REAL-TIME FRAMEWORK FOR ITER CONTROL SYSTEMS

W. Lee†, A. Zagar, B. Bauvir, T. Tak, ITER Organization, St. Paul Lez Durance Codex, France
A. Winter, Max Planck Institut für Plasmaphysik, Greifswald, Germany
M. Knap, P. Karlovsek, Cosylab d.d., Ljubljana, Slovenia
S. Lee, Korea Institute of Fusion Energy, Daejeon, Republic of Korea
P. Perek, D. Makowski, Lodz University of Technology, Lodz, Poland

Abstract

The ITER Real-Time Framework (RTF) is a middleware providing common services and capabilities to build real-time control applications in ITER such as the Plasma Control System (PCS) and plasma diagnostics.

The RTF dynamically constructs applications at runtime from the configuration. The principal building blocks that compose an application process are called Function Blocks (FB), which follow a modular structure pattern. The application configuration defines the information that can influence control behaviour, such as the connections among FBs, their corresponding parameters, and event handlers. The consecutive pipeline process in a busy-waiting mode and a data-driven pattern minimizes jitter and hardens the deterministic system behaviour. In contrast, infrastructural capabilities are managed differently in the service layer using non-real-time threads. The deployment configuration covers the final placement of a program instance and thread allocation to the appropriate computing infrastructure.

In this paper, we will introduce the architecture and design patterns of the framework as well as the real-life examples used to benchmark the RTF.

INTRODUCTION

The Plasma Control System is a dominant factor for the ITER pulsed operation, it controls all aspects of the plasma discharge from powering the superconducting magnets up to plasma termination [1]. PCS takes data from sensors and applies sophisticated algorithms to generate commands that are sent to actuators to control plasma parameters, such as position, shape or stability in a real-time context. Design, development and verification of real-time software in general is a complex and often lengthy process requiring multiple iterations until all timing relationships are satisfied and the application is stable and predictable.

The RTF is a flexible high-performance software base that facilitates the development and deployment of complex real-time applications [2]. Originally developed with the aim of control algorithms, the RTF can also be the basis for real-time data processing applications in ITER diagnostic systems.

The architecture design fully considered the modularity and portability of the software, and is applicable and extendable even in none-ITER environments. It hides many details specific to real-time systems (e.g., thread management, inter-thread data transfers, etc.), making the design and development of real-time software much easier and faster.

† email address: woongryol.lee@iter.org

Strict Quality Assurance (QA) process and code audits enforced software integrity to bring reliable system operation. Along with the EPICS pvAccess interface that enriches functionality for operation, the Simulink wrapper block allows control model transition from the design to the application in an agnostic way.

ARCHITECTURE

Overview

The RTF infrastructure provides a modular, fully abstracted environment with the following key features [3]:

- FBs are self-contained and do not have any dependency on hardware, inputs, and outputs or operating system within the code. All relevant information for the modules is delivered via configuration, fully reusable in any context.
- Full configurability of FBs, which can be chained together at the developer's discretion by configuration.
- Fully data-driven workflow. The FBs can be scheduled automatically based on the availability of input data.
- Configuration-based distribution of processing logic over different threads, processes and computer nodes (hosts).
- Support integrated operability using generated code from graphical system modelling tools (e.g. Simulink [4]).
- Full integration with ITER Control Data Access and Communication (CODAC). Out of the box support for multiple interfaces to other CODAC components (e.g. networks, archive, supervision, etc.).

Figure 1 shows the architecture of the RTF and a Real Time (RT) application including their main elements and how they interact. The main elements are:

- The **RT application** contains the processing logic that runs on different threads, processes or computer nodes (hosts) and contains:
  - The scheduler handling the execution of processing of FBs.
  - The FBs representing an operation with inputs and outputs.
  - The gateways responsible for ensuring that the data is transported between the FBs running in different threads, processes or nodes.
  - The RT applications running within multiple instances of the real-time process.

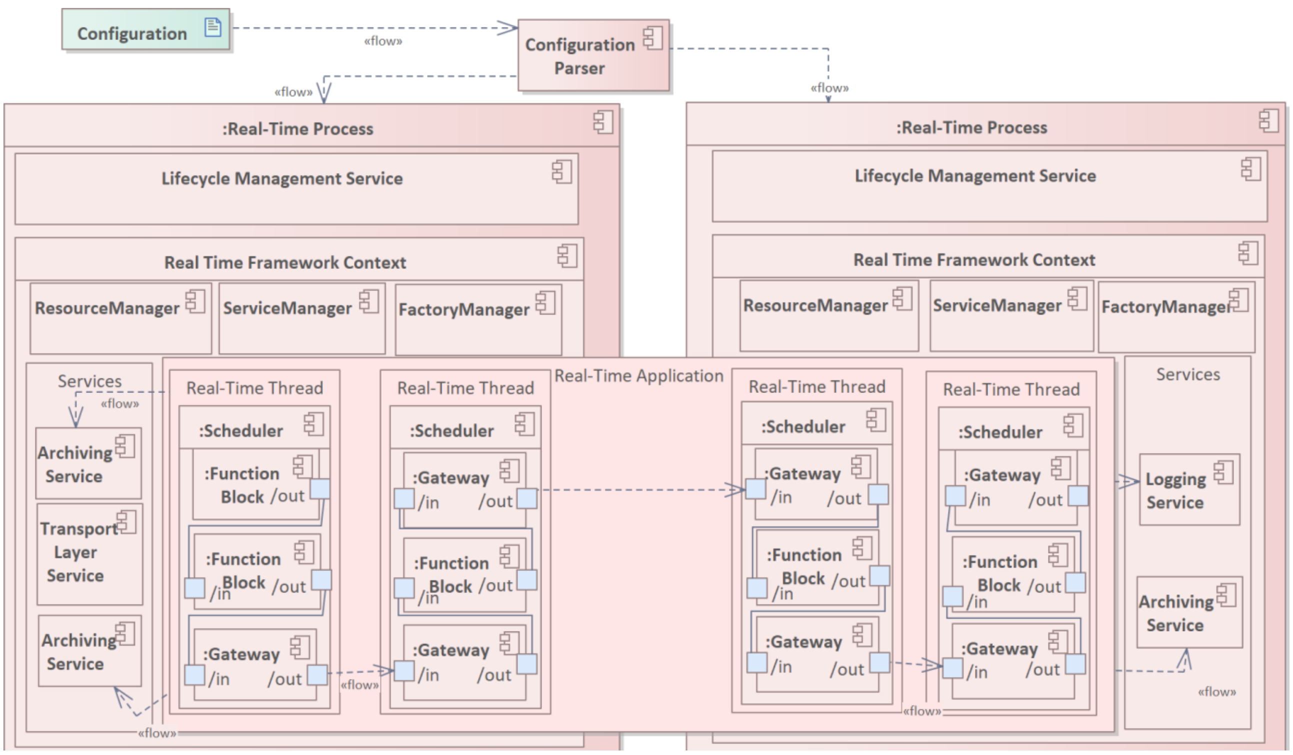


Figure 1: Context diagram of the Real-time Framework.

- The **real-time process** is the main process that runs on a node. This process has been specifically configured for real-time. It contains the following:
  - A life cycle management service managing the life cycle of the framework.
  - The real-time framework context - the key element of the real-time process consisting of Buffer, Queue and Thread Managers that help with tasks like allocating memory, transporting data and executing FBs. RTF context also includes Threads that execute the RT application and Services that provide high-level functionality for the RT application, e.g., logging and archiving.

RT Application

Figure 2 shows a simple RT application that acquires data from two inputs, adds them together and applies them to an output. Each component in Fig. 2 represents a FB, and the links show the connection between input ports (on the left of the component) to output ports (on the right of the component). The framework handles the dependency-based execution of FBs in either single-threaded or multi-threaded environments as specified in the deployment configuration.

Function Block

The Function Block is a representative atomic component to build application. The FB is similar terminology to the Blocks in the Simulink or the Functions in the LabView [5].

The FB is a function responsible for the desired operation, from primitive to complex process algorithms,

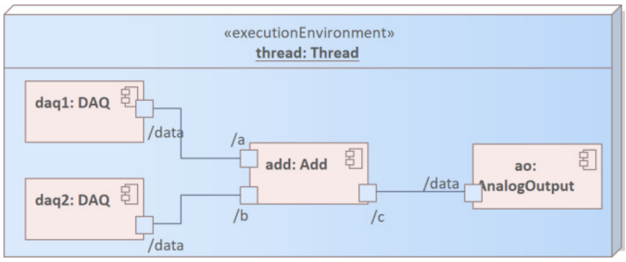


Figure 2: Example of an RT application executing in a single thread.

depending on the design intent. Each FB contains essential interface points: parameters, input signals and output signals that are constructed using a factory design pattern. Additonally it supports event trigger/handler along with prarmeter writers for asynchronous operation shown in Fig. 3.

FBs can be delivered as shared libraries and loaded to RTF-based applications in run-time as dynamic plugins. It allows decoupling user applications from core of the RTF. Designers can develop, maintain and share their custom libraries under different lifecycles and purposes. Only few of common FBs such as reading from and wrting to console or file, and basic mathematical operations are provided with the framework.

FBs can encapsulate other FBs giving the RT application an apparent hierarchical structure. In the case of composite FBs, such encapsulation can either have functional implications on e.g., scheduling or conditional execution of contained FBs, or is used to simply group FBs for entirely conceptual or convenience reasons.

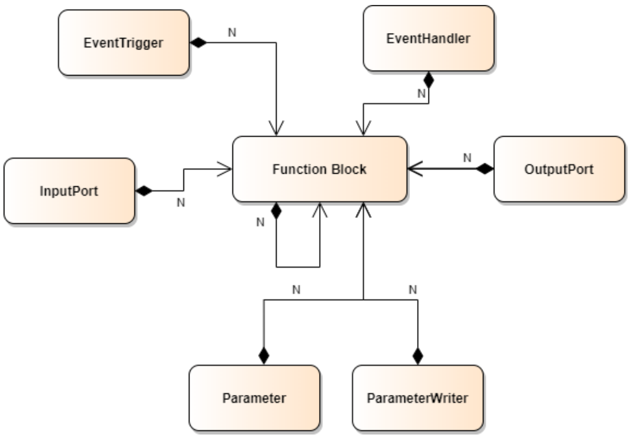


Figure 3: Schematic of a function block.

Processing of Function Blocks in a Thread

FBs are instantiated and serialized in the loading phase of the RTF context. Thereby, multiple function blocks execute as a chain on the thread where they are deployed. The ordering of function blocks is determined by the relation between function blocks once the configuration is parsed.

Synchronization in real-time threads executing the FBs is implemented with busy-wait rather than through interrupts or callbacks to avoid context switching and effectively minimize jitter and response times. Thus, RT application runs properly once it has an appropriate CPU allocation, schedule policy and priority attributes.

On start (or re-entry) of the thread, all FBs are reset. Then, the thread enters the active state, which has the following stages (Fig. 4).

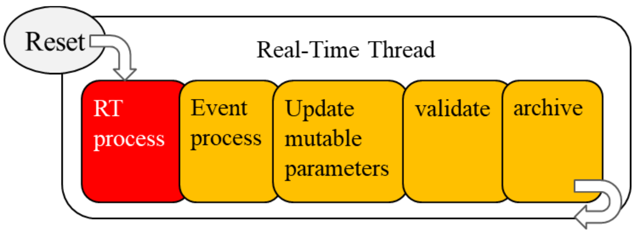


Figure 4: Processing of Function Block in a Thread.

- **RT process**: handles processing of the logic. All the inputs/outputs get updated in this stage.
- **Process events**: handles processing of all the events that have been triggered.
- **Process parameters**: updates the mutable parameters.
- **Validate**: validates the execution times and/or parameters to be validated.
- **Archive**: archives signals and all archivable objects.

The RT process is ideal for a hard real-time mode operation and should be constant over all the cycles. The underneath rule of execution of FB is to execute process method periodically under rt-thread, and thus ensure predictability of execution times. Fully data-driven mechanism requires the FB processing logic to be implemented in a non-blocking way. The remaining four consecutive jobs are executed in the offline real-time mode.

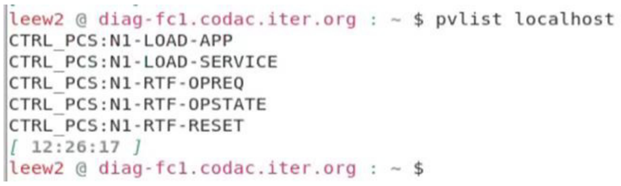
Framework Life Cycle Management

Since the framework is a part of a distributed control system, multiple instances must be centrally controlled in an organized manner. The Life Cycle Management Service (LCMS) allows to centrally orchestrate the state transition along with loading of the configuration.

In order to increase performance and reliability, it is desired to maintain the core real-time engine of RTF as lightweight as possible, and handle all configuration loading, parsing and expansion logic in an external system/process. Portability is also ensured by exposing LCMS interfaces through an API which allows site-specific protocol implementations. Two protocols are currently supported; pvAccess protocol from EPICS v7 and native TCP/IP.

Although EPICS has been selected as a standard middleware for ITER control, the evolution of configuration propagation and the need for improvement of code in terms of the software QA process have been raised. RTF renovated the primarily interface function based on the PVXS [6] as thereby.

Figure 5 shows limited number of PVs that are created just after instantiation of the RTF and are ideal for operational control. Fig. 6 depicts RTF state machine operation addressed through the {}-RTF-OPSTATE PV. After loading the configuration, additional application specific PVs will be dynamically created and published. In order to support client in a conventional manner, RTF uses the normative type, which creates a structure holding the value, alarm, and time structure.



```
leew2 @ diag-fc1.codac.iter.org : ~ $ pvlist localhost
CTRL_PCS:N1-LOAD-APP
CTRL_PCS:N1-LOAD-SERVICE
CTRL_PCS:N1-RTF-OPREQ
CTRL_PCS:N1-RTF-OPSTATE
CTRL_PCS:N1-RTF-RESET
[ 12:26:17 ]
leew2 @ diag-fc1.codac.iter.org : ~ $
```

Figure 5: Default PVs shown by EPICS bundle Command Line Interface (CLI) tool, which are minimized for configuration transmission and operation after RTF instantiation.

CONFIGURATION AND DEPLOYMENT

The RTF provides four types of configurations in an XML-based format.

- **Environment configuration** defines the computer nodes (hosts), processes and threads that host the RTF services and RT applications.
- **Service configuration** defines the services of the RTF that provide infrastructural capabilities such as a console, logging, archiving and communications between processes and across hosts.
- **RT application configuration** defines the RT applications in terms of configurable interlinked FBs that take inputs, perform processing and produce outputs.
- **Deployment configuration** assigns the services and RT applications to threads defined by the environment.

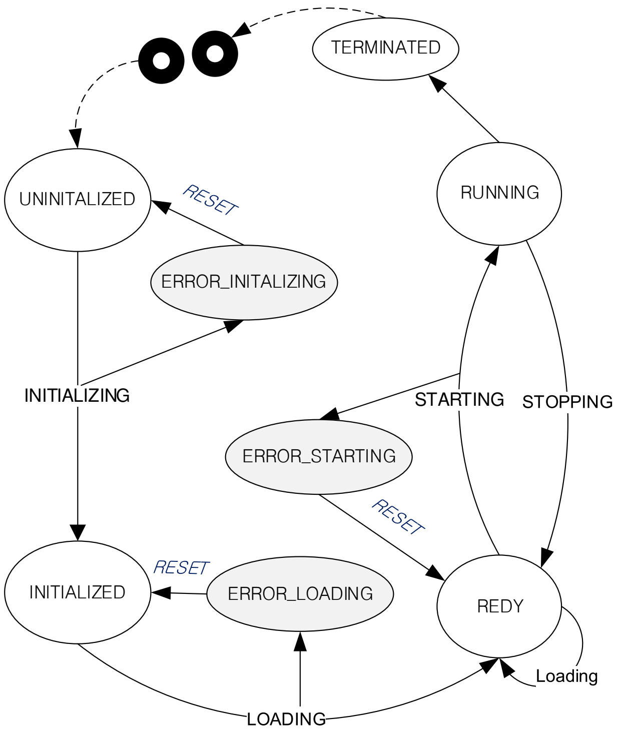


Figure 6: State diagram controlled by LCMS. *{PRFIX}-RTF-OPSTATE* PV indicates RTF state transition.

For the pvAccess based LCMS, the environment and service configuration is transmitted via *"{PREFIX}-LOAD-SERVICE"*, and *"{PREFIX}-LOAD-APP"* is assigned for the application and deployment configuration. Since the application configuration is decoupled from the deployment configuration, the RT application can be run on different computing resource defined in the environment configuration without any changes to its configuration or code even in pulse-to-pulse operation, while the environment and service configuration likely unnecessary to change.

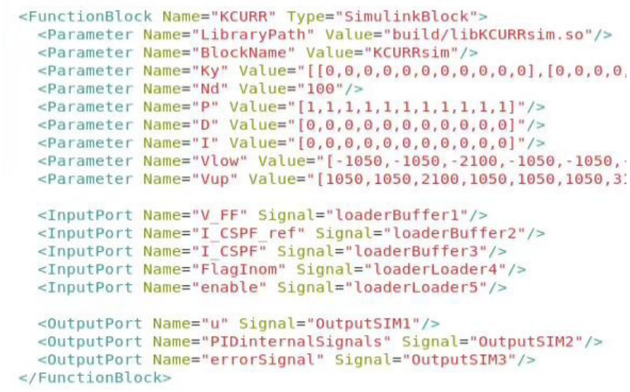
Configuration Parser

The Configuration parser parses the input XML configuration files and inserts any necessary logic to create a fully expanded configuration. In general, parser deals with whole configuration, then each RTF process extracts those parts of configuration which they are supposed to run. This fully expanded configuration is then converted to an in-memory representation of the configuration which is used by the real-time process. The parser is decoupled from the framework. The RTF provides C++ API for facilitating it into other programs as standalone console executable, *rtftool*, or other supervision application, Supervision (SUP) in ITER.

INTERFACE SUPPORT

Great emphasis has been put into the RTF to fully integrate it into the CODAC and to support all main interfaces such as Synchronous Databus Network (SDN), Data Archiving Network (DAN), and EPICS pvAccess protocol. The Nominal Device Support (NDS) service which is an intermediate layer for the physical hardware interfacing is under development. Foundational networking infrastructure is implemented as a transport layer service and therefore supports communication on all levels of RTF.

The RTF also supports Simulink interface based on automatic code generation function. A direct transfer of a Simulink model to a real-time framework FB can be achieved and successfully executed. Fig. 7 shows how compiled library from Simulink can be used inside RTF through the application configuration. This offers an extremely powerful method to obtain the necessary code directly from a model.


```
<FunctionBlock Name="KCURR" Type="SimulinkBlock">
  <Parameter Name="LibraryPath" Value="build/libKCURRsim.so"/>
  <Parameter Name="BlockName" Value="KCURRsim"/>
  <Parameter Name="Ky" Value="[[0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,
  <Parameter Name="Nd" Value="100"/>
  <Parameter Name="P" Value="[1,1,1,1,1,1,1,1,1,1,1]"/>
  <Parameter Name="D" Value="[0,0,0,0,0,0,0,0,0,0,0]"/>
  <Parameter Name="I" Value="[0,0,0,0,0,0,0,0,0,0,0]"/>
  <Parameter Name="Vlow" Value="[-1050,-1050,-2100,-1050,-1050,
  <Parameter Name="Vup" Value="[1050,1050,2100,1050,1050,1050,3

  <InputPort Name="V_FF" Signal="loaderBuffer1"/>
  <InputPort Name="I_CSPF_ref" Signal="loaderBuffer2"/>
  <InputPort Name="I_CSPF" Signal="loaderBuffer3"/>
  <InputPort Name="FlagInom" Signal="loaderLoader4"/>
  <InputPort Name="enable" Signal="loaderLoader5"/>

  <OutputPort Name="u" Signal="OutputSIM1"/>
  <OutputPort Name="PIDinternalSignals" Signal="OutputSIM2"/>
  <OutputPort Name="errorSignal" Signal="OutputSIM3"/>
</FunctionBlock>
```

Figure 7: Example how Simulink wrapper FB is configured in the application configuration, which loads the compiled library. Non-scalar data types are also fully supported.

CASE STUDY USING RTF

Multiple use case studies are being conducted from the IO and fusion community in order to identify possible limitations and improvements in parallel with implementation of RTF.

PCS Prototyping

Prototyping is an efficient way to cross-check the detailed intention of design and its implementation. The PCS components are sufficiently modularized to support as other systems become available, however the architecture remain constant for ITER lifetime. Thus it ideal for RTF capturing specific function individually and integrated efficiently along with project progress.

The PCS Compact Controller (CC) is a continuous controller among architectural components for versatile control purpose. It is representative as a Proportional, Integral, and Derivative (PID) controller with fine-tuned attributes tailored for plasma parameter control. As first work, CODAC group implemented 19 FBs including trajectory block and plant model for the simulation. The FBs are nearly map to the individual Simulink block, verified fidelity of outputs between Simulink and RTF traces [7].

Since the PCS design development is based on components running within the Simulink platform, it is under consideration the possibility to convert components from Simulink model into components compatible with the RTF in a quasi-automatic manner [8].

Consequently, Simulink wrapper function had been devised and verified proper transposition from model to application with high fidelity of result. Customised FB, *"SimulinkBlock"*, loads the compiled library, which was automatically built by the RTF CLI.

Underlying concept is supporting the mutable parameter in the same manner as conventional FB. Therefore, model designer configures the desired function only by changing parameters, while maintaining the same external interface to the other FB. Even if model changed, thereby recompile the generated code, RTF FB always persists independently.

Plasma Diagnostics Data Processing

Edge Thomson Scattering (ETS) diagnostics has been demonstrated in the running tokamak. The Thomson Scattering diagnostics gives reliable electron temperature and density profiles in magnetically confined plasma. A 5GS/s CAEN DT5742 digitizer [9] operates in pulse mode synchronized with Nd:YAG Laser system where has up to 50Hz injection rate [10]. The customized data acquisition FB archives raw data through RTF transport layer whilst the output links to the fitting FB to eliminate back scattered signal. Passed series of signal conditioning, electron temperature is measured using lookup table where calibrated data is stored as per the wavelength from the polychromator signal.

The developed prototype covering complete data acquisition, processing path, archiving as well as measurements publishing can be used as a reference example for other ITER diagnostic systems.

Poloidal Field Coil Control for Plasma Start-up

The most fundamental control module of PCS is coil power supply for discharge control. Poloidal Field (PF) coil is a main actuator for plasma breakdown and thereafter shape and position control. CODAC commenced implementing a real-life controller in order to evaluate both functional and non-functional behaviour of the PCS with collaboration of Korea Superconducting Tokamak Advanced Research (KSTAR) control team.

The full-featured mini-CODAC provides all ITER standard networking protocols such as real-time data communication, experimental data archiver and time synchronization. Additional installation of the RTF along with the PCS platform library facilitates building controller following the PCS architecture design.

11 PF controllers were devised complying with KSTAR native function model. Minimum protection function took into accounted, verified 20kHz run cycle in consecutive process pipeline such as exception handler, waveform generator, and PID function. Integrated operability was verified by implementing site-dependent interface functions such as for MDSplus, Reflective Memory network, and EPICS CA.

CONCLUSION

The RTF is a flexible high-performance software platform that facilitates the development and deployment of complex real-time applications. It was designed to be portable and modular, enabling high reusability and maintainability of components constituting the real-time applications.

Factory design pattern, and rich function for multi-threaded program enables building application through configuration-driven process.

Prototyping activities on some of the operating Tokamaks have demonstrated its applicability for the implementation of ITER real-time control systems.

ACKNOWLEDGEMENTS

The views and opinions expressed herein do not necessarily reflect those of the ITER Organization.

REFERENCES

[1] J.A. Snipes *et al.*, "ITER Plasma Control System Final Design and Preparation for First Plasma", *Nuclear Fusion,* vol. 61, no. 10, Sept. 2021. doi:10.1088/1741-4326/ac2339

[2] A. Winter *et al.,* "The ITER real-time framework final design", presented at the 11th IAEA Technical Meeting on CODAC, Greifswald, Germany, May 2017.

[3] W. Lee *et al.,* "Software Architecture and Design Document for the ITER Real-time Framework", ITER Internal Communication.

[4] Simulink, https://www.mathworks.com/help/simulink

[5] NI LabView, https://www.ni.com/labview.html

[6] https://mdavidsaver.github.io/pvxs/index.html

[7] L. Zabeo, B. Bauvir, W. Lee, *et al.*, "PCS Proto-typing activity", ITER internal communication.

[8] L. Zabeo *et al.*, "Work-flow process from simulation to operation for the Plasma Control System for the ITER first plasma", *Fusion Eng. Des.,* vol .146, pp. 1146-1149, 2019. doi:10.1016/j.fusengdes.2019.02.101

[9] https://www.caen.it/products/dt5742/

[10] J.H. Lee *et al.*, "Research of Fast DAQ system in KSTAR Thomson scattering diagnostic", presented at the 18th Laser Aided Plasma Diagnostics (LAPD18), Prague, Sep. 2017. doi:10.1088/1748-0221/12/12/C12035