ROMULUSLIB: AN AUTONOMOUS, TCP/IP-BASED, MULTI-ARCHITECTURE C NETWORKING LIBRARY FOR DAQ AND CONTROL APPLICATIONS

A. Yadav, H. Boukabache*, N. Gerber†, K. Ceesay-Seitz, D. Perrin
European Organization for Nuclear Research (CERN), Geneva, Switzerland.

Abstract

The new generation of Radiation Monitoring electronics developed at CERN, called the CERN RadiatiOn Monitoring Electronics (CROME), is a Zynq-7000 SoC-based Data Acquisition and Control system that replaces the previous generation to offer a higher safety standard, flexible integration and parallel communication with devices installed throughout the CERN complex. A TCP/IP protocol based C networking library, ROMULUSlib, was developed that forms the interface between CROME and the SCADA supervision software through the ROMULUS protocol. ROMULUSlib encapsulates Real-Time and Historical data, parameters and acknowledgement data in TCP/IP frames that offers high reliability and flexibility, full-duplex communication with the CROME devices and supports multi-architecture development by utilization of the POSIX standard. ROMULUSlib is autonomous as it works as a standalone library that can support integration with supervision applications by addition or modification of parameters of the data frame. This paper discusses the ROMULUS protocol, the ROMULUS Data frame and the complete set of commands and parameters implemented in the ROMULUSlib for CROME supervision.

INTRODUCTION

The Occupational Health & Safety and Environmental Protection (HSE) at CERN obliges to the protection of CERN personnel and the public from any unjustified exposure to ionising radiation. The radiation protection group (RP) at HSE has the mandate to monitor the radiological impact of CERN's accelerators and installations by active monitoring and logging of radiation levels at different experimental sites spanning the CERN complex. To facilitate this, the new generation of CERN RadiatiOn Monitoring Electronics, called CROME [1], was developed by the CROME team of the Instrumentation and Logistics (IL) section within the RP group and is responsible for the design, development, installation and maintenance of these specialised radiation monitoring systems. Starting from Long Shutdown 2 (LS2) of the Large Hadron Collider (LHC) in 2019, the older generation of radiation monitors, namely The Area Controller (ARCON) is being replaced by the new CROME devices and will be operational for the Run 3 of the LHC in February 2022. The consolidation of the current generation of RAdiation Monitoring System for the Environment and Safety (RAMSES) monitors by CROME is planned to be completed by the Long Shutdown 3 (LS3) in late 2027.

* Corresponding author Dr. H.B. (hamza.boukabache@cern.ch)

† N.G. is a former CERN Fellow. He is currently at CSEM, Neuchâtel (CH)

The CROME devices consist of the autonomous Monitoring Units, Alarm Units, a Junction Box and an Uninterruptible Power Supply. [2–4] The autonomous monitoring units, called the CROME Measuring and Processing Units (CMPUs), consist of an ionization chamber and an electronic readout system. The CMPU can be either a wall-mounted system where the CMPU is directly attached to the ionization chamber or it can be a rack-mounted system where the CMPU is connected to the ionization chamber with a specialized cable. The rack-mounted system is used for monitoring areas with high radiation levels that can damage the readout electronics. In this case, a custom plastic ionization chamber with graphite coating is placed directly into that area with high radiation levels, whereas the readout electronics are placed in an area of lower radiation for their protection. The ionization chamber detects ionizing radiation and converts it to a readable current value. A read-out chip measures this current, which can be within 2 fA to 1 µA. Because these currents can be so low, a specialized cable SPA6 was developed by the RP team at CERN for the rack-mounted system. The SPA6 cable is used for Signal and High Voltage lines up to one kilometer distance. The CMPU's front-end readout interface transmits the current value to the FPGA programmable logic (PL) of a Zync-7000-based System-on-Chip (SoC), which uses it to calculate the real-time radiation dose rate as well as the total radiation dose received in the monitored area at the ionization chamber location. All safety-critical decisions and actions such as measurement, dose rate calculation, temperature compensation, alarm generation and interlock generation are performed by the PL. This is done to ensure system reliability by implementing operations within a Finite-State Machine. A complex programmable logic device (CPLD)-based watchdog works in tandem with the SoC. It monitors the PL state machine to ensure correct dose rate calculations and overall functionality. It is allowed to reset/reboot the SoC when it is in an undefined state. At the detection of dangerous conditions, e.g. if the radiation dose or dose rate exceeds a defined limit, the CMPU automatically generates local and remote alarms and a beam interlock signal that stops the concerned accelerator or machine. Parameters like dose and dose rate limits, current-to-radiation conversion factors, and many more can be configured remotely by the authorized members of the radiation protection group. A schematic of the CROME devices and it's part is shown in Fig. 1.

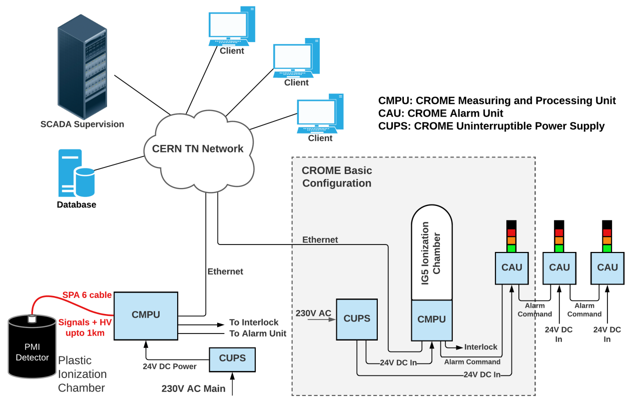


Figure 1: CROME Radiation Monitor components and basic configuration.

The radiation monitoring CROME devices at CERN are required to fulfil safety requirement in accordance with the IEC 60532 standard for radiation protection instrumentation. This is in compliance to the task of interlock triggering of machines in case of high radiation levels. Therefore, CROME devices have been developed to satisfy Safety Integrity Level 2 (SIL 2), as defined in the IEC 61508 standard in accordance with the IEC 60532 for functional safety of Electrical/Electronic/Programmable Electronic Safety-related Systems. We have therefore followed a process and implemented measures to reduce the risk of dangerous failures during the development and operation while following stringent methodology to avoid both random and systematic faults for both firmware [5] and hardware [6].

The CROME hardware deployed throughout the CERN complex communicates with a SCADA supervision system, called Radiation and Environment Monitoring Unified Supervision (REMUS) [7, 8]. In order to facilitate reliable connectivity and logging capabilities, the development of a dedicated TCP/IP-based C networking library, ROMULUSlib was initiated by the CROME team. ROMULUSlib is the communication interface between CROME and REMUS. It sits in the userspace of a POSIX compliant OS and allows multi-user full-duplex communication with CROME devices through REMUS and/or an 'expert application' connected to the devices through the network via Ethernet connection. ROMULUSlib compiles on multiple architectures and at present, ROMULUSlib has been built and tested successfully with gcc 4.8.5 on x84_64, arm32 and with Apple clang version 11.0.3 on x86_64 Darwin MacOS Kernel 19.4.0. In this paper we present the details of ROMULUSlib, the construction of the data packets for command and data communication over the TCP/IP network. We present the overall structure of the ROMULUSlib code base and a set of utilities that are a useful aid for anyone instantiating ROMULUSlib for their DAQ and Control use case. Beside this, an independent project, RomLibEmu [9] was initiated to develop a regression testing framework for debugging systematic faults during development and before the release of every new version of the ROMULUSlib.

This paper is organized as follows: in background and related work, we discuss some of the previous library implementations developed for TCP/IP supervision and how ROMULUSlib compares to the same. In the next section, we give a brief overview of the CROME radiation monitoring system architecture and the REMUS Supervision system, followed by the details of the implementation of ROMULUSlib that forms the interface for the SoC to SCADA communication. In this section we present the details of the TCP/IP frame, supported commands, parameters and ranges with their corresponding acknowledgement response and test utilities. ROMULUSlib supports both real-time stream and buffered data stream such as status request, historical record and events that are discussed here. In the end, we present the current application use cases and tests deployed at CERN and at the European Spallation Source (ESS) followed by the future work and adaption of ROMULUSlib library for varied use cases.

BACKGROUND AND RELATED WORKS

Early control operations at CERN radiation protection employed PLC to SCADA based communication for control application. CERN also uses this for LHC Cryogenics control in accelerator and experimental control, Gas systems, Cooling systems, HVAC etc. For control and monitoring, Ethernet based communication protocols such as Profinet and Ethernet/IP are employed [10]. The UNICOS framework is one example of the same [11]. Another example of the communication protocol used for real-time automation is the MODBUS TCP/IP which uses the MODBUS protocol for communication encapsulated within the TCP/IP wrapper for communication over the ethernet.

CERN currently started very recently using Zynq-based SoCs for a wide range of applications in detector control and front-end data acquisition electronics designs. These SoCs offer greater flexibility to the user for design on programmable logic, offers an embedded linux platform on the processing system for high-level software development and provides faster performance due to the PS-PL interface. The open-source WinCC OA is the currently widely adopted SCADA framework for detector control applications which is also used by REMUS supervision [8]. One of the adopted communication frameworks in ATLAS Detector for DAQ to DCS communication is the Quasar framework [12] which allows building of OPC UA servers for client-server communication. It is employed at ATLAS's Tile Calorimeter measurement[13] and Global Feature Extrator (gFEX) Hardware Trigger[14] among others. Quasar's client-server communication model integrates with the WinCC OA SCADA and allows read/write of predefined variables, and creation of multiple OPC servers. A new library, MilkyWay, based on FreeOpcUa is further being developed in Python [15].

While PLC-to-SCADA based communication frameworks exist; with the introduction of SoC-to-SCADA systems, fast and reliable safety critical communications framework needed to be developed. Therefore, the development of communications protocol for safety-critical radiation monitoring and control was initiated in 2015 by the CROME team at CERN. The communications specifications provided by the ROMULUSlib allows user to have communication reliability of TCP/IP while having greater control over creation of variable length data frames for DAQ and control applications. ROMULUSlib runs on each communicating device and communicates with the WinCC OA based SCADA through a driver. The users have the flexibility to update their own parameters of interest which compiles seamlessly with ROMULUSlib. This further simplifies integration with embedded linux userspace application development which makes ROMULUSlib suitable for adoption for SoC-to-SCADA TCP/IP communication systems.

CROME: RADIATION MONITORING ELECTRONICS AT CERN

The CROME device is developed to provide a versatile interface for:

1. Continuous real-time monitoring of ambient dose equivalent rates over up to nine orders of magnitude.
2. Alarm and interlock functionality with a probability of failure down to $10e^{-7}$.
3. Long term permanent and reliable data logging by linking to a SCADA supervision or an expert application running on a portable PC via Ethernet.
4. Edge computing with powerful processing capabilities for embedded calculation.

The basic configuration of the CROME device consists of the three main parts: CROME Measuring and Processing Unit (CMPU), the CROME Alarm Unit (CAU) and the CROME Uninterruptible Power Supply (CUPS). These are connected to the global supervision structure as shown in Fig. 1. The CMPU hosts the System-on-Module board which implements a Zynq-7020. This SoC is composed of the 32-bit ARM Processing System (PS) which runs CROMiX-18 a custom embedded Linux distribution developed using the YOCTO project and the Programmable Logic (PL) FPGA fabric that is programmed with a parameterizable hardware implementation for readout, calculations and safety-critical controls.

The front-end electronics is also part of the CMPU. It performs analogue-to-digital conversion of the current signal generated by the radiation detector to which it is attached providing a continuous real-time measurement of ambient dose equivalent rates. The CMPU in turn generates radiation alarm and interlock signals and enables long-term and data logging by integration with REMUS. The communication interface for the CROME devices and the REMUS supervision is the ROMULUSlib and it provides four main functionalities:

1. Networking: All communications happening over TCP/IP frame is handled by the functions defined in ROMULUSlib.
2. TCP/IP frame construction: ROMULUSlib provides structs to encapsulate different data variables with support for nearly all C data types. This can be easily updated by updating the struct table. The construction of ROMULUS frame is handled by the ROMULUSlib, including functions to access and modify specific parts of the frame which provides much more flexibility and control to the user.
3. Multiple Communication Modes: ROMULUSlib provides full-duplex communication over TCP/IP via multiple communication modes to transmit single, multiple and infinite frames which can be used for streaming data in multiple ways as per application requirement.
4. Utilities: Multiple utilities are provided within the ROMULUSlib for utilities such as log reporting of application flow, warnings and errors, functions to print frame, struct and internal data struct, checksum check functions and check functions for memory allocation and free memory.

CERN's REMUS supervision [7, 8], is a WinCC OA based SCADA supervision system which is used for communication and parameterization of CROME devices. REMUS integrates the SCADA system with the open-source streaming platform Apache Kafka, a widely adopted technology, that allows data-streaming in near real-time data to the Data Visualization Tools and Web Interfaces. REMUS provides a secure interface for full-duplex communication interface with external Control devices. A functional architecture of the REMUS supervisory control along with the application of ROMULUSlib for data streaming and control from WinCC OA is shown in Fig. 2.

ROMULUSLIB

The CROME devices communicate with REMUS using the full-fledged TCP/IP protocol which is implemented in ROMULUSlib, a stand-alone TCP/IP networking library developed in C that supports cross-compilation and portability by making use of the POSIX-standard, more specifically sockets. TCP/IP communication has been chosen to ensure high reliability and automatic data reordering. Therefore, all data communicated as ROMULUS data packets are encapsulated in TCP/IP frames. The device responds to most of the messages sent from the supervision. This is reconfigurable by defining the structure of the ROMULUS frames and defining the response packets as per requirements specifications. For each received message the device returns an acknowledge message. The acknowledgement messages also ensure correct functionality of devices by having unique request-acknowledge pair. Specific error messages are communicated to be returned to ensure that there is a consistent answer from the device (e.g. malformed or unknown message). All packages received by the devices also contain a Sequence ID of the sender. The response or acknowledge message will include this Sequence ID. This ensures that the device can have multiple concurrent users. Besides SCADA

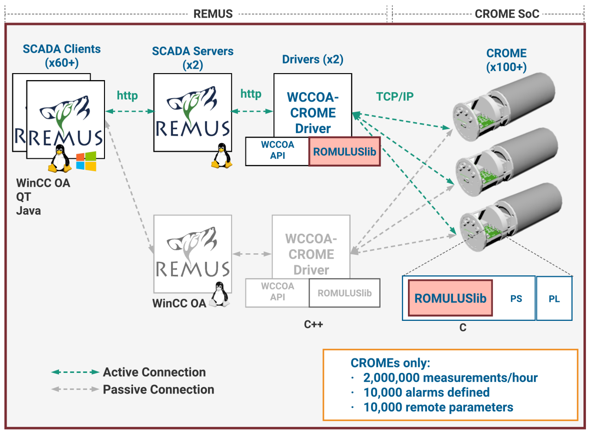


Figure 2: REMUS Supervisory Control and Data Acquisition Architecture [7, 8], Courtesy of Adrien Ledeul

supervision, the device is also fully configurable and parameterizable by a dedicated standalone ‘expert application’ that runs on a portable computer and is connected either directly to the device or through the network via the Ethernet socket. For this application, the CROME Team has developed a standalone application in NI LabVIEW that can connect to the CROME devices through the network.

In this section, we present the protocol used for the communication between the CROME devices and the REMUS supervision. The protocol is also used to communicate with the NI LabVIEW-based expert application.

ROMULUS Data Frame

The ROMULUS data packet, as shown in Fig. 3, is encapsulated into a TCP/IP frame whose payload is limited to ETHERET_MTU (1500 Bytes) while having no fixed length for data packet itself in order to obtain more flexibility. Within the ROMULUS Data packet, the header is formed by: data length of the DATA, ID of the source of the message, ID of the message, the command code and the number of parameters. This is followed by the Data corresponding to the command code and the footer, which is the checksum.

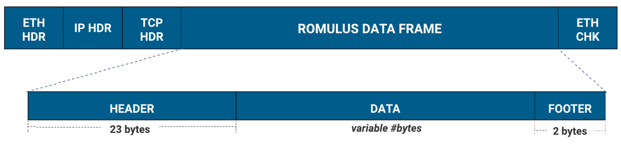


Figure 3: ROMULUS TCP/IP Data Packet

The maximum data length of the DATA in bytes can be calculated as follows:
*max*DLC = ETHERNET_MTU - [*sizeof*(IP header) + *sizeof*(TCP header) + *sizeof*(romulus_frame_header) + *sizeof*(romulus_frame_footer)]

Both, command and response packets have the same structure. A generic DATA segment of the ROMULUS Data Frame for Get/Set commands is shown in Fig. 4. Note that, multiple parameters can be communicated within one data

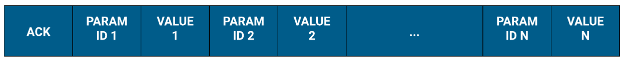


Figure 4: Data segment of any generic get/set command in ROMULUSlib

frame. Therefore, it is possible to initiate multiple operations through a single message. Thus, the variable length of the ROMULUS data frame is due to the field DATA whose length is variable. In the response frame, if no error has occurred, the content of the DATA field of response will begin with an Acknowledgement (ACK) code. Otherwise, if an error occurs, the device returns an error code corresponding to one of the related errors, such as Acknowledgment, Illegal command, Illegal value etc.

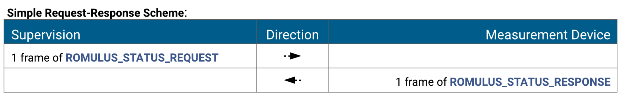
We have provided command definition and data frame construction support for Simple and Complex Request-Response Schemes of communication within ROMULUSlib, which are discussed in the following section.

ROMULUSlib Commands

ROMULUSlib commands define how the device and supervision can communicate using the ROMULUS protocol. ROMULUSlib currently supports 18 commands to interact with the CERN’s REMUS/ROMULUS protocol. These commands are categorized into two different schemes of communication within ROMULUSlib. These are namely Simple Request-Response Scheme and Complex Request-Response Scheme.

ROMULUS Simple Request-Response Scheme: This is the most common mode for communication using the ROMULUS protocol. The supervision sends a request to the measurement device, which is then responded to with a response frame as shown in Fig. 5. Example of this Simple Request-Response scheme is Get and Set Parameter Request and Response, Get and Set Status Request and Response, and Get and Set Timestamp Request and Response.

ROMULUS Complex Request-Response Scheme: Within the complex Request-Response scheme the communication


Simple Request-Response Scheme:

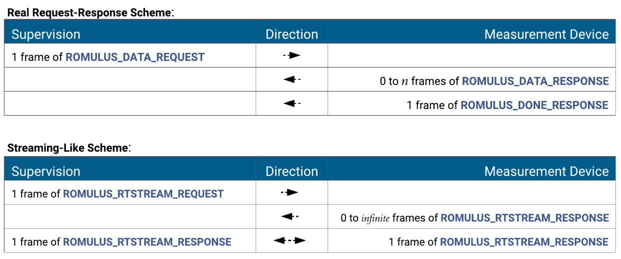
Supervision	Direction	Measurement Device
1 frame of ROMULUS_STATUS_REQUEST	→	
	←	1 frame of ROMULUS_STATUS_RESPONSE

Figure 5: ROMULUS Simple Request-Response Scheme

could be a Real request-response scheme or a Streaming request-response scheme, as shown in Fig. 6.

The Real request-response scheme communication mode is similar to the simple request-response scheme, with the only difference in being that the measurement device can reply with multiple response frames while the last possible response frame is always fixed to indicate the end of the communication. Example of Real Request-Response scheme is Historical Data Request and Response frames, which initiates communication of multiple frames with the last frame being the Historical Data Done command frame.

In the Streaming request-response scheme, the supervision sends one request to initiate streaming and then the measurement device sends a possibly infinite number of responses until either side breaks the connection or sends a stop command to terminate the stream. Example of Streaming Request-Response scheme is Real-Time Stream request and response which can be terminated by Real-Time Stream Stop command.


Real Request-Response Scheme:

Supervision	Direction	Measurement Device
1 frame of ROMULUS_DATA_REQUEST	→	
	←	0 to n frames of ROMULUS_DATA_RESPONSE
	←	1 frame of ROMULUS_DONE_RESPONSE

Streaming-Like Scheme:

Supervision	Direction	Measurement Device
1 frame of ROMULUS_RTSTREAM_REQUEST	→	
	←	0 to *infinite* frames of ROMULUS_RTSTREAM_RESPONSE
1 frame of ROMULUS_RTSTREAM_RESPONSE	↔	1 frame of ROMULUS_RTSTREAM_RESPONSE

Figure 6: ROMULUS Complex Request-Response Scheme

Associated with each command are the certain set of user defined parameters and value pair that the user would want to create for their own applications. This is implemented as C Struct data type. Within the structs, the user can define all the different parameters for which a unique ID is automatically generated by ROMULUSlib at compile time which is used in ROMULUS frame construction. Within the C struct, the library currently supports all essential data types: bool, unsigned char/uint8, int32, int64 and float. To categorically define the communicated parameters, eight structs each with a different use case, are defined. A summary of the C struct is provided in Table 1.

ROMULUSlib Architecture and Functions

The data to be communicated is prepared by the CROME devices and handled by ROMULUSlib for communication. ROMULUSlib provides functions for constructing and communicating TCP/IP frames with the supervision system. Frame construction is carried out by functions defined within romulus_frame.h; including function to define the life cycle of a frame, access functions to access and modify specific parts of the frame, validator function for checking the validity of the frame and utility functions to print and debug the frame. Once a valid frame is constructed, functions defined within romulus_net.h are responsible for TCP/IP communication with the supervision system. This includes functions to control and send and receive romulus frames. ROMULUSlib also provides functions to report version and along with run logs, warning and error messages, and also generates timestamps in both, UNIX epoch and human readable form. A brief summary of all important functions of ROMULUSlib with it's placement in the corresponding header is given is Table 2 and a dependency graph of all the ROMULUSlib headers is shown in Fig. 7.

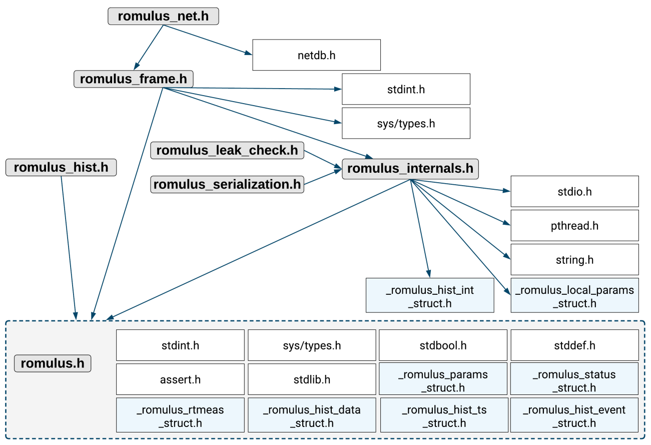

romulus_net.h
netdb.h
romulus_frame.h
stdint.h
sys/types.h
romulus_leak_check.h
romulus_hist.h
romulus_serialization.h
romulus_internals.h
stdio.h
pthread.h
string.h
_romulus_hist_int _struct.h
_romulus_local_params _struct.h
romulus.h
stdint.h
sys/types.h
stdbool.h
stddef.h
assert.h
stdlib.h
_romulus_params _struct.h
_romulus_status _struct.h
_romulus_rtmeas _struct.h
_romulus_hist_data _struct.h
_romulus_hist_ts _struct.h
_romulus_hist_event _struct.h

Figure 7: Dependency Graph of all ROMULUSlib Headers

ROMULUSlib Utilities

ROMULUSlib works as a standalone library for Linux User Space application. In order to facilitate debugging, ROMULUSlib also provides additional utilities. These utilities are meant to aid the developer in writing complete embed-

Table 1: Summary of Supported Data from CROME Organized in C Struct

Struct	**Description**
romulus_hist_data_struct	Struct for historic data variables.
romulus_hist_event_struct	Struct for hist event variables.
romulus_hist_int_struct	Struct for historic data that is not used by the supervision.
romulus_hist_ts_struct	Struct to define time ranges.
romulus_local_params_struct	Struct for Local parameters to parameterize the system locally.
romulus_params_struct	Struct for Romulus parameters, used to parameterize the system and read back inherent parameters from the system.
romulus_rtmeas_struct	Struct which contains real time measurements.
romulus_status_struct	Struct which contains variables to indicate system status .

Table 2: Summary of ROMULUSlib Functionalities

File	Description
romulus.c	Contains: - all the different structs which are used by the ROMULUS library, - all debug functions to query and manipulate the amount of log, - time function definitions to generate timestamp in machine and human readable form, - all function definitions to read the version of the library, - all function and data type definitions for ROMULUSlib's internal types, - definitions to create parameter protection masks to specify the write-protected parameters.
romulus_frame.c	Contains: - all structure definitions including data types and constants related to romulus frames, - all functions required to define the life-cycle of a TCP/IP frame such as header allocation, initialization, allocation, sequence ID generation and free frame memory, - all functions required to access parts of the TCP/IP frame such as, read header, footer, response code and data with offset and append header, footer and data to the frame, - a specific function to create request and response frames, including functions to check validity of the frames, - a specific function to update the parameters of the struct with contents of the TCP/IP frame, - a specific function which are used to manipulate/use status frames, historical data and events frame and real-time measurement frame, - utility functions such as print frame for debugging.
romulus_serialisation.c	Functions to read struct data from files.
romulus_hist.c	All definitions concerning historic data and events.
romulus_internals.c	- Struct, variable & function definition for romulus local parameters, internal historical data. - Definition of printing and IO functions. - Definition of internal data structures. - Struct, variable and function definition for romulus checksum. - Definition of some utility macros.
romulus_leak_check.c	Functions for leack checking, The leakcheck functions print information about memory allocations and calls to free().
romulus_net.c	- All type definitions for romulus frame based network connections. - All functions required to manage the life-cycle of connections to transmit romulus frames - All functions required to send and receive romulus frames. - All functions required to control the sending and reception of romulus frames.

ded applications. Some of these utility applications are as follows:

1. remote_stream: Remote Stream application queries the Embedded Application running on PS and prints the TCP/IP frames for Real-Time Data streaming on port n+1.

2. struct_info_printer: Prints the all Struct Information such as member name, ID and data type as defined by the user.

3. remote_dump: Remote Dump application queries the Embedded Application running on PS and prints status, parameters and hist data by fetching the TCP/IP frames streaming on port n.

RESULTS

At present, 150 CROME devices have been deployed at CERN as the new generation of radiation monitors, all communicating simultaneously with the REMUS supervision via ROMULUSlib ver6.2. The next major release for ROMULUSlib ver7.0 is tested and is scheduled for deployment in December 2021. All tests of ROMULUSlib are performed by a standalone regression testing framework, RomLibEmu[9]. RomLibEmu is developed in Python and works independently of ROMULUSlib to test CROME device's reaction to misconfigurations. This provides a stress testing framework for the application's network interface and test the overall robustness of the software towards the injected faults errors.

EPICS Integration

CROME radiation monitors equipped with ROMULUSlib are currently also in use at the European Spallation Source (ESS) in Lund, Sweden. The ESS' Radiological and Environmental Monitoring System (REMS)[16] uses a two fold approach: Influx DB and Grafana approach for quick online integration, and EPICS integration for real time DAQ and control.

The ESS REMS development team has integrated EPICS [17], the software infrastructure for development of distributed control systems widely used in particle accelerators, large physics experiments and telescopes. The ESS made use of the asynDriver module, a general purpose driver that integrates the device code to the hardware-level code. To integrate EPICS with CROME, initialization files to establish communication link to CROME devices have to be defined using ROMULUSlib. Suitable database files to describe the record types and names of the process variables have to be defined. The CROME devices communicate with the EPICS Input-Output-Controllers that are deployed on a dedicated virtual machine as services and launched at start-up. The collected measurements are posted to an influx database which is then made available via the Grafana interface on the ESS network.

CONCLUSION AND FUTURE WORK

In this paper, we have presented ROMULUSlib, a standalone TCP/IP networking library developed in C. We have presented the ROMULUS data packet structure within the TCP/IP frame, and the communication protocol through request-response mechanism which allows supervision system to request single frame, multiple frames or streaming data frames. Besides this, a comprehensive feature list of all ROMULUSlib functionalities is presented along with debug tools. ROMULUSlib communicates seamlessly with SCADA supervision systems for DAQ and control applications while reliably carrying out millions of data packet transactions every hour (e.g. REMUS supervision handles 2,000,000 measurements/hour from CROME devices currently operational at CERN).

ROMULUSlib makes use of C Struct for storage of application variables of corresponding data types. These C Struct representation allows users to define their own variables as per the application requirement and recompile to autonomously generate ROMULUSlib executable in either arm32 or x86_64 architectures with the new set of application variables. ROMULUSlib currently supports bool, unsigned char/uint8, int32, int64 and float data types to define application variables. Support for uint32 and uint64 will be added in the next versions.

The library has also been developed into a Labview VI using native Labview primitives which allows fast testing of new features and debugging applications. ROMULUSlib currently has been tested for reliability using RomLibEmu, an independent software framework developed in Python; and is presently operational with REMUS supervision system at CERN in Geneva, Switzerland and REMS EPICS supervision system at ESS in Lund, Sweden. In the future versions, we intend to add more regression tests for the library and while robust in application, we would be adding further to the instruction set as per application requirement, along with scaling the library to support varied processor architectures.

ACKNOWLEDGEMENTS

The authors would like to thank Markus Widorski (CERN) for REMUS integration tests, Alasdair Day (ESS) and Juha Hast (ESS) for the performance tests at ESS and the integration of the library into EPICS supervisory system.

REFERENCES

[1] *CROME*, `https://crome.web.cern.ch/`.

[2] H. Boukabache *et al.*, "Towards a novel modular architecture for cern radiation monitoring," *Radiation protection dosimetry*, vol. 173, no. 1-3, pp. 240–244, 2017.

[3] C. Toner *et al.*, "Fault resilient fpga design for 28 nm zynq system-on-chip based radiation monitoring system at cern," *Microelectronics Reliability*, vol. 100, p. 113 492, 2019.

[4] H. Boukabache, "Crome remote management of soc-based radiation monitors both at cern and ess," System-on-Chip 2nd Workshop - CERN, 2021. `https://indico.cern.ch/event/996093/`

[5] K. Ceesay-Seitz, H. Boukabache, and D. Perrin, "A functional verification methodology for highly parametrizable, continuously operating safety-critical fpga designs: Applied to the cern radiation monitoring electronics (crome)," in *SAFECOMP 2020 : International Conference on Computer Safety, Reliability, and Security*, 2020.

[6] S. K. Hurst, H. Boukabache, and D. Perrin, "Overview of a complete hardware safety integrity verification according to iec 61508 for the cern next generation of radiation monitoring safety system," in *Proceedings of the 30th European Safety and Reliability Conference and 15th Probabilistic Safety Assessment and Management Conference*, 2020.

[7] A. Ledeul, A. Savulescu, G. S. Millan, and B. Styczen, *Data streaming with apache kafka for cern supervision, control and data acquisition system for radiation and environmental protection*, 2019.

[8] A. R. Ledeul, "Integration of crome into remus," SoC Interest Group Meeting, 2020. `https://indico.cern.ch/event/882283/`

[9] K. Ceesay-Seitz, M. Leveneur, H. Boukabache, and D. Perrin, "Romlibemu: Network interface stress tests for the cern radiation monitoring electronics (crome)," in *18th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'21)*, 2021.

[10] R. Mastyna, J. Casas-Cubillos, E. Blanco Vinuela, N. Trikoupis, and M. Felser, "Profinet communication card for the cern cryogenics crate electronics instrumentation," 2017.

[11] H. Milcent, E. Blanco, F. Bernard, and P. Gayet, "Unicos: An open framework," in *12th ICALEPCS Int. Conf. on Accelerator and Large Expt. Physics Control Systems. Grenoble (France)*, 2009, pp. 12–16.

[12] S. Schlenker, C.-V. Soare, D. Abalo Miron, V. Filimonov, B. Farnham, and P. Nikiel, "Quasar-a generic framework for rapid development of opc ua servers," 2015.

[13] M. G. D. Gololo, "Soc developments for the detector control system of atlas tile calorimeter at the hl-lhc," System-on-Chip 2nd Workshop - CERN, 2021. `https://indico.cern.ch/event/996093/`

[14] E. Smith, "Zynq us+ mpsoc in the gfex hardware trigger in atlas," System-on-Chip 2nd Workshop - CERN, 2021. `https://indico.cern.ch/event/996093/`

[15] P. Moschovakos, “Socs for detector controls and their applications,” System-on-Chip 2nd Workshop - CERN, 2021. https://indico.cern.ch/event/996093/

[16] J. Hast, “Ess use case of the crome monitor with epics,” 2nd System-on-Chip Workshop - CERN, 2021. https://indico.cern.ch/event/996093/

[17] *EPICS*, https://epics-controls.org/.