritten (see Figure 27). Once there is a new entry on the circular array, the variable extraction task is enabled. After a certain period of time, it is checked whether all the expected datagrams have been received; this process is done just once in order to verify that the used ENI file corresponds with the received frames. Additionally, in order to discard redundant frames, it is verified whether the frame has been already processed. Then, in case that two network interfaces are available, it is checked if it is necessary to look for the redundant frame. By taking a look to the data within the datagrams, the Datalogger is able to determine whether all the slaves have given a response or not. Therefore, in case that the redundant frame is required, it is searched in the circular array. Once it is found, the datagrams are combined.

Eventually, the variables are extracted from the datagram. At this point, as it has been previously explained, it is well known in which datagram is allocated each variable. Thus, given that the offset to the position of the variable within the datagram has been previously determined, the value in the offset position is stored in the structure of the corresponding variable. Then, when the Storage Subsytem is commanded to record the data, the variable values are extracted from the variable structure and they are written in the MDF file.

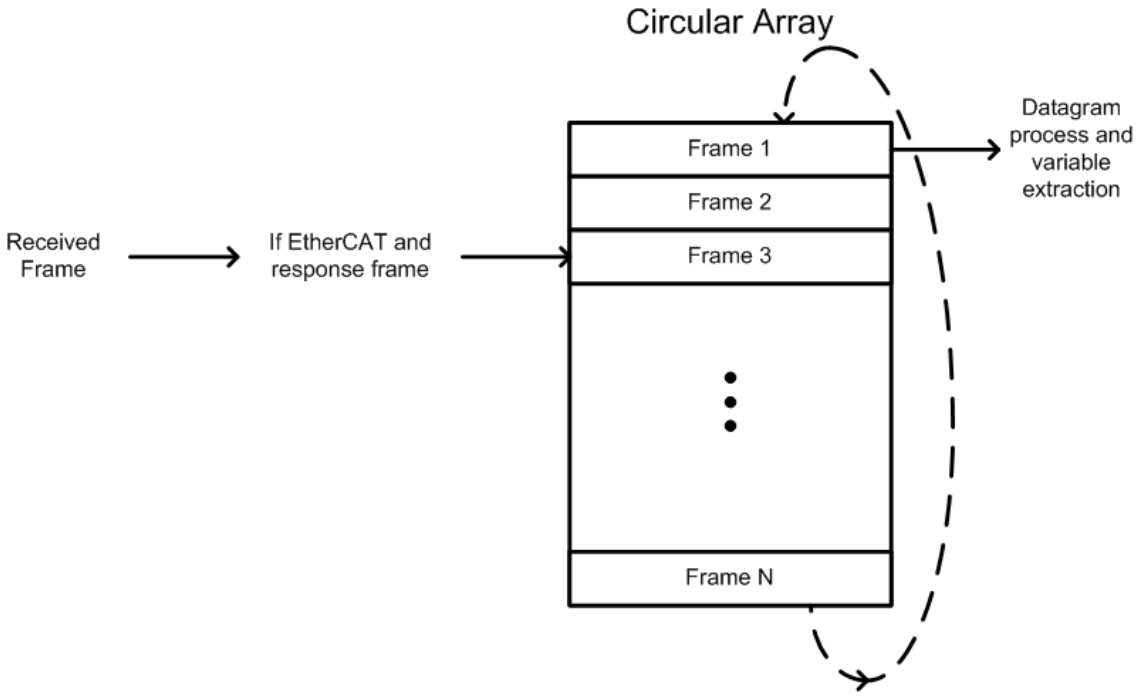


Figure 27: Circular Array operation principle.

5.1.2.3 Validation

As detailed in the two previous sections, the Datalogger takes network frames as inputs and extract the variables generating an MDF file, which is the system output. Therefore, to validate the design, the Datalogger has to be provided with network frames and, afterwards, it has to be checked whether the variables recorded into the MDF file match the expected ones.

The first step is to generate an ENI file. To that purpose, an EtherCAT network is simulated using the TwinCAT tool. In Figure 28, it is shown a screenshot where it can be observed that the network has one slave (an FC1100 evaluation board) and it has, in addition to other variables, one input and one output with SCL nomenclature, which are SCL_Var_WD and SCL_Var_D (see 5.1.2.1.2.1), respectively. Now that the EtherCAT network has been properly configured, the ENI file can be exported.

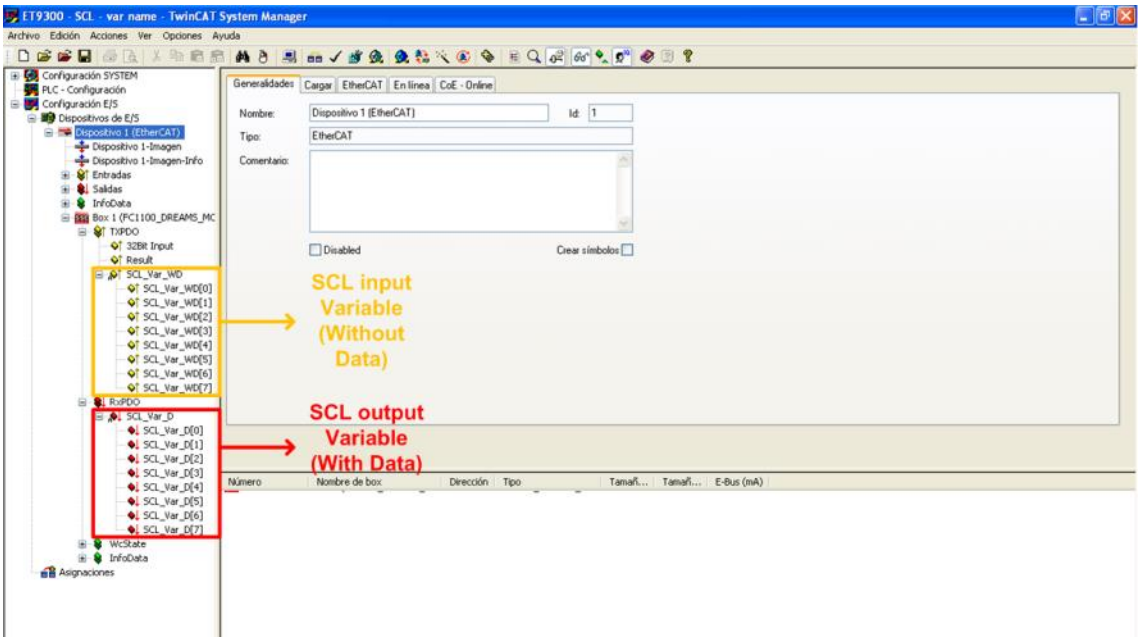


Figure 28: EtherCAT network configured using TwinCAT tool

Next, the file with the network frames has to be created. At this point, Wireshark software tool is used in order to record a communication between the TwinCAT (which acts as a master) and the FC1100 evaluation board. In Figure 29, a piece of the Wireshark capture is shown.

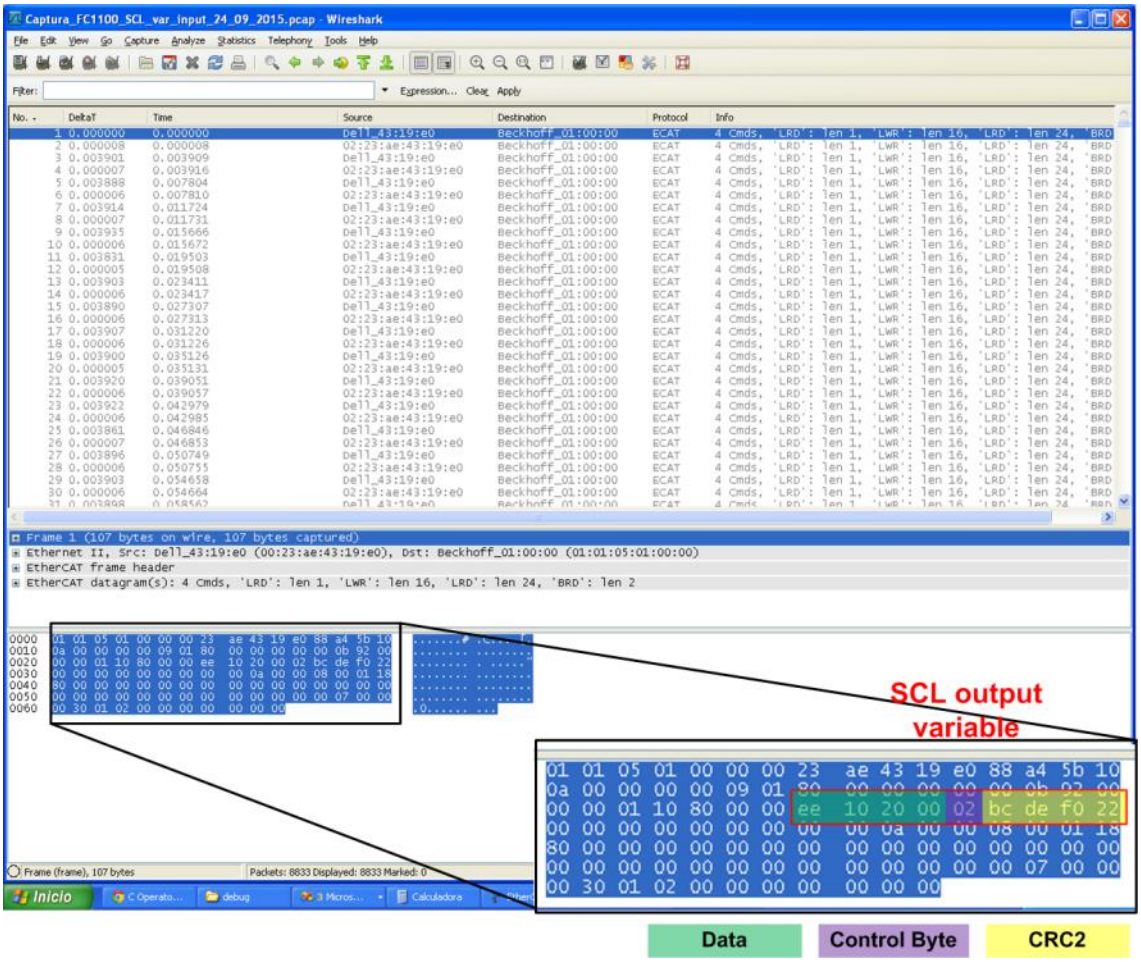


Figure 29: Wireshark capture of the frames exchanged by the EtherCAT master and the FC1100

The Datalogger is fed with this Wireshark file and it is triggered to log, which results in an output MDF file with the variables shown in Figure 28. As can be observed in Figure 30, the Datalogger has created an MDF file with two Channel Groups that contain the two standard input variables (32Bit Input and Result) and, in addition, eight extra variables (five in Channel Group 1 and three in Channel Group 2) have been created. These extra variables correspond to the SCL variables shown in Figure 28. According to subsection 5.1.2.1, a SCL input variable WD (without data) has to be split into five variables which are: FV_activated (1 bit), WD_timeout (1 bit), CE_CRC (1 bit), Device_Fault (1 bit) and CRC2 (4 bytes). Regarding to a SCL output variable D (with data), three extra variables have to be created, which are: Data (4 bytes), activate_FV (1 bit) and CRC2 (4 bytes). Furthermore, as can be observed in Figure 30, the Datalogger has successfully created the MDF file.

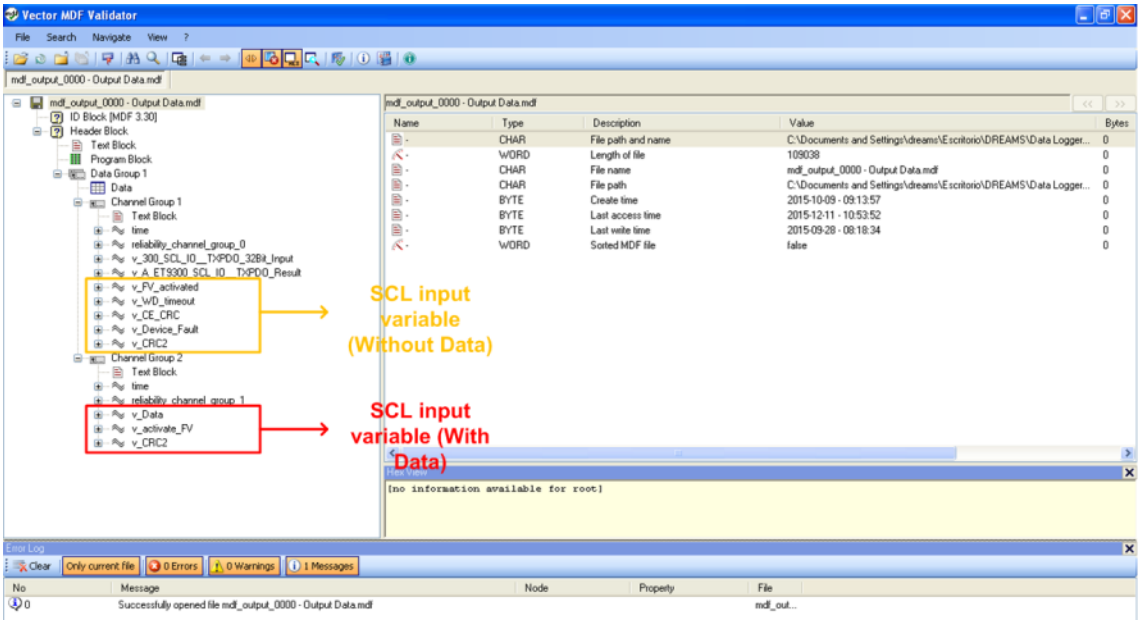


Figure 30: View of the MDF file generated by the Datalogger

In order to check that the variables have been successfully logged, Figure 31 and Figure 32 show the value of two logged variables: Data and CRC2 of the SCL output variable (CANalyzer software form Vector has been used as data plotter). It can be observed that the logged value (in decimal notation) for Data and CRC2 is 2101486 and 5.8621101e8, respectively. This values, in hexadecimal notation, equal to 2010EE (EE0102 changing the endianness) and 22F0DEC2 (2CDEF022 changing the endianness) what matches the values shown in Figure 29, except for the first byte of the CRC2, which has been rounded by the plotter and shows a 2C instead of a BC.

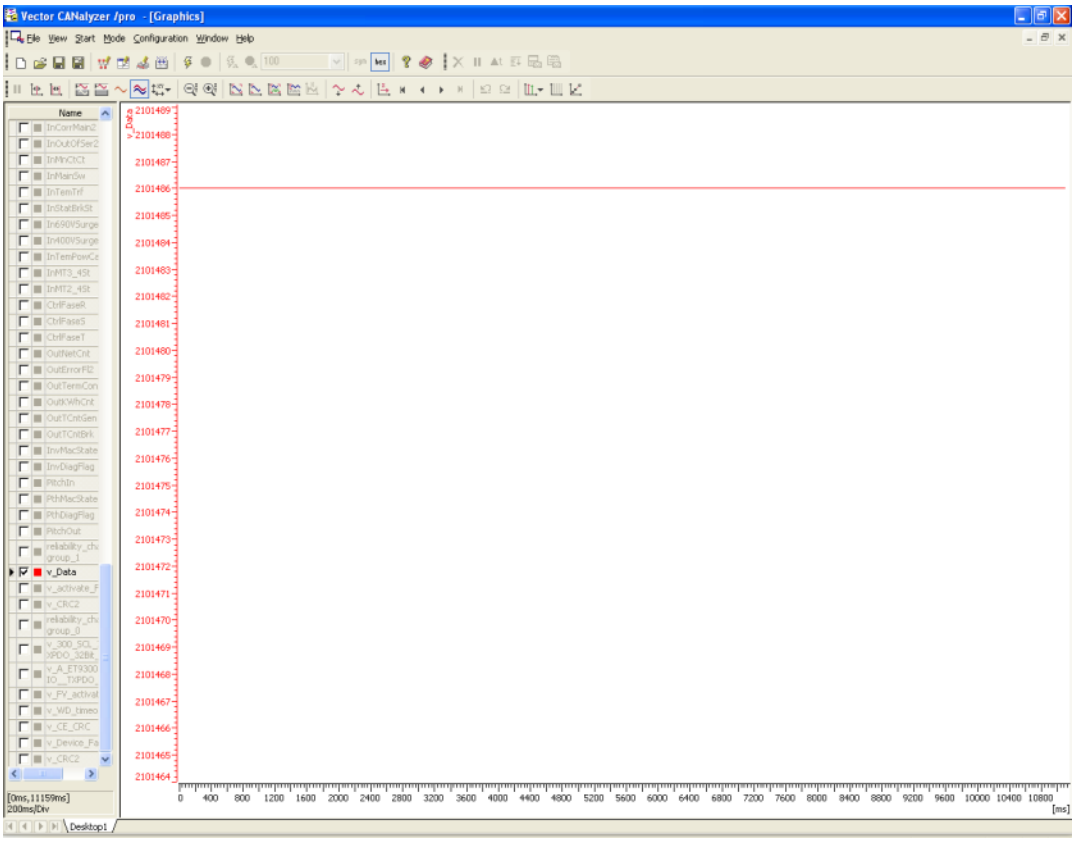


Figure 31: Logged value of Data variable (SCL output)

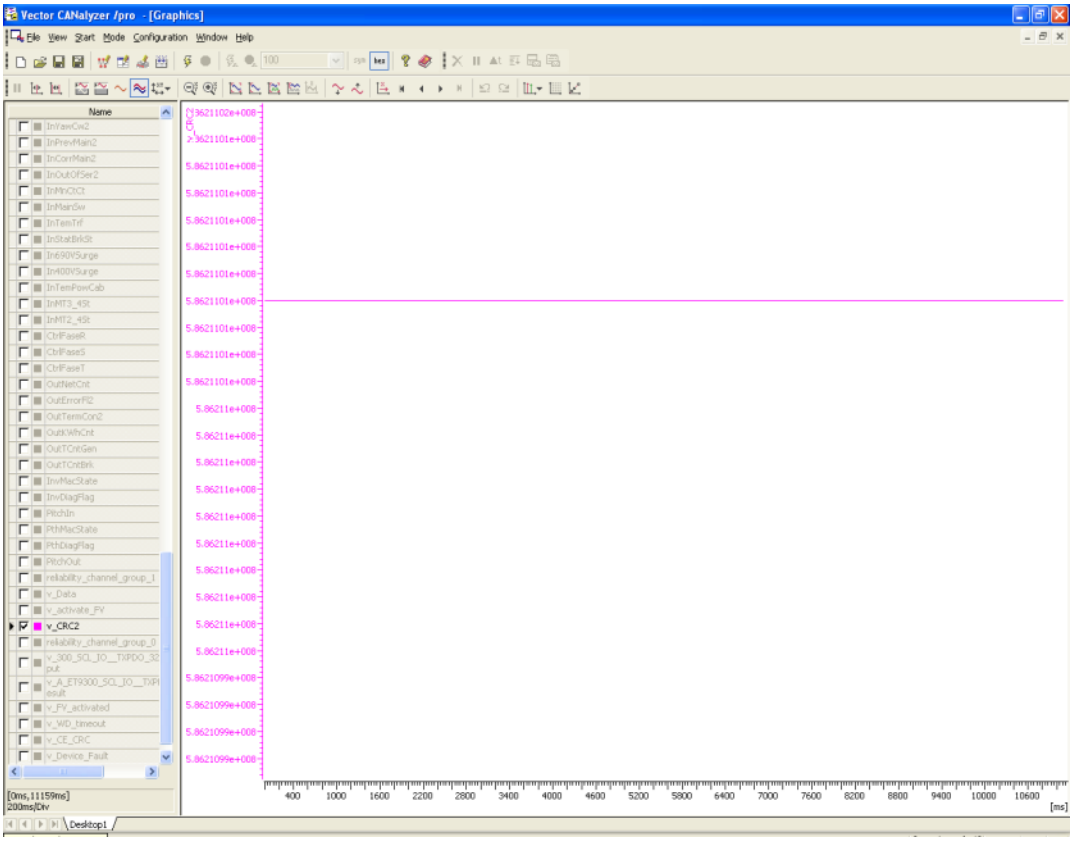


Figure 32: Logged value of CRC2 variable (SCL output)

ANNEX A: SCL PDU'S STATUS/CONTROL BYTE

Status Byte:

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Res	cons_nr_R	Toggle_d	FV_activated	WD_timeout	CE_CRC	Device_Fault	iPar_OK

Figure 33. Status Byte

- **Bit 7:** reserved.
- **Bit 6 (cons_nr_R):** set when F-Device has reset its consecutive number.
- **Bit 5 (Toggle_d):** Toggle bit indicating a trigger to increment the virtual consecutive number within the F-Host.
- **Bit 4 (FV_activated):** set during start-up and in cases of any communication error. Triggers safety state.
- **Bit 3 (WD_timeout):** set when the F-Device recognizes a communication error due to watch dog time.
- **Bit 2 (CE_CRC):** set when the F-Device recognizes a communication failure due to a CRC2 error.
- **Bit 1 (Device_Fault):** set by the specific device technology firmware if there is a malfunctioning in the F-Device.
- **Bit 0 (iPar_OK):** set when the F-Device has new parameters values assigned.

Control Byte:

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Res	Res	Toggle_h	activate_FV	res	R_cons_nr	OA_Req	iPar_EN

Figure 34: Control Byte

- **Bit 7:** reserved.
- **Bit 6:** reserved.
- **Bit 5 (Toggle_h):** Toggle bit indicating a trigger to increment the virtual consecutive number within the F-Device.
- **Bit 4 (activate_FV):** can be set to force the outputs of an F-Device to configured or built-in fail-safe values.
- **Bit 3:** reserved.
- **Bit 2 (R_cons_nr):** set when the F-Host detects a communication error, either by the Status Byte or by itself. It commands the F-Device to reset the consecutive number.
- **Bit 1 (OA_Req):** not safety related. Should be used by the F-Device to indicate locally the request for an operator acknowledgement.
- **Bit 0 (iPar_EN):** set by the F application within an F-Host in case of a parameterization request.

For further information regarding the Status/Contol byte, refer to the IEC 61784-3-3 document [1].

6 Bibliography

[1] IEC, "IEC 61784-3-3," in Industrial Communication Networks, ed, 2007, p. 122.

[2] L. Rubio, "Functional Specification," vol. 1, ed, 2014, p. 15.

[3] L. Rubio, "Safety Requirement Specification," vol. 1, ed, 2014, p. 22.

[4] L. Rubio, "Safety Concept," ed, 2014, p. 33.

[5] M. C. Zubia, "DREAMS SCL," in SCL's Design vol. 1, ed: Ikerlan, 2015, p. 37.

[6] DREAMS Consortium. Distributed Real-time Architecture for Mixed Criticality Systems, 2014. Deliverable D1.2.1

[7] DREAMS Consortium. High-level Design of Mixed-Criticality Cluster Communication Services, 2014. Deliverable D3.1.1

[8] DREAMS Consortium. First Implementation of Mixed-Criticality Cluster Communication Services, 2015. Deliverable D3.1.2

[9] DREAMS Consortium. Final Implementation of Mixed-Criticality Cluster Communication Services, 2016. Deliverable D3.1.3

[10] DREAMS Consortium. High-level Design of Global Resource Management Services, 2014. Deliverable D3.2.1

[11] DREAMS Consortium. First Implementation of Global Resource Management Services, 2015. Deliverable D3.2.2

[12] DREAMS Consortium. Final Implementation of Global Resource Management Services, 2016. Deliverable D3.2.3

[13] DREAMS Consortium. High-Level Design of Cluster-level Safety and Security Services, 2014. Deliverable D3.3.1

[14] DREAMS Consortium. First Implementation of Cluster-level Safety and Security Services, 2015. Deliverable D3.3.2

[15] DREAMS Consortium. Final Implementation of Cluster-Level Safety and Security Services, 2016. Deliverable D3.3.3

[16] DREAMS Consortium. System Demonstrator running mixed-criticality healthcare and entertainment use case, 2017. Deliverable 8.2.1

[17] SAE International. SAE AS 6802: Time-Triggered Ethernet, 2011. http://standards.sae.org/as6802/