Base Load Power Hub (BLPH)

Logic Diagram Print Out EMCS

Document Number: 1724000594-IN-DG-00027

Supplier Document Number: E366772C00-EMCS_LD

Document Class: 3

01	30/07/2025	Issue for Information (IFI)	CF	SA	VO
00	20/02/2025	Issue for Information (IFI)	CF	SA	VO
Rev.	Date	Description	Prepared	Reviewed	Approved

YOKOGAWA Co-innovating tomorrow™

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

Revision / Change Log

Rev.	Description
00	Issue for Information (IFI)
01	Issue for Information (IFI)
02	Issue for Information (IFI)

HOLDS LIST

No.	Description
1	
2	
3	
4	
5	
6	
7	

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

Table of contents

1 Introduction 5

1.1 Project Overview 5

1.1.1 Project Location 5

1.1.2 Project Concept 6

2 Document Scope 8

2.1 References 8

2.2 Definitions And Abbreviations 10

3 Software Architecture 12

4 Software Code 16

4.1 EMCS main variables 16

4.2 Main_CODE 23

4.3 FORMULAS 31

4.4 EMCS_MODE_ACTIVATOR 39

4.5 EMCS_MODE_SELECTOR 39

4.6 IDLE MODE_EXECUTOR 54

4.7 IMPORT CHARGING MODE EXECUTOR 58

4.8 IMPORT NOT ABLE TO CHARGE EXECUTOR 61

4.9 HOTSTANDBY MODE_EXECUTOR 64

4.10 EXPORT MODE_EXECUTOR 67

4.11 COLDSTANDBY MODE EXECUTOR 71

4.12 EMERGENCY MODE EXECUTOR 74

4.13 POWEROUT MODE EXECUTOR 78

4.14 MAINTENANCE MODE EXECUTOR 82

4.15 RESEARCH MODE EXECUTOR 85

4.16 NONE MODE EXECUTOR 89

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

List of Figures

Figure 1 HOLLANDSE KUST NOORD WIND PARK Location Map 5
Figure 2 - Wind Park Details 6
Figure 3 – BLPH Concept 7
Figure 4 – EMCS Mode Selection 13
Figure 5 – Mode state machine and transitions 14

List of tables

Table 1 - Reference Documents 9
Table 2 - Definitions and Abbreviations 11

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

1 INTRODUCTION

1.1 PROJECT OVERVIEW

Crosswind, a Shell and Eneco joint venture, was awarded the tender to develop the offshore Hollandse Kust Noord (HKN) wind farm, located 18.5km off the coast of Netherlands near the town of Egmond aan Zee. The consortium plans to have HKN operational in 2023 with an installed capacity of 759MW generating at least 3.3TWh per year. This is sufficient renewable power to supply more than 1 million Dutch households with green electricity.

Offshore wind farms produce intermittent electricity and balancing these dips and peaks and integrating this electricity into the national energy system requires new technologies. For this reason, the offshore wind farm will include several new technologies that could be implemented at full-scale in the future. The new technologies being explored include a floating solar park, a short-duration battery storage as well as long-term storage via green hydrogen produced by water electrolysis and subsequently consumed in a fuel cell. The combination of these individual measures, herein referred to as the Base Load Power Hub (BLPH) will ensure a continuous power supply regardless of the wind situation.

1.1.1 Project Location

The Hollandse Kust Noord (HKN) wind farm is located 18.5km off the coast of the Netherlands near the town of Egmond aan Zee.

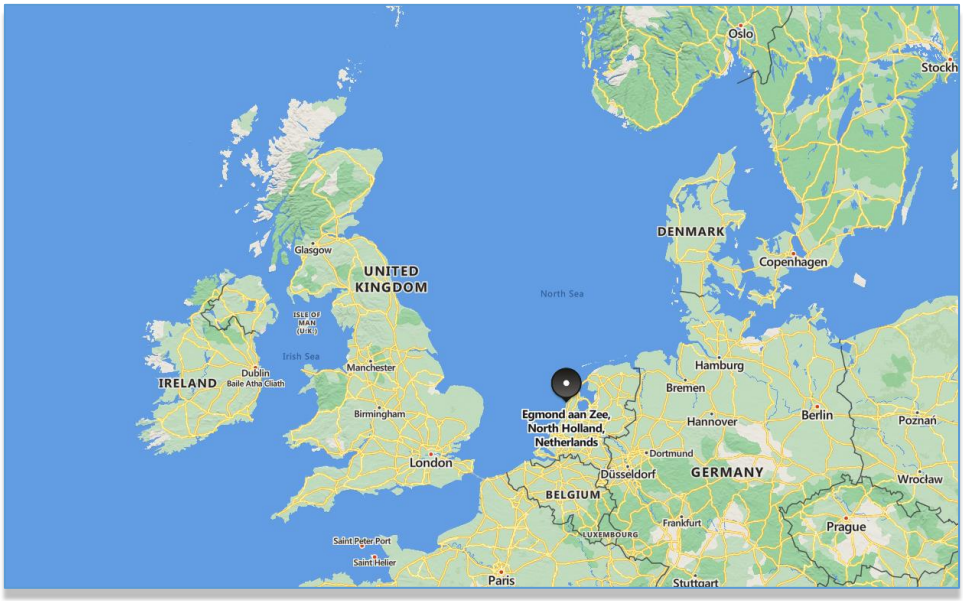


Figure 1 HOLLANDSE KUST NOORD WIND PARK Location Map

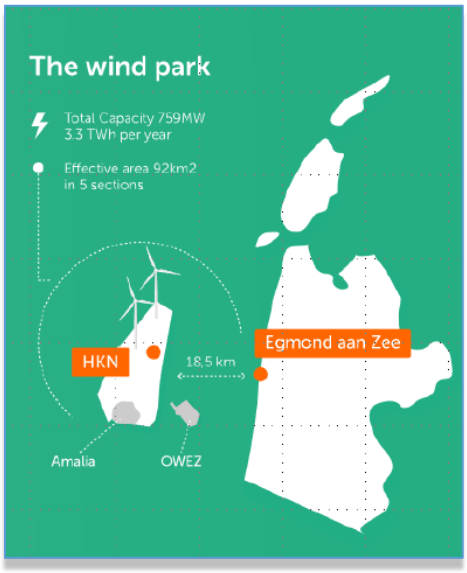


Figure 2 - Wind Park Details

The wind farm will consist of No. 69 x 11.5MW wind turbine generators each to be installed on a dedicated monopile structure. A 70th monopile located on coordinates E 590719.1, N 5841602.3 is reserved for installing the platform accommodating the BLPH. The monopile structure was designed and installed by Van Oord Offshore Wind (VooW) and therefore not part of the BLPH scope.

1.1.2 Project Concept

All components included in the BLPH will be integrated onshore on a topsides and installed in an offshore location within the boundaries of the HKN wind farm. At its simplest, the BLPH will store energy during periods of high electricity generation from renewable sources (a wind turbine generator and the floating solar PV farm) and release it during periods of low generation. In practice. It is expected that the energy stored will supplement generation from renewable sources to ensure that at least 1.0MW of power is exported to the electrical grid regardless of wind/solar availability.

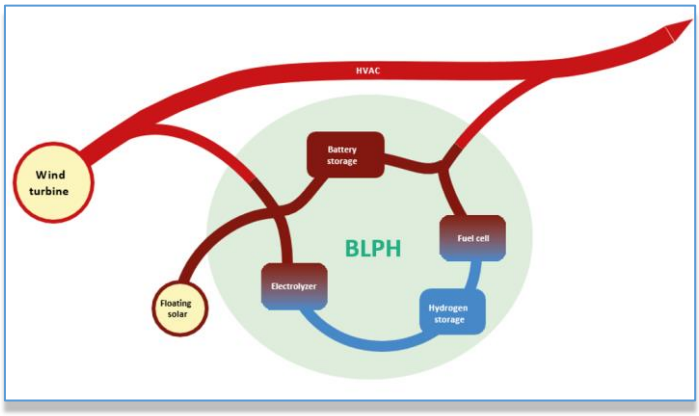


Figure 3 – BLPH Concept

Electricity in the BLPH will be stored in the form of electrons in a battery energy storage system (BESS) as well as converted into green hydrogen via water electrolysis. When the power output from the connecting wind turbine drops to below 1 MWe, the BLPH will return its stored energy to supplement the remaining power output. This will be by discharging the BESS, while stored hydrogen will be converted back into electricity using a fuel cell to extend the discharge duration.

The purpose of the BLPH s to function as a testbed for the integration of energy storage technologies with renewable generation. It is expected that equipment installed on the BLPH will operate as Normally Unmanned Installation for at least two (2) years with no or minimal maintenance and it will have its own independent control room with the ability to communicate with Crosswind's control room.

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

2 DOCUMENT SCOPE

The purpose of this document is to provide a general description of the control logic implemented in the Energy Management and Control System (EMCS) of the CROSSWIND BLPH project.

2.1 REFERENCES

Document No.	Description
1724000594-IN-SP-00019	EMCS Software Functional Design Specification (FDS-SW)
123A90-DDL-IN-RE-40095	Process Control Logics Description
123A90-DDL-EL-SP-20141	Electrical System Description & Design Philosophy
123A90-DDL-IN-SP-40098	Control System (PCS) Specification
123A90-DDL-EL-RE-20176	Electrical System Study Report - Transient In-Rush
123A90-DDL-EL-RE-20171	Electrical System Study Report - Short Circuit
K559-2254-PM-PLN-003	Basic Design Engineering Package BLPH FEED
123A90-PML-PM-TQ-00050	Reactive power control requirement for BLPH
123A90-DDL-EL-DG-20158	Overall/Key Single Line Diagram
1724000594-IN-DD-00015	System Architecture
SRN – 04270	Operations Modes Review – Base Load Power Hub
123A90-DDL-IN-SP-40097	System Specification for Main Automation Contractor
123A90-DDL-IN-SP-40100	Safety System (ESD, Fire and Gas) Specification)
123A90-MOM-RM-STAT-CW-003	Minute of Meeting - Freqcon & Statron-Control Workshop with Rosetti's MAC
123A90-MOM-RM-STAT-CW-003	123A90 - Interface control workshop with Rosetti MAC-BESS-Point 9 and 10-Synchronization Procedure-Via Mail
123A90-PML-PM-TQ-00053	Technical Query – Interface with BESS
123A90-MOM-ELO-ROS-CW-003	Minute of Meeting- Electrolyzer- CPI Interface Meeting

123A90-PML-PM-TQ-00052	Technical Query – Interface with Electrolyzer
123A90-PML-PM-TQ-00051	Technical Query – Interface with Fuel Cell
123A90-MOM-RM-BAL-CW-004	Minute of Meeting – Control Workshop with Rosetti's MAC.
123A90-PML-PM-TQ-00055	Technical Query – Interface with Solar Farm
123A90-PML-PM-TQ-00054	Technical Query – Interface with Windfarm and HNA5
123A90-MOM-ELE-ROS-CW-023	Minute of Meeting - Control Workshop with Rosetti's MAC
K559-2254-CWIWP4202201-C08-001	BESS - Control Philosophy & Block Logic Diagrams
C63-AUT-002	Electrolyzer – Communication Table
K559-2254-CWIWP4202202-C08-001	BALLARD- Control Philosophy & Block Logic Diagrams
CWHKN-INN-INP-SPC-00005	SCADA Functional Design Specification
123A90-DDL-EL-LIS-20016	Electrical Load List & Balance (CCS MODE 1 'IMPORT')
123A90-DDL-EL-LIS-20143	Electrical Load List & Balance (CCS MODE 2 'HOT STANDBY')
123A90-DDL-EL-LIS-20144	Electrical Load List & Balance (CCS MODE 3 'EXPORT')
123A90-DDL-EL-LIS-20145	Electrical Load List & Balance (CCS MODE 4 'COLD STANDBY')
123A90-DDL-EL-LIS-20146	Electrical Load List & Balance (CCS MODE 4A 'EMERGENCY MODE')
23A2007_240628_PROFINET_Register BESS_V1_0_0_3	BESS_PROFINET_SIGNAL LIST
K559-2254-CWIWP4202201-J10-001	FREQCON MSC - Product Description
MAN002 Integration Manual FCwave 2.2.0	FC Wave Marine Fuel Cell Integration Manual
CW-BALL-TQ-0008 - Fuel Cell - Rev B - BAL RESPONSE	Minute of Meeting Response
Crosswind Modbus Table - Main PLC - External Cut	Fuel Cell MODBUS Map
HKN1-02-REG-341	Manual for controlling the solar farm via the client connector REV A

Table 1 - Reference Documents

2.2 DEFINITIONS AND ABBREVIATIONS

ALARP	As Low as Reasonably Possible
AVR/OLTC	Automatic Voltage Regulator/On Load Tap Changer
BESS	Battery Energy Storage System
BLPH	Base Load Power Hub
BPCS	Basic Process Control System
COMPANY or Client (CPY)	Crosswind
CCS	Central Control System (PCS + EMCS)
CPI	Company Provided Item
DCS	Distributed Control System
ESD	Emergency Shut down
EWS	Engineering Workstation
EMCS	Electrical Management Control System
FEED	Front-end Engineering Design
FGS	Fire & Gas System
FO	Fiber Optics
GUI	Graphical User Interface
HMI	Human Machine Interface
HKN	Hollandse Kust Noord
HV-SCADA	The given name for the COMPANY's control system for the HKN wind farm infrastructure. BLPH's control system shall operate autonomously but will interface with HV-SCADA
HSE	Health, Safety & Environment
HV	High Voltage
HVAC	Heating, Ventilation and Air Conditioning
IACS	Industrial Automation & Control System
ITER	Instrument & Telecoms Equipment Room (located Offshore)
LV	Low Voltage
MV	Medium Voltage
OER	Onshore Equipment Room
PAS	Process Automation System
PFD	Process Flow Diagram
P&ID	Piping & Instrumentation Diagram
PSD	Process Shut Down
PV	Photo Voltaic
SCADA	Supervisory Control and Data Acquisition
SoC	State of Charge
SF	Solar Farm

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

SS	Safety System (SIS+FGS)
TQ	Technical Query
UCP	Unit Control Panel
UFD	Utility Flow Diagram
UPS	Uninterrupted Power Supply
WTG	Wind Turbine Generator

Table 2 - Definitions and Abbreviations

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

3 SOFTWARE ARCHITECTURE

The EMCS control logic is written using the set of language of IEC 61131-3 standard and run in SEL RTAC 3555 industrial computers (PLC).

The basic concept of the EMCS is that the system is in one of the modes defined in the

FDS (1724000594-IN-SP-00019) :

- IDLE
- IMPORT CHARGING
- IMPORT NOT ABLE TO CHARGE
- HOT STANDBY
- EXPORT
- COLD STANDBY
- EMERGENCY
- POWER OUT
- RESEARCH
- MAINTENANCE
- NONE

When a mode is selected, the EMCS execute a set of routines to perform the actions (start and stop packages, impose setpoint, open or close CBs, …) required by the execution logic defined for the mode.

The Figure 4 shows a diagram representing the software components involved in the mode selection:

- EMCS Mode Processor: responsible for evaluating the precondition and to select the current mode. Internally it contains two sub components: the Mode Selector and the Mode Activator.
- Real-Time Data Cache: responsible to store the snapshot of the field variables
- Protocol Drivers: a set of clients implementing a specific protocol (MODBUS, IEC 61850, OPC-UA) responsible to collects the data from the field devices
- Field Devices: are an abstraction of the field devices
- Mode[i]: software components responsible to execute the control logic of a specific mode.

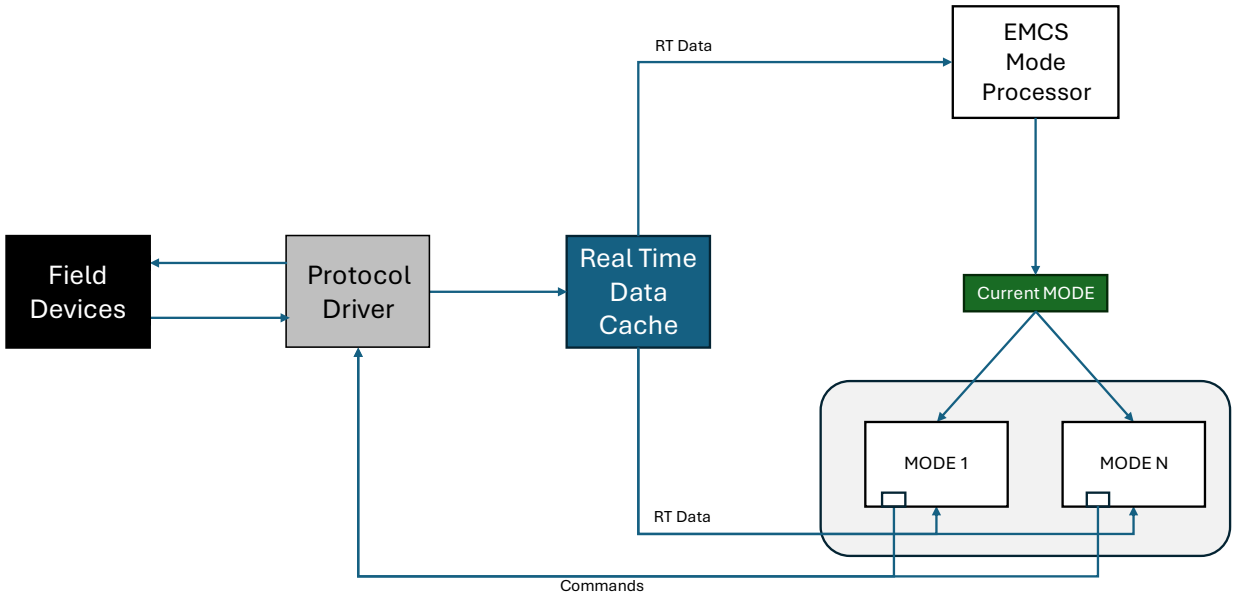


Figure 4 – EMCS Mode Selection

Each mode is implemented as an independent Function Block with its internal state machine. The picture Figure 5 shows the status of a mode and the process involved in the transition between one mode and another. The picture shows as an example the Research and Maintenance mode, but the concept can be extended to all modes.

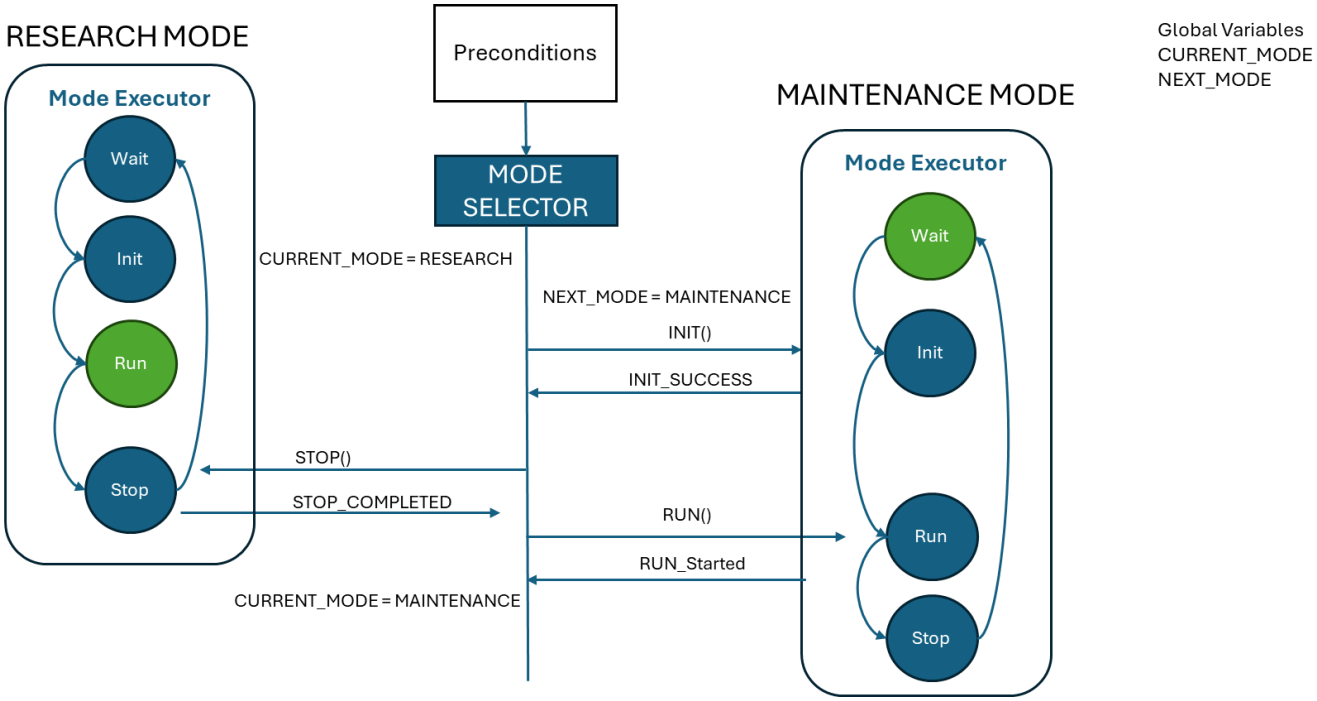


Figure 5 – Mode state machine and transitions

In the above example, the Mode Selector block continuously evaluates the preconditions received as input to establish if a MODE transition is required. The Mode Selector uses two global variables:

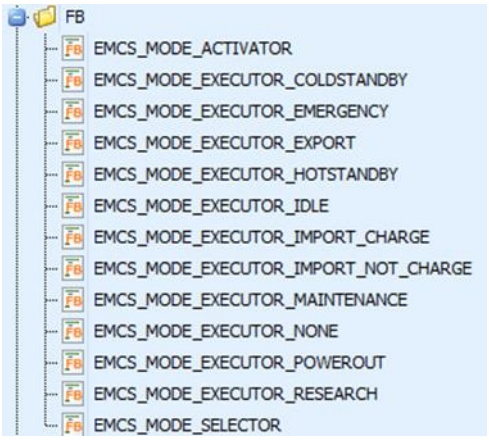
- CURRENT_MODE: reports on what is the current executing MODE
- NEXT_MODE: reports on what the next MODE to execute should be based on the status of the preconditions.

In the example, the precondition regarding the EMCS Manual-Automatic Mode is changing from Research to Maintenance. This happens when the SCADA operator switches manually the AUTO-MANUAL mode selector from Research to Maintenance MODE.

Whenever the NEXT_MODE differs from the CURRENT_MODE, a transition is initiated. During the transition, first it is evaluated if the transition to the NEXT MODE can be performed.

In the example, the Mode Selector calls the *init* function of the Maintenance Mode Executor to check if the transition can be done. If the *init* function returns success, the Mode Executor calls the stop function to the Research Mode executor and then calls the *run* function to the Maintenance Mode Executor. After the transition both CURRENT_MODE and NEXT_MODE variables report Maintenance MODE.

From the RTAC project point of view, each mode logic is contained in a dedicated file.



This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

4 SOFTWARE CODE

In this chapter the code regarding the EMCS MODEs transition and execution is printed out.

4.1 EMCS MAIN VARIABLES

```
VAR_GLOBAL
      IsMaster : BOOL;
      NodeId : INT;
      PlatformSupplyGuarantee : BOOL;          // Formula
      AuxLoad : REAL;                          // SG100D1_A4_TR_200DE1_RealPower
      GridTied : BOOL;
      GridVoltageOK : BOOL;                    // Tell if the voltage above the MV breaker is OK and
                                               // the BLPH can be reconnetced
      AutoManualMode : UINT := EMCS_AUTOMANUAL_ENUM.MANUAL_RESEARCH;   //   EMCS_AUTOMANUAL_ENUM   =>
                                               0:AUTO; 1:Maintenance; 2:Research
      ResidualOperationTime : REAL;
      TotalActivePower : REAL;                 // Balance of the BLPH
      BessFuellCellActivePower: REAL;          // BESS + FUELL CELL active power
      OperatorSupplyGuarantee : UINT;          // EMCS_SUPPLY_GUARANTEE: Command from SCADA

      HV_SCADA_Connected : BOOL;
      HNA5_ActivePower : REAL :=500;           // kW
      HNA5_Fault : BOOL;                       // formula
      HNA5_SupplyGuarantee : BOOL;             // Tells if there is a disconnection
      HNA5_StatusCode : INT;                   // WindTurbine status. WindFarm.HNA5-AFA78.StatusCode
      HNA5_QA1CB_Status : UINT;                // Signal coming from HV-SCADA (WindFarm)=> 0: Opened,
                                               1: Closed, 2: Intermediate, 3: Disturbed

      BESS_Connected : BOOL;
      BESS_DisconnectionTimer: INT;
      BESS_ShutdownTimer: INT;
      BESS_ReconnectionTimer: INT;
      BESS_IsReconnecting :BOOL;               // Is reconnectiong to the electrical BUS
      BESS_OperatingStatus : BESS_OPERATING_STATUS_ENUM;
      BESS_OperatingStatusCmd : INT ;          // Stop=0; Start=8192
      BESS_ResetCmd : BOOL ;                   // Reset=TRUE. Impulsive command. It is autoreset
                                               // after 3 seconds
      BESS_FullyCharged : BOOL;
      BESS_Healthy : BOOL;
```

```
BESS_HasVoltage : BOOL;
BESS_GridVoltageOK : BOOL;
BESS_ActivePowerSetPointCmd :REAL;
BESS_ActivePowerSetPoint :REAL;
BESS_ActivePower :REAL;
BESS_SOC : REAL;                              // %
BESS_AvailableEnergyPerc : REAL;              // %
BESS_DischargeEnergy : REAL;                  // kWh
BESS_ChargeEnergy : REAL;                     // kWh
BESS_GridFormingPermissive : BOOL;
BESS_GridConnectPermissive : BOOL;            // Operator removes the grid forming permissive
                                              // to go back in grid following
BESS_StatusAvailable : BOOL;                  // Status3 // True: Started; False:Stopped
BESS_StatusWarning: BOOL;                     // Status1
BESS_StatusError: BOOL;                       // Status1
BESS_StatusPhreHeat: BOOL;                    // Status1
BESS_StatusStandBy: BOOL;                     // Status1
BESS_StatusPreCharge: BOOL;                   // Status1
BESS_StatusReady: BOOL;                       // Status1
BESS_StatusActiveInOperation: BOOL;           // Status1
BESS_StatusSafetyRelaisTriggered: BOOL;       // Status1
BESS_StatusSmokeDetectorTriggered: BOOL;      // Status1
BESS_StatusMainSwitchOn: BOOL;

BUS_6_6kV_HasVoltage : BOOL;

Electrolyzer_Connected : BOOL;
Electrolyzer_DisconnectionTimer: INT;
Electrolyzer_ShutdownTimer: INT;
Electrolyzer_ReconnectionTimer: INT;
Electrolyzer_IsReconnecting :BOOL;            // Is reconnectiong to the electrical BUS
Electrolyzer_ActivePowerConsumption : REAL;   // This is mapped to
                                              //SG100D1_A1_ELECTROLYZER_RealPower. Doesn't
                                              // come from the Electrolyzer
Electrolyzer_ActivePowerSetPointCmd : REAL;   // Command for the Active Power set Point
Electrolyzer_FlowSetPointCmd : REAL;          // Command for the Flow set Point
Electrolyzer_FlowSetPoint : REAL;             // Return Flow set Point (feedbsack)
Electrolyzer_SetPointChoiseCmd  : INT;        // Command for choosing the type of set point.
                                              // 0:Power, 1:Flow
```

```
Electrolyzer_OperatingStatus : ELECTROLYZER_OPERATING_STATUS_ENUM := -1;
Electrolyzer_OperatingStatusCmd : INT;              // 1:Stop: 4:Production
Electrolyzer_OperatingStatusCmd_Private : INT;
Electrolyzer_ActivationThreshold : REAL;
Electrolyzer_KeepRunning : BOOL;                    // Flag indicating that the Electrolyzer
                                                    // cannot be stopped until it finish to
                                                    // charge the vessels
Electrolyzer_CanRun : BOOL;                  // Flag calculated in function of external
                                             // conditions (see FORMULAS)

FuelCell_Connected : BOOL;
FuelCell_RemoteCtrl: BOOL;
FuelCell_DisconnectionTimer: INT;
FuelCell_ShutdownTimer: INT;
FuelCell_ReconnectionTimer: INT;
FuelCell_IsReconnecting :BOOL;               // Is reconnectiong to the electrical BUS
FuelCell_ActivePower : REAL;                 // Fuel Cell Actual Production
FuelCell_ActivePowerSetPointCmd : REAL;      // kW
FuelCell_ActivePowerChangeRate : REAL;       // kW/s
FuelCell_ActivePowerTarget : REAL;
FuelCell_ActivePowerTargetInt : REAL;
FuelCell_OperatingStatusCmd  : INT;
FuelCell_OperatingStatusCmd_Private  : INT;
FuelCell_ResetCmd : BOOL ;                   // Reset=TRUE. Impulsive command. It is
                                             // autoreset after 3 seconds
FuelCell_OperatingStatus  : FUEL_CELL_OPERATING_STATUS_ENUM;
FuelCell_ConversionEfficiency : REAL;
FuelCell_Efficiency : REAL;                  // Number in the range [0.0 ; 1.0]
FuelCell_ResidualEnergy :REAL;
FuelCell_CanRun : BOOL;                      // Flag calculated in function of external
                                             // conditions (see FORMULAS)
FuelCell_FC1_State : FUEL_CELL_MODULE_STATUS_ENUM;
FuelCell_FC2_State : FUEL_CELL_MODULE_STATUS_ENUM;
FuelCell_FC3_State : FUEL_CELL_MODULE_STATUS_ENUM;
FuelCell_FC4_State : FUEL_CELL_MODULE_STATUS_ENUM;
FuelCell_FC5_State : FUEL_CELL_MODULE_STATUS_ENUM;
FuelCell_FC6_State : FUEL_CELL_MODULE_STATUS_ENUM;
```

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

```
SolarFarm_Connected : BOOL;
SolarFarm_DisconnectionTimer: INT;
SolarFarm_ShutdownTimer: INT;
SolarFarm_ReconnectionTimer: INT;
SolarFarm_IsReconnecting :BOOL;          // Is reconnectiong to the electrical BUS
SolarFarm_OperatingStatus : SOLAR_FARM_OPERATING_STATUS_ENUM;
SolarFarm_StartupCmd :BOOL;
SolarFarm_StartupCmd_Private :BOOL;
SolarFarm_ShutdownCmd :BOOL;
SolarFarm_ActivePower :REAL;
SolarFarm_ActivePowerSetPoint: REAL;
SolarFarm_ActivePowerSetPointCmd: REAL;
SolarFarm_Fault :BOOL;
SolarFarm_Maintenance :BOOL;
SolarFarm_CB_OpenRequest :BOOL;     //With this bit the solar farm requests the'EMCS to open a
                                    // CB
SolarFarm_CB_CloseRequest :BOOL;    //With this bit the sola farmr requests the'EMCS to close a
                                    // CB
SolarFarm_Healthy : BOOL;
SolarFarm_CanRun : BOOL;            // Flag calculated in function of external conditions (see
                                    FORMULAS)
SolarFarm_SendActivePowerSetPoint: SolarFarm_SendActivePowerSetPointFB;
SolarFarm_SetPointWaitTimer : TON;       // Timeout for the application of a set point

HNA5_P_GT_GUAR : PowerLevelStatus;
HNA5_P_LT_GUAR : PowerLevelStatus;
HNA5_P_GT_SURP : PowerLevelStatus;

// Switchgear of SOLAR FARM
SG200D2_Connected : BOOL;                // Connection Status. Mapped to the client of the
                                         // Switch Gear
SG200D2_CloseCmd  : BOOL;                // Command. Mapped to Solar Farm register. The Solar
                                         //  Farm uses this register to ask operation on CB

SG200D2_OpenCmd : BOOL;                  // Command. Mapped to Solar Farm register. The Solar
                                         // Farm uses this register to ask operation on CB
SG200D2_Closed  : BOOL;                  // Status

SG200D2_Open : BOOL;                     // Status
SG200D2_Alarm : BOOL;
```

```
      //=====================================================
      // SG100D1 Switch Gear
      //=====================================================
      SG100D1_Connected : BOOL;

      SG100D1_A1_ELECTROLYZER_CloseCmd    : BOOL;          // ELECTROLYZER Command

      SG100D1_A1_ELECTROLYZER_OpenCmd     : BOOL;          // ELECTROLYZER Command
      SG100D1_A1_ELECTROLYZER_ResetCmd    : BOOL;          // ELECTROLYZER Command

      SG100D1_A1_ELECTROLYZER_Closed      : BOOL;          // ELECTROLYZER Status

      SG100D1_A1_ELECTROLYZER_Open        : BOOL;          // ELECTROLYZER Status
      SG100D1_A1_ELECTROLYZER_RealPower   : REAL;          // ELECTROLYZER Active Power kW
      SG100D1_A1_ELECTROLYZER_Alarm       : BOOL;          //  ELECTROLYZER  Alarm  for  inconsistent
status

      // Solar Farm SG
      SG100D1_A2_TR_200ED1_CloseCmd       : BOOL;          // Solar Farm Command
      SG100D1_A2_TR_200ED1_OpenCmd        : BOOL;          // Solar Farm Command
      SG100D1_A2_TR_200ED1_ResetCmd       : BOOL;          // Solar Farm Command
      SG100D1_A2_TR_200ED1_Closed         : BOOL;          // Solar Farm Status
      SG100D1_A2_TR_200ED1_Open           : BOOL;          // Solar Farm Status
      SG100D1_A2_TR_200ED1_RealPower      : REAL;          // Solar Farm Active Power kW. Solar Farm
                                                           // Transformer
      SG100D1_A2_TR_200ED1_Alarm          : BOOL;          // Solar Farm Alarm for inconsistent
                                                           // status

      SG100D1_A3_BESS_CloseCmd            : BOOL;          // BESS Command
      SG100D1_A3_BESS_OpenCmd             : BOOL;          // BESS Command
      SG100D1_A3_BESS_ResetCmd            : BOOL;          // BESS Command
      SG100D1_A3_BESS_Closed              : BOOL;          // BESS Status
      SG100D1_A3_BESS_Open                : BOOL;          // BESS Status
      SG100D1_A3_BESS_RealPower           : REAL;          // BESS Active Power kW
      SG100D1_A3_BESS_Alarm               : BOOL;          // BESS Alarm for inconsistent status

      // LV Incoming Transformer
      SG100D1_A4_TR_200DE1_CloseCmd       : BOOL;          // LV Incomer TRAFO Command
      SG100D1_A4_TR_200DE1_OpenCmd        : BOOL;          // LV Incomer TRAFO Command
```

```
SG100D1_A4_TR_200DE1_ResetCmd       : BOOL;          // LV Incomer TRAFO Command
SG100D1_A4_TR_200DE1_Closed         : BOOL;          // LV Incomer TRAFO Status
SG100D1_A4_TR_200DE1_Open           : BOOL;          // LV Incomer TRAFO Status
SG100D1_A4_TR_200DE1_RealPower      : REAL;          // LV Incomer TRAFO Active Power kW. LV
                                                     // Incomer Transformer. It measure the
                                                     // AuxLoad
SG100D1_A4_TR_200DE1_Alarm              : BOOL;      // LV Incomer TRAFO Alarm for inconsistent
                                                     // status
// MV Incoming Transformer
SG100D1_A6_TR_100AD1_CloseCmd       : BOOL;          // MV Incomer TRAFO Command
SG100D1_A6_TR_100AD1_OpenCmd        : BOOL;          // MV Incomer TRAFO Command
SG100D1_A6_TR_100AD1_ResetCmd        : BOOL;         // MV Incomer TRAFO Command
SG100D1_A6_TR_100AD1_Closed         : BOOL;          // MV Incomer TRAFO Status
SG100D1_A6_TR_100AD1_Open           : BOOL;          // MV Incomer TRAFO Status
SG100D1_A6_TR_100AD1_VoltsMainBusL1_L2    : REAL;
SG100D1_A6_TR_100AD1_RealPower      : REAL;          // MV Incomer TRAFO Active Power kW. MV
                                                     // Incomer Transformer
SG100D1_A6_TR_100AD1_Alarm          : BOOL;          // MV Incomer TRAFO Alarm for inconsistent
                                                     // status

SG100D1_A7_FUELL_CELL_CloseCmd      : BOOL;          // FUELL_CELL Command
SG100D1_A7_FUELL_CELL_OpenCmd       : BOOL;          // FUELL_CELL Command
SG100D1_A7_FUELL_CELL_ResetCmd      : BOOL;          // FUELL_CELL Command
SG100D1_A7_FUELL_CELL_Closed        : BOOL;          // FUELL_CELL Status
SG100D1_A7_FUELL_CELL_Open          : BOOL;          // FUELL_CELL Status
SG100D1_A7_FUELL_CELL_RealPower     : REAL;          // FUELL_CELL Active Power kW
SG100D1_A7_FUELL_CELL_Alarm         : BOOL;          //FUELL_CELL Alarm for inconsistent status

//=====================================================
// SG200N1 Switch Gear
//=====================================================
SG200N1_Connected                       : BOOL;
SG200N1_CloseCmd                        : BOOL;          // Command
SG200N1_OpenCmd                         : BOOL;          // Command
SG200N1_ResetCmd                        : BOOL;          // Command
SG200N1_Closed                          : BOOL;          // Status
SG200N1_Open                            : BOOL;          // Status
SG200N1_RealPower                       : REAL;          // Active Power
```

```
    SG200N1_Alarm                    : BOOL;              //  Alarm  for  inconsistent  status


    //=====================================================
    // GIS Circuit Breaker
    //=====================================================
    HNA6_QA1CB_Connected             : BOOL;
    HNA6_QA1CB_CloseCmd              : BOOL;
    HNA6_QA1CB_OpenCmd               : BOOL;
    HNA6_QA1CB_Status                : DBPOS_T;
    HNA6_QA1CB_StatusInt             : SINT;          // To be used only for test purpose and it
                                                      // is written by the simulator. Remove the
                                                      // assignment in CircuitBreakers PROGRAM
    HNA6_QA1CB_Alarm                 : BOOL;

    HNA6_QA1_ReadyToClose : BOOL;            // This signal comes from the HV-SCADA
    HNA6_QA1CB_OpenCmd_InternalSim: BOOL;    // This signal must be replaced with the protocol
                                             // signal
    HNA6_QA1CB_CloseCmd_InternalSim: BOOL;   // This signal must be replaced with the protocol
                                             // signal

    //=====================================================
    // AVR Transformer
    //=====================================================
    AVR_Connected : BOOL;
    AVR_TapRaise : BOOL;
    AVR_TapLower : BOOL;
    AVR_TapPosition : INT;
    AVR_VoltageSetPoint : REAL;
    AVR_VoltageSetPointResearch : REAL;
    AVR_OutputVoltage : REAL;
    AVR_AutoManual : BOOL;                   // 0=Manual; 1=Automatic
    AVR_Status_OK: BOOL;                     // 0=Operational; 1=Fault
    AVR_VoltageDeviationPerc : REAL;
    AVR_IsResearchMode : BOOL;               // flag indicating if the AVR is working
                                             // in Research mode
END_VAR
```

4.2 MAIN_CODE

The main code is the entry point of the EMCS logic, and it is responsible to evaluate the MODEs preconditions and to orchestrate the transition between one mode and another.

```
PROGRAM MAIN_CODE
VAR
      modeSelector : EMCS_MODE_SELECTOR;

      PCS_PROCESS_FULLY_CHARGED : BOOL;
      HNA5_P : REAL;                                    // WindTurbine Active Power
      HoldTime : TIME;                                  // Parameter
      Auto : BOOL;
      SupplyGuarantee : BOOL;
      BESS_OK : BOOL;
      HNA5_OK : BOOL;
      Can_Stop_EL : BOOL;                               // Can stop electrolyzer

      //TIMER
      precond_TON_01_HNA5_P :BOOL;
      precond_TON_02_HNA5_P :BOOL;
      precond_TON_03_HNA5_P :BOOL;

      TON_01_HNA5_P_GT_GUAR :TON;
      TON_02_HNA5_P_LT_GUAR :TON;
      TON_03_HNA5_P_GT_SURP :TON;

      Power_Out_Exit: F_TRIG;

      SolarFarm_StartupCmd_Reset : AutoResetSignal;
      SolarFarm_ShutdownCmd_Reset : AutoResetSignal;

      MasterToSlaveTransition : F_TRIG;
      SlaveToMasterTransition : R_TRIG;

END_VAR
```

```
// Variable initializarion
HNA5_P := EMCS_Internal.HNA5_ActivePower;
SupplyGuarantee := EMCS_Internal.PlatformSupplyGuarantee;

HoldTime := T#1S * EMCS_Parameters.WINDFARM_POWER_COMPARE_HOLD_TIME;  // Default 60s
Auto := EMCS_Internal.AutoManualMode = EMCS_AUTOMANUAL_ENUM.AUTOMATIC;
BESS_OK := BESS_Healthy AND NOT PCS_Internal.BESSFault AND EMCS_Internal.SG100D1_A3_BESS_Closed;
HNA5_OK := NOT HNA5_Fault AND HV_SCADA_Connected;
Can_Stop_EL := NOT((EMCS_Internal.Electrolyzer_OperatingStatus =
      ELECTROLYZER_OPERATING_STATUS_ENUM.REMOTE_PRODUCTION OR
      EMCS_Internal.Electrolyzer_OperatingStatus =
      ELECTROLYZER_OPERATING_STATUS_ENUM.LOCAL_PRODUCTION )
      AND PCS_Internal.PermToStartEL AND NOT PCS_Internal.ProcessFullyCharged);

EMCS_Internal.Electrolyzer_KeepRunning := NOT Can_Stop_EL;

//Mapping EMCS_INTERNAL to EMCS_IN
EMCS_IN.ON_GRID := EMCS_INTERNAL.GridTied;
EMCS_IN.SUPPLY_GUA := SupplyGuarantee;

//TIMER
///////////////Precond Timer Valutation//////////////////
precond_TON_01_HNA5_P:= HNA5_P >= EMCS_Parameters.WINDFARM_SURPLUS_MIN_POWER; // Default 1 MW
precond_TON_02_HNA5_P:= HNA5_P <  EMCS_Parameters.WINDFARM_SURPLUS_MIN_POWER; // Default 1 MW
precond_TON_03_HNA5_P:= HNA5_P > EMCS_Parameters.WINDFARM_SURPLUS_MIN_POWER_ABLE_TO_CHARGE;
                                                                              // Default 2 MW
// Avvio dei timer
TON_01_HNA5_P_GT_GUAR(IN:=(precond_TON_01_HNA5_P)  ,PT:=HoldTime);
TON_02_HNA5_P_LT_GUAR(IN:=(precond_TON_02_HNA5_P ) ,PT:=HoldTime);
TON_03_HNA5_P_GT_SURP(IN:=(precond_TON_03_HNA5_P ) ,PT:=HoldTime);
```

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

```
//============================================================
// /////////////Precond Mode evalutation//////////////////
//============================================================
EMCS_IN.precond_EXPORT := NOT ESD_1D AND ((Auto AND EMCS_Internal.GridTied AND SupplyGuarantee AND
HNA5_OK AND  TON_02_HNA5_P_LT_GUAR.Q  AND BESS_OK AND BESS_IsSocOverMinimum() AND Can_Stop_EL) AND
NOT PCS_Private.PermRemovedExport);
EMCS_IN.precond_COLD_STANDBY := NOT ESD_1D AND ((Auto AND EMCS_Internal.GridTied AND SupplyGuarantee
AND HNA5_OK AND  TON_02_HNA5_P_LT_GUAR.Q  AND (NOT BESS_OK OR NOT BESS_IsSocOverMinimum())) OR
PCS_Private.PermRemovedHotStanby OR PCS_Private.PermRemovedExport);
EMCS_IN.precond_HOT_STANDBY := NOT ESD_1D AND  Auto AND EMCS_Internal.GridTied AND SupplyGuarantee
AND HNA5_OK AND  TON_01_HNA5_P_GT_GUAR.Q  AND PCS_INTERNAL.ProcessFullyCharged AND
BESS_IsFullyCharged() AND NOT PCS_Private.PermRemovedHotStanby;
EMCS_IN.precond_IMPORT := NOT ESD_1D AND Auto AND EMCS_Internal.GridTied AND SupplyGuarantee AND
HNA5_OK AND (TON_01_HNA5_P_GT_GUAR.Q OR NOT Can_Stop_EL) AND ( NOT PCS_INTERNAL.ProcessFullyCharged
OR NOT BESS_IsFullyCharged() ) ;
EMCS_IN.precond_IMPORT_ABLE_TO_CHARGE := NOT ESD_1D AND EMCS_IN.precond_IMPORT AND
(TON_03_HNA5_P_GT_SURP.Q OR NOT Can_Stop_EL) ;
EMCS_IN.precond_IMPORT_NOT_ABLE_TO_CHARGE := NOT ESD_1D AND EMCS_IN.precond_IMPORT AND NOT
TON_03_HNA5_P_GT_SURP.Q AND Can_Stop_EL;
EMCS_IN.precond_IDLE :=  NOT ESD_1D AND Auto AND EMCS_Internal.GridTied AND NOT HNA5_OK;
EMCS_IN.precond_EMERGENCY := NOT ESD_1D AND Auto AND (NOT EMCS_Internal.GridTied OR NOT
SupplyGuarantee) AND BESS_OK AND BESS_IsSocOverMinimum(); // AND EMCS_Internal.BESS_HasVoltage AND
EMCS_Internal.BUS_6_6kV_HasVoltage;
EMCS_IN.precond_POWER_OUT := (ESD_1D AND (EMCS_Internal.AutoManualMode <>
EMCS_AUTOMANUAL_ENUM.MANUAL_RESEARCH))  OR (Auto AND (NOT EMCS_Internal.GridTied OR NOT
SupplyGuarantee) AND (NOT BESS_OK OR NOT BESS_IsSocOverMinimum() OR NOT
EMCS_Internal.BESS_HasVoltage OR NOT EMCS_Internal.BUS_6_6kV_HasVoltage));
//AND (Current_Mode <> EMCS_MODE_ENUM.RESEARCH)

// MANUAL MODE
EMCS_IN.precond_MAINTENANCE := NOT ESD_1D AND EMCS_Internal.AutoManualMode =
EMCS_AUTOMANUAL_ENUM.MANUAL_MAINTENANCE ;
EMCS_IN.precond_RESEARCH := EMCS_Internal.AutoManualMode = EMCS_AUTOMANUAL_ENUM.MANUAL_RESEARCH;

// Ensure that on the exit of POWER OUT the new becames RESEARCH
Power_Out_Exit(CLK :=EMCS_IN.precond_POWER_OUT);
IF(Power_Out_Exit.Q AND Current_Mode = EMCS_MODE_ENUM.POWER_OUT) THEN
      AutoManualMode := EMCS_AUTOMANUAL_ENUM.MANUAL_RESEARCH;
END_IF

// ModeSelector FB receives in input EMCS_IN and write in the global variable
// EMCS_OUT the result of selection
modeSelector(EMCS_IN:=GVL_MAIN.EMCS_IN);
```

```
// Expose publicily the status machine status
EMCS_StatusMachine.Current_Mode := EMCS_OUT.CURRENT_MODE;
EMCS_StatusMachine.Next_Mode := EMCS_OUT.NEXT_MODE;
EMCS_StatusMachine.Current_SubMode := EMCS_OUT.CURRENT_SUB_MODE;
EMCS_StatusMachine.Next_SubMode := EMCS_OUT.NEXT_SUB_MODE;

 PCS_Internal.CurrentMode := EMCS_OUT.CURRENT_MODE;

 EMCS_Internal.HNA5_P_GT_GUAR.Inertia := TO_INT(TON_01_HNA5_P_GT_GUAR.PT/1000);
 EMCS_Internal.HNA5_P_GT_GUAR.ActivationTime := HNA5_P_GT_GUAR.Inertia - (TO_INT
      (TON_01_HNA5_P_GT_GUAR.ET/1000));
 EMCS_Internal.HNA5_P_GT_GUAR.IsActive := TON_01_HNA5_P_GT_GUAR.Q;

 EMCS_Internal.HNA5_P_LT_GUAR.Inertia := TO_INT(TON_02_HNA5_P_LT_GUAR.PT/1000);
 EMCS_Internal.HNA5_P_LT_GUAR.ActivationTime := HNA5_P_LT_GUAR.Inertia - (TO_INT
      (TON_02_HNA5_P_LT_GUAR.ET/1000));
 EMCS_Internal.HNA5_P_LT_GUAR.IsActive := TON_02_HNA5_P_LT_GUAR.Q;

 EMCS_Internal.HNA5_P_GT_SURP.Inertia := TO_INT(TON_03_HNA5_P_GT_SURP.PT/1000);
 EMCS_Internal.HNA5_P_GT_SURP.ActivationTime := HNA5_P_GT_SURP.Inertia - (TO_INT
      (TON_03_HNA5_P_GT_SURP.ET/1000));
 EMCS_Internal.HNA5_P_GT_SURP.IsActive := TON_03_HNA5_P_GT_SURP.Q;

//=====================================================
// Ensure to Stop BESS
//=====================================================
IF (NOT BESS_OK OR Current_Mode = EMCS_MODE_ENUM.MAINTENANCE) THEN
      Bess_Ensure_Is_Stopped();
 END_IF

  // Ensure to block BESS soc stay within the allowed operating range
 IF ( (BESS_IsFullyCharged() AND EMCS_Internal.BESS_ActivePowerSetPointCmd < 0) OR
      NOT BESS_IsSocOverMinimum() AND EMCS_Internal.BESS_ActivePowerSetPointCmd > 0) THEN
      BESS_SendActPowerSetPoint(0);
 END_IF

//=====================================================
```

```
// Ensure to Stop FUEL CELL
//=====================================================
IF(NOT PCS_Internal.PermToStartFC) THEN
      FuelCell_Ensure_Sleep();
END_IF

IF(Current_Mode <> EMCS_MODE_ENUM.MAINTENANCE) THEN
      IF(NOT EMCS_Internal.FuelCell_CanRun) THEN
            FuelCell_Ensure_Stop();
      END_IF
ELSE
      // In maintenance the Fuel Cell must be stopped and keep stopped (deny start)
      FuelCell_Ensure_Sleep();
END_IF

IF (Current_Mode <> EMCS_MODE_ENUM.RESEARCH OR NOT FuelCell_CanRun)  THEN
      EMCS_Internal.FuelCell_OperatingStatusCmd :=
            EMCS_Internal.FuelCell_OperatingStatusCmd_Private;
END_IF;

//=====================================================
// Ensure to Stop ELECTROLYZER
//=====================================================

 IF (ProcessFullyCharged OR NOT PCS_Internal.PermToStartEL OR Current_Mode =
            EMCS_MODE_ENUM.MAINTENANCE)  THEN
      Electrolyzer_Stop();
 END_IF;

 IF (Current_Mode <> EMCS_MODE_ENUM.RESEARCH OR NOT Electrolyzer_CanRun)  THEN
      // Override the OPC-UA tag with the private value
      EMCS_Internal.Electrolyzer_OperatingStatusCmd :=
            EMCS_Internal.Electrolyzer_OperatingStatusCmd_Private;
 END_IF;
```

```
//=====================================================
// Solar Farm
//=====================================================
IF(   Current_Mode <> EMCS_MODE_ENUM.RESEARCH  AND
      Current_Mode <> EMCS_MODE_ENUM.MAINTENANCE AND
      Current_Mode <> EMCS_MODE_ENUM.EMERGENCY AND
      Current_Mode <> EMCS_MODE_ENUM.POWER_OUT) THEN
      SolarFarm_EnsureStart();
END_IF

// Remove Solar Farm curtailment in all mode except EMERGENCY RESEARCH MAINTENANCE
IF(Current_Mode <> EMCS_MODE_ENUM.EMERGENCY AND
      Current_Mode <> EMCS_MODE_ENUM.RESEARCH  AND Current_Mode <> EMCS_MODE_ENUM.MAINTENANCE AND
            Current_Mode <> EMCS_MODE_ENUM.POWER_OUT) THEN
      // No curtailment is request, so set maximum set poinr
      SolarFarm_SendActivePowerSetPoint(setPoint:= EMCS_Parameters.SOLAR_FARM_MAXIMUM_SETPOINT);
END_IF

// Add logic to open the CB if the Set Point is not followed
// If not healthy stop to send commands
IF(Current_Mode = EMCS_MODE_ENUM.EMERGENCY) THEN
      IF(NOT EMCS_Internal.SolarFarm_Healthy)THEN
            SolarFarm_EnsureStart();
      END_IF
END_IF

IF(NOT SolarFarm_CanRun) THEN
      SolarFarm_EnsureStop();
END_IF

// Autoreset SolaFarm start/shutdown commands
SolarFarm_StartupCmd_Reset(IN:=EMCS_Internal.SolarFarm_StartupCmd);
EMCS_Internal.SolarFarm_StartupCmd  := SolarFarm_StartupCmd_Reset.OUT;
SolarFarm_ShutdownCmd_Reset(IN:=EMCS_Internal.SolarFarm_ShutdownCmd);
EMCS_Internal.SolarFarm_ShutdownCmd  := SolarFarm_ShutdownCmd_Reset.OUT;
```

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

```
//=====================================================
// PCS
//=====================================================
IF(   Current_Mode <> EMCS_MODE_ENUM.EMERGENCY AND
      Current_Mode <> EMCS_MODE_ENUM.POWER_OUT AND
      Current_Mode <> EMCS_MODE_ENUM.RESEARCH) THEN
      PCS_RemoveLoadShedRequest();
END_IF


TotalActivePower := BESS_ActivePower + FuelCell_ActivePower + SolarFarm_ActivePower -
      Electrolyzer_ActivePowerConsumption - AuxLoad;
BessFuellCellActivePower := BESS_ActivePower + FuelCell_ActivePower + 0;

Emergency_Mode_Remaining_Time();

// Start/Stop and send setpoint to Electrolyzer in RESEARCH MODE
IF(Current_Mode = EMCS_MODE_ENUM.RESEARCH)THEN
      IF(EMCS_Internal.Electrolyzer_OperatingStatusCmd =
            ELECTROLYZER_OPERATING_STATUS_CMD_ENUM.MODE_PRODUCTION)THEN// Start
            EMCS_Internal.Electrolyzer_SetPointChoiseCmd :=
                  ELECTROLYZER_SETPOINT_CHOISE_CMD_ENUM.FLOW_H2_SETPOINT;          //Flow
            EMCS_Internal.Electrolyzer_FlowSetPointCmd :=
                  EMCS_Parameters.ELECTROLYZER_MAX_FLOW_SET_POINT;
      ELSIF(EMCS_Internal.Electrolyzer_OperatingStatusCmd =
                  ELECTROLYZER_OPERATING_STATUS_CMD_ENUM.MODE_STOP)THEN      // Stop
            EMCS_Internal.Electrolyzer_SetPointChoiseCmd :=
                  ELECTROLYZER_SETPOINT_CHOISE_CMD_ENUM.FLOW_H2_SETPOINT;          //Flow
                  EMCS_Internal.Electrolyzer_FlowSetPointCmd := 0;
      END_IF
END_IF
```

```
// Start/Stop FUEL CELL in RESEARCH MODE and map setpoint
IF(Current_Mode = EMCS_MODE_ENUM.RESEARCH)THEN
      IF(EMCS_Internal.FuelCell_OperatingStatusCmd = FUEL_CELL_OPERATING_STATUS_CMD_ENUM.START)THEN

            FuelCell_Ensure_Start();
      ELSIF(EMCS_Internal.FuelCell_OperatingStatusCmd =
            FUEL_CELL_OPERATING_STATUS_CMD_ENUM.STOP)THEN   // Stop
                  FuelCell_Ensure_Stop();
      ELSIF(EMCS_Internal.FuelCell_OperatingStatusCmd =
            FUEL_CELL_OPERATING_STATUS_CMD_ENUM.SLEEP)THEN // Stop
                  FuelCell_Ensure_Sleep();
      END_IF

      EMCS_Internal.FuelCell_ActivePowerTarget := EMCS_Internal.FuelCell_ActivePowerSetPointCmd;
END_IF


MasterToSlaveTransition(CLK:=IsMaster);
SlaveToMasterTransition(CLK:=IsMaster);

IF(MasterToSlaveTransition.Q OR SlaveToMasterTransition.Q)THEN
      EMCS_Internal.AutoManualMode := EMCS_AUTOMANUAL_ENUM.MANUAL_RESEARCH;
END_IF
```

4.3 FORMULAS

```
PROGRAM FORMULAS
VAR
      BESS_Reset_Timer: TON;
      FuelCell_Reset_Timer: TON;
      FuelCell_ResetFall : F_TRIG;
      BESS_ResetGridConnectPermissive : R_TRIG;
      MV_IncomerClosed : BOOL;
END_VAR

IF(EMCS_Private.startup) THEN

      EMCS_Internal.NodeId := RTAC_Redundancy_GVL.RTAC_NodeId;
      EMCS_Internal.AutoManualMode := EMCS_AUTOMANUAL_ENUM.MANUAL_RESEARCH;        // Research
      EMCS_Internal.OperatorSupplyGuarantee := EMCS_SUPPLY_GUARANTEE.AUTOMATIC;
                                        //WindFarm.HNA5-AFA78.StatusCode==0 AND MCS.SupplyGuarantee

      EMCS_Internal.BUS_6_6kV_HasVoltage := TRUE;

      //=================================================
      // Wind Farm
      //=================================================
      EMCS_Internal.HV_SCADA_Connected := TRUE;
      EMCS_Internal.HNA5_SupplyGuarantee := TRUE ;
      EMCS_Internal.HNA5_StatusCode := 0;
      EMCS_Internal.HNA5_QA1CB_Status := CB_STATUS_ENUM.Closed;
      //=================================================
      // BESS
      //=================================================
      //EMCS_INTERNAL.BESS_FullyCharged :=0;
                  //BESS.Status3.SOC + BESS-MAXIMUM-OPERATING-RESERVE >= 100% OR BESS.Status3.SOC >
                  // BESS-MINIMUM-CHARGE-START-RESERVE
      EMCS_Internal.BESS_Connected := TRUE;
      EMCS_Internal.BESS_OperatingStatus := BESS_OPERATING_STATUS_ENUM.STOP;
      EMCS_Internal.BESS_DischargeEnergy := 0.5 * EMCS_Parameters.BESS_ENERGY_CAPACITY;
      EMCS_Internal.BESS_HasVoltage := 1;  // SG-100D1 voltage measure on the BLPH 6.6kV
      EMCS_Internal.BESS_Healthy := TRUE;  // The real value is mapped in the program
                                          // PackagesSignalMap
```

```
//================================================
// Fuel Cell
//================================================
EMCS_Internal.FuelCell_ConversionEfficiency :=
      EMCS_Parameters.FUEL_CELL_CONVERSION_EFFICIENCY;
EMCS_Internal.FuelCell_Connected := TRUE;
EMCS_Internal.FuelCell_OperatingStatusCmd_Private :=
      FUEL_CELL_OPERATING_STATUS_CMD_ENUM.SLEEP;
EMCS_Internal.FuelCell_OperatingStatus := FUEL_CELL_OPERATING_STATUS_ENUM.Sleep;
EMCS_Internal.FuelCell_ActivePowerSetPointCmd := EMCS_Parameters.FUEL_CELL_MINIMUM_SETPOINT;
EMCS_Internal.FuelCell_Efficiency := 0.5; // This variable must be mappe to the FuelCell
                                          // Modbus signal: System Efficiency
                                          // (Input Register 1202)

//================================================
// Electrolyzer
//================================================
EMCS_Internal.Electrolyzer_Connected := TRUE;
EMCS_Internal.Electrolyzer_OperatingStatusCmd_Private :=
      ELECTROLYZER_OPERATING_STATUS_CMD_ENUM.MODE_STOP;
EMCS_Internal.Electrolyzer_OperatingStatus := ELECTROLYZER_OPERATING_STATUS_ENUM.REMOTE_STOP;
EMCS_Internal.Electrolyzer_ActivePowerSetPointCmd := 0;
EMCS_Internal.Electrolyzer_FlowSetPointCmd := 0;

//================================================
// Solar Farm
//================================================
EMCS_Internal.SolarFarm_Connected := TRUE;
EMCS_Internal.SolarFarm_Healthy :=1;
EMCS_Internal.SolarFarm_OperatingStatus := SOLAR_FARM_OPERATING_STATUS_ENUM.OPERATING;
SolarFarm_SendActivePowerSetPoint(setPoint:= 0);
```

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

```
//=================================================
// SG200D2
//=================================================
SG200D2_Connected := TRUE;
SG200D2_Closed := 1; // CB of solarfarm. Initialized as CLOSED
SG200D2_Open := 0;       // CB of solarfarm

//=================================================
// SG100D1
//=================================================

SG100D1_Connected := TRUE;

SG100D1_A1_ELECTROLYZER_Closed     := 1;
SG100D1_A1_ELECTROLYZER_Open       := 0;
SG100D1_A1_ELECTROLYZER_RealPower  := 1;

SG100D1_A2_TR_200ED1_Closed        := 1;       // Solar Farm
SG100D1_A2_TR_200ED1_Open          := 0;       // Solar Farm
SG100D1_A2_TR_200ED1_RealPower     := 1;

SG100D1_A3_BESS_Closed             := 1;
SG100D1_A3_BESS_Open               := 0;
SG100D1_A3_BESS_RealPower          := 1;

SG100D1_A4_TR_200DE1_Closed        := 1;       // Auxiliaries
SG100D1_A4_TR_200DE1_Open          := 0;       // Auxiliaries

SG100D1_A6_TR_100AD1_Closed        := 1;       // Incomer
SG100D1_A6_TR_100AD1_Open          := 0;       // Incomer
SG100D1_A6_TR_100AD1_VoltsMainBusL1_L2:= 6.6;

SG100D1_A7_FUELL_CELL_Closed       := 1;
SG100D1_A7_FUELL_CELL_Open         := 0;
SG100D1_A7_FUELL_CELL_RealPower    := 1;
```

```
        //================================================
        // SG100D1
        //================================================
        SG200N1_Connected := TRUE;

        SG200N1_Closed          := 1;
        SG200N1_Open            := 0;
        SG200N1_RealPower       := 1;

        // ========================================
        // GIS
        // ========================================
        HNA6_QA1CB_StatusInt := DBPOS_T.dbpos_on;
        HNA6_QA1CB_Connected := TRUE;

        EMCS_Internal.AuxLoad := 100;            //kW

        //========================================
        // PCS Variables Initialization
        //========================================
        //PCS_Internal.HydrogenLevel := 100;     //Kg

        PCS_Internal.PermToStartFC := TRUE;
        PCS_Internal.PermToStartEL := TRUE;
        PCS_Internal.ProcessEmpty := FALSE;
        PCS_Internal.ProcessFullyCharged := FALSE;
        PCS_Internal.ESD_1D := FALSE;
        PCS_Internal.BESSFault := FALSE;
        PCS_Internal.HydrogenLevel := 500 + 0;

END_IF
//=============================================================================================
// End of initialization
//=============================================================================================
```

```
//==================================
// Supply Guarantee
//==================================
EMCS_Internal.PlatformSupplyGuarantee := (EMCS_INTERNAL.OperatorSupplyGuarantee =
      EMCS_SUPPLY_GUARANTEE.AUTOMATIC  AND HNA5_SupplyGuarantee ) OR
      EMCS_INTERNAL.OperatorSupplyGuarantee = EMCS_SUPPLY_GUARANTEE.MANUAL_ACTIVE;

// ========================================
// BESS
// ========================================
EMCS_Internal.BESS_FullyCharged := BESS_IsFullyCharged();
EMCS_Internal.BESS_AvailableEnergyPerc := (EMCS_Internal.BESS_DischargeEnergy /
EMCS_Parameters.BESS_ENERGY_CAPACITY) * 100;
EMCS_Internal.BESS_ChargeEnergy :=  EMCS_Parameters.BESS_ENERGY_CAPACITY -
EMCS_Internal.BESS_DischargeEnergy;
EMCS_Internal.BESS_OperatingStatus := BESS_CalculateOperatingStatus();
EMCS_Internal.BESS_GridFormingPermissive :=
      (BESS_Healthy AND NOT PCS_Internal.BESSFault AND
      EMCS_Internal.SG100D1_A3_BESS_Closed) AND // BESS_HasVoltage AND
      EMCS_Internal.BESS_AvailableEnergyPerc > BESS_MINIMUM_OPERATING_RESERVE AND
      NOT EMCS_Internal.BESS_GridConnectPermissive;


// Reset the BESS_GridConnectPermissive when the MV INCOMER Breaker is closed.
MV_IncomerClosed := EMCS_Internal.SG100D1_A6_TR_100AD1_Closed AND NOT
EMCS_Internal.SG100D1_A6_TR_100AD1_Open;

BESS_ResetGridConnectPermissive(CLK := MV_IncomerClosed);
IF(BESS_ResetGridConnectPermissive.Q OR MV_IncomerClosed)THEN
      EMCS_Internal.BESS_GridConnectPermissive := FALSE;
END_IF

EMCS_Internal.GridVoltageOK := EMCS_Internal.SG100D1_A6_TR_100AD1_VoltsMainBusL1_L2 > 6.0;

IF (EMCS_Internal.BESS_ResetCmd) THEN // Reset command is sent
      BESS_Reset_Timer(IN:=TRUE, PT:=T#3S);
      IF(BESS_Reset_Timer.Q = 1)THEN
            EMCS_Internal.BESS_ResetCmd := FALSE;
            BESS_Reset_Timer(IN:=FALSE, PT:=T#3S);
      END_IF
```

```
END_IF
// ========================================
// GIS / GRID TIED
// ========================================
EMCS_Internal.GridTied := (HNA6_QA1CB_Status = dbpos_on) AND                    // GIS
      (SG100D1_A6_TR_100AD1_Closed AND NOT SG100D1_A6_TR_100AD1_Open);          // SG-100D1 MV
INCOMER

// Check if the CB of the wind farm must be added (consider connection status and tag quality)
EMCS_Internal.BESS_GridVoltageOK := EMCS_Internal.GridVoltageOK;

// ========================================
// Electrolyzer
// ========================================
EMCS_Internal.Electrolyzer_ActivePowerConsumption :=
EMCS_Internal.SG100D1_A1_ELECTROLYZER_RealPower;

EMCS_Internal.Electrolyzer_CanRun :=
      NOT PCS_Internal.ProcessFullyCharged AND
      PCS_Internal.PermToStartEL AND
      NOT Electrolyzer_IsReconnecting AND
      EMCS_Internal.SG100D1_A1_ELECTROLYZER_Closed;

IF (EMCS_StatusMachine.Current_Mode = EMCS_MODE_ENUM.RESEARCH) THEN
      IF( EMCS_Internal.Electrolyzer_CanRun) THEN
            EMCS_Internal.Electrolyzer_OperatingStatusCmd_Private :=
                  EMCS_Internal.Electrolyzer_OperatingStatusCmd;
      ELSE
            EMCS_Internal.Electrolyzer_OperatingStatusCmd_Private :=
                  ELECTROLYZER_OPERATING_STATUS_CMD_ENUM.MODE_STOP;
      END_IF
END_IF
```

```
// ========================================
// FuelCell
// ========================================
FuelCell_CanRun :=
      PCS_Internal.PermToStartFC = TRUE AND
      NOT PCS_Internal.ProcessEmpty AND
      EMCS_Internal.SG100D1_A7_FUELL_CELL_Closed AND
      NOT FuelCell_IsReconnecting;

IF (EMCS_StatusMachine.Current_Mode = EMCS_MODE_ENUM.RESEARCH) THEN
      IF( EMCS_Internal.FuelCell_CanRun) THEN
            EMCS_Internal.FuelCell_OperatingStatusCmd_Private :=
                  EMCS_Internal.FuelCell_OperatingStatusCmd;
      END_IF
END_IF

// Autoreset of the FUEL_CELL ResetCmd
FuelCell_ResetFall(CLK:= EMCS_Internal.FuelCell_ResetCmd);
IF( EMCS_Internal.FuelCell_ResetCmd) THEN
      EMCS_Internal.FuelCell_OperatingStatusCmd_Private :=
            FUEL_CELL_OPERATING_STATUS_CMD_ENUM.RESET;
END_IF
IF(FuelCell_ResetFall.Q)THEN
      EMCS_Internal.FuelCell_OperatingStatusCmd_Private :=
            FUEL_CELL_OPERATING_STATUS_CMD_ENUM.FC_NONE;
END_IF;

IF (EMCS_Internal.FuelCell_ResetCmd) THEN // Reset command is sent
      FuelCell_Reset_Timer(IN:=TRUE, PT:=T#3S);
      IF(FuelCell_Reset_Timer.Q)THEN
            EMCS_Internal.FuelCell_ResetCmd := FALSE;
            FuelCell_Reset_Timer(IN:=FALSE, PT:=T#3S);
      END_IF
END_IF
```

```
// ========================================
// SolarFarm
// ========================================
SolarFarm_CanRun := FALSE;    // OVERRIDE: EMCS_Internal.SG100D1_A2_TR_200ED1_Closed;

IF (EMCS_StatusMachine.Current_Mode = EMCS_MODE_ENUM.RESEARCH) THEN
      IF(EMCS_Internal.SolarFarm_CanRun) THEN
            EMCS_Internal.SolarFarm_StartupCmd_Private := EMCS_Internal.SolarFarm_StartupCmd;
      END_IF
END_IF

// ========================================
// WindFarm
// ========================================
EMCS_Internal.HNA5_Fault :=
      HNA5_StatusCode = HNA5_STATUS_CODE_ENUM.Stopped_GridError OR
      HNA5_StatusCode = HNA5_STATUS_CODE_ENUM.Stopped_ErrorWithAutoReset OR
      HNA5_StatusCode = HNA5_STATUS_CODE_ENUM.Stopped_ManualAlarmResetRequired OR
      HNA5_StatusCode = HNA5_STATUS_CODE_ENUM.SafetyStop OR
      HNA5_StatusCode = HNA5_STATUS_CODE_ENUM.RemoteStop OR
      HNA5_QA1CB_Status = CB_STATUS_ENUM.Opened;

EMCS_Internal.AuxLoad := SG100D1_A4_TR_200DE1_RealPower;
EMCS_Internal.IsMaster := RTAC_Redundancy_GVL.Master;

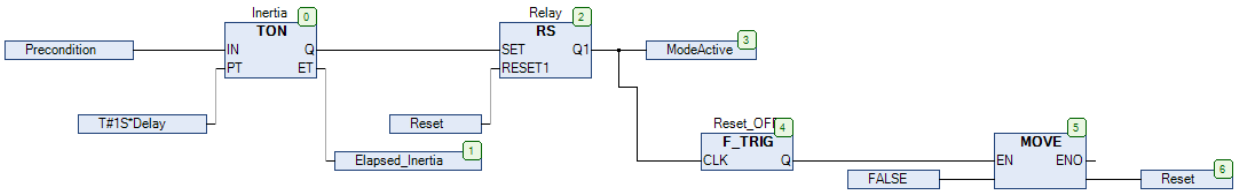
FuelCell_ActivePowerChangeRate := EMCS_Parameters.FUEL_CELL_ACTIVE_POWER_CHANGE_RATE;
```

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

4.4 EMCS_MODE_ACTIVATOR

```
FUNCTION_BLOCK EMCS_MODE_ACTIVATOR
VAR
      Inertia: TON;
      Reset_OFF: F_TRIG;
      Relay: RS;

END_VAR
VAR_INPUT
      Precondition: BOOL;
      Reset : BOOL;
      Delay :INT :=30;
END_VAR
VAR_OUTPUT
      ModeActive : BOOL;
      Elapsed_Inertia : TIME;
END_VAR
```



4.5 EMCS_MODE_SELECTOR

```
FUNCTION_BLOCK EMCS_MODE_SELECTOR
VAR

      CurrentMode: INT;
      NextMode :INT;
      CurrentPrecond : ARRAY[0..11] OF BOOL;
      PreviousPrecond : ARRAY[0..11] OF BOOL;
      CURRENT_MODE_TO_STOP :INT;
      MAINTENANCE_RUN_REQ :R_TRIG;
      RESEARCH_RUN_REQ :R_TRIG;
```

```
IMPORT_RUN_REQ :R_TRIG;
IMPORT_ABLE_RUN_REQ :R_TRIG;
IMPORT_NOT_ABLE_RUN_REQ :R_TRIG;
NONE_RUN_REQ :R_TRIG;
IDLE_RUN_REQ :R_TRIG;
HOT_STANDBY_RUN_REQ :R_TRIG;
EXPORT_RUN_REQ :R_TRIG;
COLD_STANDBY_RUN_REQ :R_TRIG;
EMERGENCY_RUN_REQ :R_TRIG;
POWEROUT_RUN_REQ :R_TRIG;

MAINTENANCE_STOP_REQ :R_TRIG;
RESEARCH_STOP_REQ :R_TRIG;
IMPORT_STOP_REQ :R_TRIG;
IMPORT_ABLE_STOP_REQ :R_TRIG;
IMPORT_NOT_ABLE_STOP_REQ :R_TRIG;
NONE_STOP_REQ :R_TRIG;
IDLE_STOP_REQ :R_TRIG;
HOT_STANDBY_STOP_REQ :R_TRIG;
EXPORT_STOP_REQ :R_TRIG;
COLD_STANDBY_STOP_REQ :R_TRIG;
EMERGENCY_STOP_REQ :R_TRIG;
POWEROUT_STOP_REQ :R_TRIG;

MAINTENANCE_IDLE_REQ :R_TRIG;
RESEARCH_IDLE_REQ :R_TRIG;
IMPORT_IDLE_REQ :R_TRIG;
IMPORT_ABLE_IDLE_REQ :R_TRIG;
IMPORT_NOT_ABLE_IDLE_REQ :R_TRIG;
NONE_IDLE_REQ :R_TRIG;
IDLE_IDLE_REQ :R_TRIG;
HOT_STANDBY_IDLE_REQ :R_TRIG;
EXPORT_IDLE_REQ :R_TRIG;
COLD_STANDBY_IDLE_REQ :R_TRIG;
EMERGENCY_IDLE_REQ :R_TRIG;
POWEROUT_IDLE_REQ :R_TRIG;

MAN_ON :R_TRIG;
```

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

```
      WaitForExportPermissiveTimer : TON;
      WaitForHotStandbyPermissiveTimer : TON;

END_VAR
VAR_INPUT
      EMCS_IN : EMCS_MODE_PROCESSOR_IN;
END_VAR

VAR_OUTPUT

END_VAR


IF(EMCS_AUX_GVL.startup) THEN

PreviousPrecond[0]:=1;

END_IF;

CurrentPrecond[0] := EMCS_IN.precond_NONE;
CurrentPrecond[1] := EMCS_IN.precond_IDLE;
CurrentPrecond[3] := EMCS_IN.precond_IMPORT_ABLE_TO_CHARGE;
CurrentPrecond[4] := EMCS_IN.precond_IMPORT_NOT_ABLE_TO_CHARGE;
CurrentPrecond[5] := EMCS_IN.precond_HOT_STANDBY;
CurrentPrecond[6] := EMCS_IN.precond_EXPORT;
CurrentPrecond[7] := EMCS_IN.precond_COLD_STANDBY;
CurrentPrecond[8] := EMCS_IN.precond_EMERGENCY;
CurrentPrecond[9] := EMCS_IN.precond_POWER_OUT;
CurrentPrecond[10] := EMCS_IN.precond_MAINTENANCE;
CurrentPrecond[11] := EMCS_IN.precond_RESEARCH;

ModeActivatorNone(Precondition:= EMCS_IN.precond_NONE);
ModeActivatorIdle(Precondition:= EMCS_IN.precond_IDLE);
ModeActivatorImportAbleToCharge(Precondition:= EMCS_IN.precond_IMPORT_ABLE_TO_CHARGE);
ModeActivatorImportNotAbleToCharge(Precondition:= EMCS_IN.precond_IMPORT_NOT_ABLE_TO_CHARGE);
ModeActivatorHotStandBy(Precondition:= EMCS_IN.precond_HOT_STANDBY);
ModeActivatorExport(Precondition:= EMCS_IN.precond_EXPORT);
ModeActivatorColdStandBy(Precondition:= EMCS_IN.precond_COLD_STANDBY);
ModeActivatorEmergency(Precondition:= EMCS_IN.precond_EMERGENCY);
```

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy

```
ModeActivatorPowerOut(Precondition:= EMCS_IN.precond_POWER_OUT);
ModeActivatorMaintenance(Precondition:= EMCS_IN.precond_MAINTENANCE);
ModeActivatorResearch(Precondition:= EMCS_IN.precond_RESEARCH);

ModeActivatorNone.Delay := None_Mode_Delay ;
ModeActivatorPowerout.Delay := Powerout_Mode_Delay;
ModeActivatorEmergency.Delay := Emergency_Mode_Delay ;
ModeActivatorIdle.Delay :=    Idle_Mode_Delay;
ModeActivatorColdStandby.Delay :=   ColdStandBy_Mode_Delay  ;
ModeActivatorHotStandBy.Delay :=    HotStandBy_Mode_Delay ;
ModeActivatorImportAbleToCharge.Delay :=Import_Able_Mode_Delay ;
ModeActivatorImportNotAbleToCharge.Delay :=Import_Not_Able_Mode_Delay ;
ModeActivatorImport.Delay :=Import_Mode_Delay ;
ModeActivatorExport.Delay :=Export_Mode_Delay ;
ModeActivatorResearch.Delay :=Research_Mode_Delay  ;
ModeActivatorMaintenance.Delay :=Maintenance_Mode_Delay  ;

// Call to the executor
////////////////////////////////////////////////////////////////////////////////////////////////////
//////////
// MAINTENANCE
////////////////////////////////////////////////////////////////////////////////////////////////////
//////////
IF(ModeActivatorMaintenance.ModeActive AND NOT (EMCS_OUT.CURRENT_MODE = MAINTENANCE)) THEN

            EMCS_OUT.NEXT_MODE:=EMCS_MODE_ENUM.MAINTENANCE;
            EMCS_OUT.NEXT_SUB_MODE:=INIT;
            ModeExecutorMaintenance(INIT_REQUEST:=TRUE);

      IF (ModeExecutorMaintenance.Init_State AND NOT GVL_MAIN.FirstScanPLC) THEN

            CURRENT_MODE_TO_STOP:=EMCS_OUT.CURRENT_MODE;
          Stop_SubMode(CURRENT_MODE_TO_STOP);

            IF(CheckSubState(CURRENT_MODE_TO_STOP))THEN
                  ModeExecutorMaintenance(RUN_REQUEST:=TRUE);
            END_IF;

      END_IF;
```

```
      //Stop the MODE blocked in init if the preconditions change
      IF (ModeExecutorMaintenance.Init_State=FALSE  AND ModeActivatorMaintenance.Precondition=FALSE)
      THEN
                  Stop_SubMode(MAINTENANCE);
      END_IF


      IF (ModeExecutorMaintenance.Run_State) THEN
            EMCS_OUT.CURRENT_MODE:= GVL_MAIN.EMCS_OUT.CURRENT_MODE;
            EMCS_OUT.NEXT_SUB_MODE:=RUNNING;
            EMCS_OUT.CURRENT_SUB_MODE:=RUNNING;
      END_IF


END_IF;
IF (ModeExecutorMaintenance.Run_State) THEN
ModeExecutorMaintenance(RUN_REQUEST:=TRUE); //test
END_IF

////////////////////////////////////////////////////////////////////////////////////////////////////
//////////
// RESEARCH
////////////////////////////////////////////////////////////////////////////////////////////////////
//////////
IF(ModeActivatorResearch.ModeActive AND NOT (EMCS_OUT.CURRENT_MODE = RESEARCH)) THEN
            EMCS_OUT.NEXT_MODE:=EMCS_MODE_ENUM.RESEARCH;
            EMCS_OUT.NEXT_SUB_MODE:=INIT;
            ModeExecutorResearch(INIT_REQUEST:=TRUE);
      IF (ModeExecutorResearch.Init_State  AND NOT GVL_MAIN.FirstScanPLC) THEN
            CURRENT_MODE_TO_STOP:=EMCS_OUT.CURRENT_MODE;
            Stop_SubMode(CURRENT_MODE_TO_STOP);
      IF(CheckSubState(CURRENT_MODE_TO_STOP))THEN
            ModeExecutorResearch(RUN_REQUEST:=TRUE);
      END_IF;
      END_IF


      //Stop the MODE blocked in init if the preconditions change
```

```
      IF (ModeExecutorResearch.Init_State=FALSE  AND ModeActivatorResearch.Precondition=FALSE) THEN

                  Stop_SubMode(RESEARCH);
      END_IF


      IF (ModeExecutorResearch.Run_State) THEN
            EMCS_OUT.CURRENT_MODE:=GVL_MAIN.EMCS_OUT.CURRENT_MODE;
            EMCS_OUT.NEXT_SUB_MODE:=RUNNING;
            EMCS_OUT.CURRENT_SUB_MODE:=RUNNING;
      END_IF

END_IF;
IF (ModeExecutorResearch.Run_State) THEN
ModeExecutorResearch(RUN_REQUEST:=TRUE); //test
END_IF

//////////////////////////////////////////////////////////////////////////////////////////////////
//////////
// NONE
//////////////////////////////////////////////////////////////////////////////////////////////////
//////////
IF((ModeActivatorNone.ModeActive AND NOT (EMCS_OUT.CURRENT_MODE = NONE)) OR GVL_MAIN. FirstScanPLC)
THEN


            EMCS_OUT.NEXT_MODE:=EMCS_MODE_ENUM.NONE;
            EMCS_OUT.NEXT_SUB_MODE:=INIT;
            ModeExecutorNone(INIT_REQUEST:=TRUE);


      IF (ModeExecutorNone.Init_State AND NOT GVL_MAIN. FirstScanPLC) THEN
            CURRENT_MODE_TO_STOP:=EMCS_OUT.CURRENT_MODE;
            Stop_SubMode(CURRENT_MODE_TO_STOP);
      IF(CheckSubState(CURRENT_MODE_TO_STOP))THEN
      ModeExecutorNone(RUN_REQUEST:=TRUE);
      END_IF;
      END_IF

      //Stop the MODE blocked in init if the preconditions change
```

```
      IF (ModeExecutorNone.Init_State=FALSE  AND ModeActivatorNone.Precondition=FALSE) THEN

                  Stop_SubMode(NONE);
      END_IF

      IF (ModeExecutorNone.Run_State) THEN
            EMCS_OUT.CURRENT_MODE:= GVL_MAIN.EMCS_OUT.CURRENT_MODE;
            EMCS_OUT.NEXT_SUB_MODE:=RUNNING;
            EMCS_OUT.CURRENT_SUB_MODE:=RUNNING;
      END_IF



END_IF;

(*
IF(ModeActivatorImport.ModeActive AND NOT (EMCS_OUT.CURRENT_MODE = IMPORT)) THEN


            EMCS_OUT.NEXT_MODE:=EMCS_MODE_ENUM.IMPORT;
            EMCS_OUT.NEXT_SUB_MODE:=INIT;
            ModeExecutorImport(INIT_REQUEST:=TRUE);


      IF (ModeExecutorImport.Init_State  AND NOT GVL_MAIN. FirstScanPLC) THEN
            CURRENT_MODE_TO_STOP:=EMCS_OUT.CURRENT_MODE;
            Stop_SubMode(CURRENT_MODE_TO_STOP);
      IF(CheckSubState(CURRENT_MODE_TO_STOP))THEN
      ModeExecutorImport(RUN_REQUEST:=TRUE);
      END_IF;
      END_IF



      IF (ModeExecutorImport.Run_State) THEN
            EMCS_OUT.CURRENT_MODE:= GVL_MAIN.EMCS_OUT.CURRENT_MODE;
            EMCS_OUT.NEXT_SUB_MODE:=RUNNING;
            EMCS_OUT.CURRENT_SUB_MODE:=RUNNING;
      END_IF
```

```
END_IF;*)


////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////
// Import Not Able Charge
////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////
IF(ModeActivatorImportAbleToCharge.ModeActive AND NOT (EMCS_OUT.CURRENT_MODE =
IMPORT_ABLE_TO_CHARGE)) THEN



            EMCS_OUT.NEXT_MODE:=EMCS_MODE_ENUM.IMPORT_ABLE_TO_CHARGE;
            EMCS_OUT.NEXT_SUB_MODE:=INIT;
            ModeExecutorImportAbleToCharge(INIT_REQUEST:=TRUE);



      IF (ModeExecutorImportAbleToCharge.Init_State   AND NOT GVL_MAIN.FirstScanPLC) THEN

            CURRENT_MODE_TO_STOP:=EMCS_OUT.CURRENT_MODE;
            Stop_SubMode(CURRENT_MODE_TO_STOP);
      IF(CheckSubState(CURRENT_MODE_TO_STOP))THEN
      ModeExecutorImportAbleToCharge(RUN_REQUEST:=TRUE);
      END_IF;
      END_IF


//Stop the MODE blocked in init if the preconditions change
      IF (ModeExecutorImportAbleToCharge.Init_State=FALSE  AND
ModeActivatorImportAbleToCharge.Precondition=FALSE) THEN
                  Stop_SubMode(IMPORT_ABLE_TO_CHARGE);
      END_IF


      IF (ModeExecutorImportAbleToCharge.Run_State) THEN
            EMCS_OUT.CURRENT_MODE:= GVL_MAIN.EMCS_OUT.CURRENT_MODE;
            EMCS_OUT.NEXT_SUB_MODE:=RUNNING;
            EMCS_OUT.CURRENT_SUB_MODE:=RUNNING;
      END_IF


END_IF;
IF (ModeExecutorImportAbleToCharge.Run_State) THEN
```

```
ModeExecutorImportAbleToCharge(RUN_REQUEST:=TRUE); //test
END_IF

////////////////////////////////////////////////////////////////////////////////////////////////////
// Import Not Able To Charge
////////////////////////////////////////////////////////////////////////////////////////////////////
IF(ModeActivatorImportNotAbleToCharge.ModeActive AND NOT (EMCS_OUT.CURRENT_MODE =
IMPORT_NOT_ABLE_TO_CHARGE)) THEN


            EMCS_OUT.NEXT_MODE:=EMCS_MODE_ENUM.IMPORT_NOT_ABLE_TO_CHARGE;
            EMCS_OUT.NEXT_SUB_MODE:=INIT;
            ModeExecutorImportNotAbleToCharge(INIT_REQUEST:=TRUE);


      IF (ModeExecutorImportNotAbleToCharge.Init_State   AND NOT GVL_MAIN.FirstScanPLC) THEN

            CURRENT_MODE_TO_STOP:=EMCS_OUT.CURRENT_MODE;
            Stop_SubMode(CURRENT_MODE_TO_STOP);
      IF(CheckSubState(CURRENT_MODE_TO_STOP))THEN
      ModeExecutorImportNotAbleToCharge(RUN_REQUEST:=TRUE);
      END_IF;
      END_IF


//Stop the MODE blocked in init if the preconditions change
      IF (ModeExecutorImportNotAbleToCharge.Init_State=FALSE  AND
ModeActivatorImportNotAbleToCharge.Precondition=FALSE) THEN
                  Stop_SubMode(IMPORT_NOT_ABLE_TO_CHARGE);
      END_IF


      IF (ModeExecutorImportNotAbleToCharge.Run_State) THEN
            EMCS_OUT.CURRENT_MODE:= GVL_MAIN.EMCS_OUT.CURRENT_MODE;
            EMCS_OUT.NEXT_SUB_MODE:=RUNNING;
            EMCS_OUT.CURRENT_SUB_MODE:=RUNNING;
      END_IF


END_IF;
IF (ModeExecutorImportNotAbleToCharge.Run_State) THEN
```

```
ModeExecutorImportNotAbleToCharge(RUN_REQUEST:=TRUE); //test
END_IF

////////////////////////////////////////////////////////////////////////////////////////////////////
// HOT STANDBY
////////////////////////////////////////////////////////////////////////////////////////////////////
IF(ModeActivatorHotStandBy.ModeActive AND NOT (EMCS_OUT.CURRENT_MODE = HOT_STANDBY)) THEN


            EMCS_OUT.NEXT_MODE:=EMCS_MODE_ENUM.HOT_STANDBY;
            EMCS_OUT.NEXT_SUB_MODE:=INIT;
            ModeExecutorHotStandBy(INIT_REQUEST:=TRUE);


      IF (ModeExecutorHotStandBy.Init_State   AND NOT GVL_MAIN.FirstScanPLC) THEN
            CURRENT_MODE_TO_STOP:=EMCS_OUT.CURRENT_MODE;
            Stop_SubMode(CURRENT_MODE_TO_STOP);
      IF(CheckSubState(CURRENT_MODE_TO_STOP))THEN
      ModeExecutorHotStandby(RUN_REQUEST:=TRUE);
      END_IF;
      END_IF

      //Stop the MODE blocked in init if the preconditions change
      IF (ModeExecutorHotStandBy.Init_State=FALSE  AND ModeActivatorHotStandBy.Precondition=FALSE)
THEN
                  Stop_SubMode(HOT_STANDBY);
      END_IF

      IF (ModeExecutorHotStandBy.Run_State) THEN
            EMCS_OUT.CURRENT_MODE:= GVL_MAIN.EMCS_OUT.CURRENT_MODE;
            EMCS_OUT.NEXT_SUB_MODE:=RUNNING;
            EMCS_OUT.CURRENT_SUB_MODE:=RUNNING;
      END_IF

END_IF;

// Remove the permissive to stay in HOT STANDBY due to change in preconditions or to PCS permission
remove
```

```
IF(ModeExecutorHotStandBy.Run_State AND (NOT PCS_Private.PermEnter_HotStandBy OR NOT BESS_Healthy OR
PCS_Internal.BESSFault)) THEN
      PCS_Private.PermRemovedHotStanby := TRUE;
END_IF

// Remove the permissive to stay in HOT STANDBY due to timeout in PCS permission receiving
WaitForHotStandbyPermissiveTimer(IN:=ModeExecutorHotStandBy.Init_WaitForPermissive,
PT:=PCS_PermissiveModeTimeout);

// Remove the permissive to stay in EXPORT due to timeout in PCS permission receiving
IF(WaitForHotStandbyPermissiveTimer.Q)THEN
      PCS_Private.PermRemovedHotStanby := TRUE;
END_IF

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////
// EXPORT
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////
IF(ModeActivatorExport.ModeActive AND NOT (EMCS_OUT.CURRENT_MODE = EXPORT)) THEN


            EMCS_OUT.NEXT_MODE:=EMCS_MODE_ENUM.EXPORT;
            EMCS_OUT.NEXT_SUB_MODE:=INIT;
            ModeExecutorExport(INIT_REQUEST:=TRUE);


      IF (ModeExecutorExport.Init_State   AND NOT GVL_MAIN.FirstScanPLC) THEN
            CURRENT_MODE_TO_STOP:=EMCS_OUT.CURRENT_MODE;
            Stop_SubMode(CURRENT_MODE_TO_STOP);
      IF(CheckSubState(CURRENT_MODE_TO_STOP))THEN
            ModeExecutorExport(RUN_REQUEST:=TRUE);
      END_IF;
      END_IF

      //Stop the MODE blocked in init if the preconditions change
      IF (ModeExecutorExport.Init_State=FALSE  AND ModeActivatorExport.Precondition=FALSE) THEN

            Stop_SubMode(EXPORT);
      END_IF
```

```
      IF (ModeExecutorExport.Run_State) THEN
            EMCS_OUT.CURRENT_MODE:= GVL_MAIN.EMCS_OUT.CURRENT_MODE;
            EMCS_OUT.NEXT_SUB_MODE:=RUNNING;
            EMCS_OUT.CURRENT_SUB_MODE:=RUNNING;

      END_IF

END_IF;

IF (ModeExecutorExport.Run_State) THEN
      ModeExecutorExport(RUN_REQUEST:=TRUE); //test
END_IF

// Remove the permissive to stay in HOT STANDBY due to PCS permission remove
IF (ModeExecutorExport.Run_State AND NOT PCS_Private.PermEnter_Export) THEN
      PCS_Private.PermRemovedExport := TRUE;
END_IF

WaitForExportPermissiveTimer(IN:=ModeExecutorExport.Init_WaitForPermissive,
PT:=PCS_PermissiveModeTimeout);

// Remove the permissive to stay in EXPORT due to timeout in PCS permission receiving
IF(WaitForExportPermissiveTimer.Q)THEN
      PCS_Private.PermRemovedExport := TRUE;
END_IF

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////
// COLD STANDBY
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////
IF(ModeActivatorCOLDStandBy.ModeActive AND NOT (EMCS_OUT.CURRENT_MODE = COLD_STANDBY)) THEN

            EMCS_OUT.NEXT_MODE:=EMCS_MODE_ENUM.COLD_STANDBY;
            EMCS_OUT.NEXT_SUB_MODE:=INIT;
            ModeExecutorCOLDStandBy(INIT_REQUEST:=TRUE);

      IF (ModeExecutorCOLDStandBy.Init_State   AND NOT GVL_MAIN.FirstScanPLC) THEN
            CURRENT_MODE_TO_STOP:=EMCS_OUT.CURRENT_MODE;
```

```
            Stop_SubMode(CURRENT_MODE_TO_STOP);
      IF(CheckSubState(CURRENT_MODE_TO_STOP))THEN
      ModeExecutorColdStandby(RUN_REQUEST:=TRUE);
      END_IF;
      END_IF

      //Stop the MODE blocked in init if the preconditions change
      IF (ModeExecutorColdStandby.Init_State=FALSE  AND ModeActivatorColdStandby.Precondition=FALSE)
THEN
                  Stop_SubMode(COLD_STANDBY);
      END_IF


      IF (ModeExecutorCOLDStandBy.Run_State) THEN
            EMCS_OUT.CURRENT_MODE:= GVL_MAIN.EMCS_OUT.CURRENT_MODE;
            EMCS_OUT.NEXT_SUB_MODE:=RUNNING;
            EMCS_OUT.CURRENT_SUB_MODE:=RUNNING;
      END_IF

END_IF;

IF (ModeExecutorCOLDStandBy.Run_State) THEN
      ModeExecutorCOLDStandBy(RUN_REQUEST:=TRUE);
END_IF

////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////
// EMERGENCY
////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////
IF(ModeActivatorEmergency.ModeActive AND NOT (EMCS_OUT.CURRENT_MODE = Emergency)) THEN
            EMCS_OUT.NEXT_MODE:=EMCS_MODE_ENUM.Emergency;
            EMCS_OUT.NEXT_SUB_MODE:=INIT;
            ModeExecutorEmergency(INIT_REQUEST:=TRUE);

      IF (ModeExecutorEmergency.Init_State   AND NOT GVL_MAIN.FirstScanPLC) THEN
            CURRENT_MODE_TO_STOP:=EMCS_OUT.CURRENT_MODE;
            Stop_SubMode(CURRENT_MODE_TO_STOP);
      IF(CheckSubState(CURRENT_MODE_TO_STOP))THEN
      ModeExecutorEmergency(RUN_REQUEST:=TRUE);
```

```
      END_IF;
      END_IF

      //Stop the MODE blocked in init if the preconditions change
      IF (ModeExecutorEmergency.Init_State=FALSE  AND ModeActivatorEmergency.Precondition=FALSE)
THEN
                  Stop_SubMode(EMERGENCY);
      END_IF

      IF (ModeExecutorEmergency.Run_State) THEN
            EMCS_OUT.CURRENT_MODE:= GVL_MAIN.EMCS_OUT.CURRENT_MODE;
            EMCS_OUT.NEXT_SUB_MODE:=RUNNING;
            EMCS_OUT.CURRENT_SUB_MODE:=RUNNING;
      END_IF

END_IF;

IF (ModeExecutorEmergency.Run_State) THEN
      ModeExecutorEmergency(RUN_REQUEST:=TRUE);
END_IF

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////
// POWER OUT
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////
IF(ModeActivatorPowerout.ModeActive AND NOT (EMCS_OUT.CURRENT_MODE = POWER_OUT)) THEN


            EMCS_OUT.NEXT_MODE:=EMCS_MODE_ENUM.POWER_OUT;
            EMCS_OUT.NEXT_SUB_MODE:=INIT;
            ModeExecutorPowerOut(INIT_REQUEST:=TRUE);


      IF (ModeExecutorPowerOut.Init_State   AND NOT GVL_MAIN.FirstScanPLC) THEN
            CURRENT_MODE_TO_STOP:=EMCS_OUT.CURRENT_MODE;
            Stop_SubMode(CURRENT_MODE_TO_STOP);
      IF(CheckSubState(CURRENT_MODE_TO_STOP))THEN
            ModeExecutorPowerout(RUN_REQUEST:=TRUE);
      END_IF;
```

```
      END_IF

      //Stop the MODE blocked in init if the preconditions change
      IF (ModeExecutorPowerOut.Init_State=FALSE  AND ModeActivatorPowerOut.Precondition=FALSE) THEN

                  Stop_SubMode(POWER_OUT);
      END_IF

      IF (ModeExecutorPowerOut.Run_State) THEN
            EMCS_OUT.CURRENT_MODE:= GVL_MAIN.EMCS_OUT.CURRENT_MODE;
            EMCS_OUT.NEXT_SUB_MODE:=RUNNING;
            EMCS_OUT.CURRENT_SUB_MODE:=RUNNING;
      END_IF

END_IF;

IF (ModeExecutorPowerout.Run_State) THEN
      ModeExecutorPowerout(RUN_REQUEST:=TRUE);
END_IF

////////////////////////////////////////////////////////////////////////////////////////////////////
// IDLE
////////////////////////////////////////////////////////////////////////////////////////////////////
IF(ModeActivatorIdle.ModeActive AND NOT (EMCS_OUT.CURRENT_MODE = IDLE)) THEN


            EMCS_OUT.NEXT_MODE:=EMCS_MODE_ENUM.IDLE;
            EMCS_OUT.NEXT_SUB_MODE:=INIT;
            ModeExecutorIdle(INIT_REQUEST:=TRUE);



      IF (ModeExecutorIdle.Init_State    AND NOT GVL_MAIN.FirstScanPLC) THEN
            CURRENT_MODE_TO_STOP:=EMCS_OUT.CURRENT_MODE;
            Stop_SubMode(CURRENT_MODE_TO_STOP);
      IF(CheckSubState(CURRENT_MODE_TO_STOP))THEN
      ModeExecutorIdle(RUN_REQUEST:=TRUE);
      END_IF;
```

```
      END_IF

      //Stop the MODE blocked in init if the preconditions change
      IF (ModeExecutorIdle.Init_State=FALSE  AND ModeActivatorIdle.Precondition=FALSE) THEN

                  Stop_SubMode(IDLE);
      END_IF

      IF (ModeExecutorIdle.Run_State) THEN
            EMCS_OUT.CURRENT_MODE:= GVL_MAIN.EMCS_OUT.CURRENT_MODE;
            EMCS_OUT.NEXT_SUB_MODE:=RUNNING;
            EMCS_OUT.CURRENT_SUB_MODE:=RUNNING;
      END_IF

END_IF;

IF (ModeExecutorIdle.Run_State) THEN
      ModeExecutorIdle(RUN_REQUEST:=TRUE);
END_IF

//////////////////////////////////////////////////////////////////
PreviousPrecond := CurrentPrecond;
```

4.6 IDLE MODE_EXECUTOR

```
FUNCTION_BLOCK EMCS_MODE_EXECUTOR_IDLE
VAR
Mode_State :INT :=0;    (*  0= Idle 1 = Init, 2 = Run, 3 = Stop *)

Azioni_INIT : BOOL := FALSE;
Azioni_RUN : BOOL := FALSE;
Azioni_STOP : BOOL := FALSE;
Azioni_Idle : BOOL := FALSE;

END_VAR

VAR_INPUT

IDLE_REQUEST :BOOL;
```

```
INIT_REQUEST :BOOL;
RUN_REQUEST :BOOL;
STOP_REQUEST :BOOL;

IDLE_RUN_REQ :R_TRIG;
IDLE_IDLE_REQ :R_TRIG;
IDLE_INIT_REQ :R_TRIG;
IDLE_STOP_REQ :R_TRIG;



END_VAR

VAR_OUTPUT

 CurrentSubState :EMCS_SUB_MODE;
 Init_State :BOOL;
 Run_State :BOOL;
 Stop_State :BOOL;

END_VAR


IDLE_INIT_REQ(CLK:=INIT_REQUEST);
IDLE_IDLE_REQ(CLK:=IDLE_REQUEST);
IDLE_RUN_REQ(CLK:=RUN_REQUEST);
IDLE_STOP_REQ(CLK:=STOP_REQUEST);

IF IDLE_IDLE_REQ.Q THEN
 Mode_State:=0;
 IDLE_REQUEST:=FALSE;
 END_IF;

IF IDLE_INIT_REQ.Q THEN
 Mode_State:=1;
 INIT_REQUEST:=FALSE;
 END_IF;

IF IDLE_RUN_REQ.Q THEN
```

```
 Mode_State:=2;
 RUN_REQUEST:=FALSE;
 END_IF;


IF IDLE_STOP_REQ.Q THEN
 Mode_State:=3;
 STOP_REQUEST:=FALSE;
 END_IF;


CASE Mode_State OF

    1: (* Init *)

            currentSubState.CURRENT_SUB_MODE_STATE :=Mode_State;

            PCS_EMCS_IDLE_REQUEST();

        Azioni_INIT := TRUE;  (* Esegue delle azioni preliminari per avviare la modalità *)
            Azioni_Idle := FALSE;
            INIT_REQUEST:=FALSE;

            IF (TRUE) THEN
            Init_State:=1;

             (*ricevuto il consenso dopo le azioni preliminari si avvia effettivamente la modalità*)
            END_IF;

    2: (* Run *)
      Azioni_RUN := TRUE;
        Init_State:=0;
        Azioni_INIT := FALSE;
        Init_State:=0;
        RUN_REQUEST:=FALSE;

        IF (TRUE) THEN  (*ricevuto il consenso dopo le azioni di run *)

             Run_State:=1;
```

```
            CurrentSubState.CURRENT_SUB_MODE_STATE := Running;
            EMCS_OUT.CURRENT_MODE:= EMCS_MODE_ENUM.IDLE;

            FuelCell_Ensure_Stop();
            BESS_Charge_IdleMode();

           IF NOT ProcessFullyCharged THEN
                Electrolyzer_Start_IdleMode();
           END_IF;
       END_IF;


    3: (* Stop *)

           Run_State:=0;
           Mode_State :=3;
         Azioni_STOP := TRUE;
           CurrentSubState.CURRENT_SUB_MODE_STATE :=Stopping;


           IF (TRUE) THEN
           Stop_state:=1;
           Mode_state :=0; (*ricevuto conferma che tutto sia in stop*)
           END_IF;

      0: (* Idle *)

            Stop_State :=FALSE;
            Run_State :=FALSE;
            Init_State :=FALSE;
            Azioni_run := FALSE;
            Azioni_STOP := FALSE;
            Azioni_INIT := FALSE;

         Azioni_Idle := TRUE;
            CurrentSubState.CURRENT_SUB_MODE_STATE :=Idle_SUB;

END_CASE
```

4.7 IMPORT CHARGING MODE EXECUTOR

```
FUNCTION_BLOCK EMCS_MODE_EXECUTOR_IMPORT_CHARGE
VAR
Mode_State :INT :=0;    (*  0= Idle 1 = Init, 2 = Run, 3 = Stop *)

Azioni_INIT : BOOL := FALSE;
Azioni_RUN : BOOL := FALSE;
Azioni_STOP : BOOL := FALSE;
Azioni_Idle : BOOL := FALSE;

END_VAR

VAR_INPUT

IDLE_REQUEST :BOOL;
INIT_REQUEST :BOOL;
RUN_REQUEST :BOOL;
STOP_REQUEST :BOOL;

IMPORT_CHARGE_RUN_REQ :R_TRIG;
IMPORT_CHARGE_IDLE_REQ :R_TRIG;
IMPORT_CHARGE_INIT_REQ :R_TRIG;
IMPORT_CHARGE_STOP_REQ :R_TRIG;



END_VAR

VAR_OUTPUT

 CurrentSubState :EMCS_SUB_MODE;
 Init_State :BOOL;
 Run_State :BOOL;
 Stop_State :BOOL;

END_VAR

IMPORT_CHARGE_INIT_REQ(CLK:=INIT_REQUEST);
IMPORT_CHARGE_IDLE_REQ(CLK:=IDLE_REQUEST);
```

```
IMPORT_CHARGE_RUN_REQ(CLK:=RUN_REQUEST);
IMPORT_CHARGE_STOP_REQ(CLK:=STOP_REQUEST);

IF IMPORT_CHARGE_IDLE_REQ.Q THEN
 Mode_State:=0;
 IDLE_REQUEST:=FALSE;
 END_IF;

IF IMPORT_CHARGE_INIT_REQ.Q THEN
 Mode_State:=1;
 INIT_REQUEST:=FALSE;
 END_IF;

IF IMPORT_CHARGE_RUN_REQ.Q THEN
 Mode_State:=2;
 RUN_REQUEST:=FALSE;
 END_IF;


IF IMPORT_CHARGE_STOP_REQ.Q THEN
 Mode_State:=3;
 STOP_REQUEST:=FALSE;
 END_IF;


CASE Mode_State OF

    1: (* Stato Init *)

           currentSubState.CURRENT_SUB_MODE_STATE :=Mode_State;
           Azioni_Idle := FALSE;

       PCS_EMCS_IMPORT_CHARGE_REQUEST();
         INIT_REQUEST:=FALSE;

           IF (PCS_Private.PermEnter_ImportAbleToCharge) THEN
                 IF (BESS_Can_Stop() AND FuelCell_Can_Stop()) THEN
                       Init_State:=1;
                 END_IF;
```

```
        END_IF;

2: (* Run *)
  Azioni_RUN := TRUE;
    Init_State:=0;
    Azioni_INIT := FALSE;
    RUN_REQUEST :=FALSE;

    BESS_Charge_ImportMode();

    IF (TRUE) THEN

         Run_State:=1;
         CurrentSubState.CURRENT_SUB_MODE_STATE := Running;
         EMCS_OUT.CURRENT_MODE:= EMCS_MODE_ENUM.IMPORT_ABLE_TO_CHARGE;

         FuelCell_Ensure_Stop();

        IF NOT ProcessFullyCharged THEN
             Electrolyzer_Start_ImportMode();
        END_IF;
    END_IF;

3: (* Stato Stop *)

        Run_State:=0;
        Mode_State :=3;
      Azioni_STOP := TRUE;
        CurrentSubState.CURRENT_SUB_MODE_STATE :=Stopping;

        IF (TRUE) THEN
        Stop_state:=1;
        Mode_state :=0;
        END_IF;

  0: (* Idle *)

         Stop_State :=FALSE;
```

```
            Run_State :=FALSE;
            Init_State :=FALSE;
            Azioni_run := FALSE;
            Azioni_STOP := FALSE;
            Azioni_INIT := FALSE;

             INIT_REQUEST:=FALSE;
             STOP_REQUEST:=FALSE;
             RUN_REQUEST:=FALSE;

         Azioni_Idle := TRUE;
            CurrentSubState.CURRENT_SUB_MODE_STATE :=Idle_SUB;

END_CASE
```

4.8 IMPORT NOT ABLE TO CHARGE EXECUTOR

```
FUNCTION_BLOCK EMCS_MODE_EXECUTOR_IMPORT_NOT_CHARGE
VAR
Mode_State :INT :=0;    (*  0= Idle 1 = Init, 2 = Run, 3 = Stop *)

Azioni_INIT : BOOL := FALSE;
Azioni_RUN : BOOL := FALSE;
Azioni_STOP : BOOL := FALSE;
Azioni_Idle : BOOL := FALSE;

END_VAR

VAR_INPUT

IDLE_REQUEST :BOOL;
INIT_REQUEST :BOOL;
RUN_REQUEST :BOOL;
STOP_REQUEST :BOOL;

IMPORT_NOT_CHARGE_RUN_REQ :R_TRIG;
IMPORT_NOT_CHARGE_IDLE_REQ :R_TRIG;
IMPORT_NOT_CHARGE_INIT_REQ :R_TRIG;
IMPORT_NOT_CHARGE_STOP_REQ :R_TRIG;
```

```
END_VAR

VAR_OUTPUT

 CurrentSubState :EMCS_SUB_MODE;
 Init_State :BOOL;
 Run_State :BOOL;
 Stop_State :BOOL;

END_VAR


IMPORT_NOT_CHARGE_INIT_REQ(CLK:=INIT_REQUEST);
IMPORT_NOT_CHARGE_IDLE_REQ(CLK:=IDLE_REQUEST);
IMPORT_NOT_CHARGE_RUN_REQ(CLK:=RUN_REQUEST);
IMPORT_NOT_CHARGE_STOP_REQ(CLK:=STOP_REQUEST);

IF IMPORT_NOT_CHARGE_IDLE_REQ.Q THEN
 Mode_State:=0;
 IDLE_REQUEST:=FALSE;
 END_IF;

IF IMPORT_NOT_CHARGE_INIT_REQ.Q THEN
 Mode_State:=1;
 INIT_REQUEST:=FALSE;
 END_IF;

IF IMPORT_NOT_CHARGE_RUN_REQ.Q THEN
 Mode_State:=2;
 RUN_REQUEST:=FALSE;
 END_IF;


IF IMPORT_NOT_CHARGE_STOP_REQ.Q THEN
 Mode_State:=3;
 STOP_REQUEST:=FALSE;
```

```
 END_IF;


CASE Mode_State OF

    1: (* Stato Init *)

            currentSubState.CURRENT_SUB_MODE_STATE :=Mode_State;
            Azioni_Idle := FALSE;
            INIT_REQUEST:=FALSE;

        PCS_EMCS_IMPORT_NOT_CHARGE_REQUEST();


            IF (PCS_Private.PermEnter_ImportNotAbleToCharge) THEN
                  IF (BESS_Can_Stop() AND FuelCell_Can_Stop()) THEN
                  Init_State:=1;
                  END_IF;

            END_IF;

    2: (* Stato Run *)
      Azioni_RUN := TRUE;
        Init_State:=0;
        Azioni_INIT := FALSE;
        RUN_REQUEST:=FALSE;

        IF (TRUE) THEN

             Run_State:=1;
             CurrentSubState.CURRENT_SUB_MODE_STATE := Running;
             EMCS_OUT.CURRENT_MODE:= EMCS_MODE_ENUM.IMPORT_NOT_ABLE_TO_CHARGE;

            BESS_Set_Idle();
            Electrolyzer_Stop();
            FuelCell_Ensure_Stop();
        END_IF;
```

```
    3: (* Stato Stop *)

          Run_State:=0;
          Mode_State :=3;
        Azioni_STOP := TRUE;
          CurrentSubState.CURRENT_SUB_MODE_STATE :=Stopping;

          IF (TRUE) THEN
          Stop_state:=1;
          Mode_state :=0;
          END_IF;

      0: (* Stato Idle *)

           Stop_State :=FALSE;
           Run_State :=FALSE;
           Init_State :=FALSE;
           Azioni_run := FALSE;
           Azioni_STOP := FALSE;
           Azioni_INIT := FALSE;

           INIT_REQUEST:=FALSE;
           STOP_REQUEST:=FALSE;
           RUN_REQUEST:=FALSE;

        Azioni_Idle := TRUE;
           CurrentSubState.CURRENT_SUB_MODE_STATE :=Idle_SUB;

END_CASE
```

4.9 HOTSTANDBY MODE_EXECUTOR

```
FUNCTION_BLOCK EMCS_MODE_EXECUTOR_HOTSTANDBY
VAR
      Mode_State :INT :=0;    (*  0= Idle 1 = Init, 2 = Run, 3 = Stop *)
      Azioni_INIT : BOOL := FALSE;
      Azioni_RUN : BOOL := FALSE;
```

```
      Azioni_STOP : BOOL := FALSE;
      Azioni_Idle : BOOL := FALSE;
END_VAR
VAR_INPUT
      IDLE_REQUEST :BOOL;
      INIT_REQUEST :BOOL;
      RUN_REQUEST :BOOL;
      STOP_REQUEST :BOOL;

      HOTSTANDBY_RUN_REQ :R_TRIG;
      HOTSTANDBY_IDLE_REQ :R_TRIG;
      HOTSTANDBY_INIT_REQ :R_TRIG;
      HOTSTANDBY_STOP_REQ :R_TRIG;
END_VAR
VAR_OUTPUT
      CurrentSubState :EMCS_SUB_MODE;
      Init_WaitForPermissive :BOOL;
      Init_State :BOOL;
      Run_State :BOOL;
      Stop_State :BOOL;
END_VAR


HOTSTANDBY_INIT_REQ(CLK:=INIT_REQUEST);
HOTSTANDBY_IDLE_REQ(CLK:=IDLE_REQUEST);
HOTSTANDBY_RUN_REQ(CLK:=RUN_REQUEST);
HOTSTANDBY_STOP_REQ(CLK:=STOP_REQUEST);

IF HOTSTANDBY_IDLE_REQ.Q THEN
 Mode_State:=0;
 IDLE_REQUEST:=FALSE;
 END_IF;

IF HOTSTANDBY_INIT_REQ.Q THEN
 Mode_State:=1;
 INIT_REQUEST:=FALSE;
 END_IF;

IF HOTSTANDBY_RUN_REQ.Q THEN
```

```
 Mode_State:=2;
 RUN_REQUEST:=FALSE;
 END_IF;


IF HOTSTANDBY_STOP_REQ.Q THEN
 Mode_State:=3;
 STOP_REQUEST:=FALSE;
 END_IF;


CASE Mode_State OF

    1: (* Stato Init *)

            currentSubState.CURRENT_SUB_MODE_STATE :=Mode_State;

            PCS_EMCS_HOT_STANDBY_REQUEST();

        Azioni_INIT := TRUE;
            Azioni_Idle := FALSE;

            INIT_REQUEST:=FALSE;

            Init_WaitForPermissive := TRUE;
            IF (PCS_Private.PermEnter_HotStandBy) THEN
                  Init_State:=1;
                  Init_WaitForPermissive := FALSE;
            END_IF;

    2: (* Run *)
      Azioni_RUN := TRUE;
        Init_State:=0;
        Azioni_INIT := FALSE;

        IF (TRUE) THEN

             Run_State:=1;
             CurrentSubState.CURRENT_SUB_MODE_STATE := Running;
```

```
            EMCS_OUT.CURRENT_MODE:= EMCS_MODE_ENUM.HOT_STANDBY;

            FuelCell_Ensure_Stop();
        END_IF;

    3: (* Stop *)

            Run_State:=0;
            Mode_State :=3;
          Azioni_STOP := TRUE;
            CurrentSubState.CURRENT_SUB_MODE_STATE :=Stopping;

            IF (TRUE) THEN
            Stop_state:=1;
            Mode_state :=0;
            END_IF;

      0: (* Idle *)

             Stop_State :=FALSE;
             Run_State :=FALSE;
             Init_State :=FALSE;
             Azioni_run := FALSE;
             Azioni_STOP := FALSE;
             Azioni_INIT := FALSE;

             Init_WaitForPermissive := FALSE;

         Azioni_Idle := TRUE;
             CurrentSubState.CURRENT_SUB_MODE_STATE :=Idle_SUB;
        END_CASE
```

4.10 EXPORT MODE_EXECUTOR

```
FUNCTION_BLOCK EMCS_MODE_EXECUTOR_EXPORT
VAR
Mode_State :INT :=0;    (*  0= Idle 1 = Init, 2 = Run, 3 = Stop *)

Azioni_INIT : BOOL := FALSE;
Azioni_RUN : BOOL := FALSE;
```

```
Azioni_STOP : BOOL := FALSE;
Azioni_Idle : BOOL := FALSE;

END_VAR

VAR_INPUT

IDLE_REQUEST :BOOL;
INIT_REQUEST :BOOL;
RUN_REQUEST :BOOL;
STOP_REQUEST :BOOL;

EXPORT_RUN_REQ :R_TRIG;
EXPORT_IDLE_REQ :R_TRIG;
EXPORT_INIT_REQ :R_TRIG;
EXPORT_STOP_REQ :R_TRIG;



END_VAR

VAR_OUTPUT

 CurrentSubState :EMCS_SUB_MODE;
 Init_WaitForPermissive :BOOL;
 Init_State :BOOL;
 Run_State :BOOL;
 Stop_State :BOOL;

END_VAR


EXPORT_INIT_REQ(CLK:=INIT_REQUEST);
EXPORT_IDLE_REQ(CLK:=IDLE_REQUEST);
EXPORT_RUN_REQ(CLK:=RUN_REQUEST);
EXPORT_STOP_REQ(CLK:=STOP_REQUEST);

IF EXPORT_IDLE_REQ.Q THEN
 Mode_State:=0;
```

```
 IDLE_REQUEST:=FALSE;

 END_IF;

IF EXPORT_INIT_REQ.Q THEN
 Mode_State:=1;
 INIT_REQUEST:=FALSE;
 END_IF;

IF EXPORT_RUN_REQ.Q THEN
 Mode_State:=2;
 RUN_REQUEST:=FALSE;
 END_IF;


IF EXPORT_STOP_REQ.Q THEN
 Mode_State:=3;
 STOP_REQUEST:=FALSE;
 END_IF;


CASE Mode_State OF

    1: (* Init *)

            currentSubState.CURRENT_SUB_MODE_STATE :=Mode_State;
            EXPORT_TEXT1 := "Stopping Electrloyzer";//test


            IF (Electrolyzer_Stop()) THEN

                  PCS_EMCS_EXPORT_REQUEST();
                  EXPORT_TEXT1 := "Waiting for PCS Response";//test
            END_IF

            INIT_REQUEST:=FALSE;

            Init_WaitForPermissive := TRUE;
            IF (PCS_Private.PermEnter_Export) THEN
```

```
            Init_State:=1;    effettivamente la modalità*)
            Init_WaitForPermissive := FALSE;
        END_IF;

2: (*Run *)

  BESS_Discharge_ExportMode();

    PCS_EMCS_FC_START_REQUEST();

    Init_State:=0;
    RUN_REQUEST:=FALSE;

    EXPORT_TEXT1 := "Waiting for PCS Response to Start request";
    IF (PCS_Internal.PermToStartFC) THEN

         FuelCell_START_EXPORT_MODE();
    END_IF;

   Run_State:=1;
  CurrentSubState.CURRENT_SUB_MODE_STATE := Running;
  EMCS_OUT.CURRENT_MODE:= EMCS_MODE_ENUM.EXPORT;


3: (* Stop *)

        Run_State:=0;
        Mode_State :=3;
      Azioni_STOP := TRUE;
        STOP_REQUEST:=FALSE;
        CurrentSubState.CURRENT_SUB_MODE_STATE :=Stopping;
        Stop_Fuel_Cell();

        IF (FuelCell_Stopped) THEN
        Stop_state:=1;
        Mode_state :=0;
        END_IF;

  0: (* Idle *)
```

```
            Stop_State :=FALSE;
            Run_State :=FALSE;
            Init_State :=FALSE;
            Azioni_run := FALSE;
            Azioni_STOP := FALSE;
            Azioni_INIT := FALSE;

            INIT_REQUEST:=FALSE;
            RUN_REQUEST:=FALSE;
            STOP_REQUEST:=FALSE;

           Init_WaitForPermissive := FALSE;

        Azioni_Idle := TRUE;
            CurrentSubState.CURRENT_SUB_MODE_STATE :=Idle_SUB;
END_CASE
```

4.11 COLDSTANDBY MODE EXECUTOR

```
FUNCTION_BLOCK EMCS_MODE_EXECUTOR_COLDSTANDBY
VAR
Mode_State :INT :=0;    (*  0= Idle 1 = Init, 2 = Run, 3 = Stop *)

Azioni_INIT : BOOL := FALSE;
Azioni_RUN : BOOL := FALSE;
Azioni_STOP : BOOL := FALSE;
Azioni_Idle : BOOL := FALSE;

END_VAR

VAR_INPUT

IDLE_REQUEST :BOOL;
INIT_REQUEST :BOOL;
RUN_REQUEST :BOOL;
STOP_REQUEST :BOOL;

COLDSTANDBY_RUN_REQ :R_TRIG;
COLDSTANDBY_IDLE_REQ :R_TRIG;
```

```
COLDSTANDBY_INIT_REQ :R_TRIG;
COLDSTANDBY_STOP_REQ :R_TRIG;



END_VAR

VAR_OUTPUT

 CurrentSubState :EMCS_SUB_MODE;
 Init_State :BOOL;
 Run_State :BOOL;
 Stop_State :BOOL;

END_VAR


COLDSTANDBY_INIT_REQ(CLK:=INIT_REQUEST);
COLDSTANDBY_IDLE_REQ(CLK:=IDLE_REQUEST);
COLDSTANDBY_RUN_REQ(CLK:=RUN_REQUEST);
COLDSTANDBY_STOP_REQ(CLK:=STOP_REQUEST);

IF COLDSTANDBY_IDLE_REQ.Q THEN
 Mode_State:=0;
 IDLE_REQUEST:=FALSE;
 END_IF;

IF COLDSTANDBY_INIT_REQ.Q THEN
 Mode_State:=1;
 INIT_REQUEST:=FALSE;
 END_IF;

IF COLDSTANDBY_RUN_REQ.Q THEN
 Mode_State:=2;
 RUN_REQUEST:=FALSE;
 END_IF;


IF COLDSTANDBY_STOP_REQ.Q THEN
```

```
 Mode_State:=3;
 STOP_REQUEST:=FALSE;
 END_IF;


CASE Mode_State OF

    1: (* Stato Init *)

            currentSubState.CURRENT_SUB_MODE_STATE :=Mode_State;

        Azioni_INIT := TRUE;
            Azioni_Idle := FALSE;

            PCS_EMCS_COLD_STANDBY_REQUEST();

            INIT_REQUEST:=FALSE;

            IF (TRUE) THEN
                  Init_State:=1;
            END_IF;

    2: (* Run *)
      Azioni_RUN := TRUE;
        Init_State:=0;
        Azioni_INIT := FALSE;
        RUN_REQUEST :=FALSE;

        // Reset the events of permission removing that causes to move in ColdeStandBy (this mode)
        PCS_Private.PermRemovedHotStanby := FALSE;
        PCS_Private.PermRemovedExport := FALSE;

        IF (TRUE) THEN  (*ricevuto il consenso dopo le azioni di run *)

             Run_State:=1;
             CurrentSubState.CURRENT_SUB_MODE_STATE := Running;
             EMCS_OUT.CURRENT_MODE:= EMCS_MODE_ENUM.COLD_STANDBY;

             FuelCell_Ensure_Sleep();
```

```
        END_IF;



    3: (* Stop *)

          Run_State:=0;
          Mode_State :=3;
        Azioni_STOP := TRUE;
          CurrentSubState.CURRENT_SUB_MODE_STATE :=Stopping;

          IF (TRUE) THEN
                Stop_state:=1;
                Mode_state :=0; (*ricevuto conferma che tutto sia in stop*)
          END_IF;

      0: (* Idle *)

           Stop_State :=FALSE;
           Run_State :=FALSE;
           Init_State :=FALSE;
           Azioni_run := FALSE;
           Azioni_STOP := FALSE;
           Azioni_INIT := FALSE;

           INIT_REQUEST:=FALSE;
           RUN_REQUEST:=FALSE;
           STOP_REQUEST:=FALSE;

        Azioni_Idle := TRUE;
           CurrentSubState.CURRENT_SUB_MODE_STATE :=Idle_SUB;

END_CASE
```

4.12 EMERGENCY MODE EXECUTOR

```
FUNCTION_BLOCK EMCS_MODE_EXECUTOR_EMERGENCY
VAR
Mode_State :INT :=0;    (*  0= Idle 1 = Init, 2 = Run, 3 = Stop *)
```

```
Azioni_INIT : BOOL := FALSE;
Azioni_RUN : BOOL := FALSE;
Azioni_STOP : BOOL := FALSE;
Azioni_Idle : BOOL := FALSE;

END_VAR

VAR_INPUT

IDLE_REQUEST :BOOL;
INIT_REQUEST :BOOL;
RUN_REQUEST :BOOL;
STOP_REQUEST :BOOL;

EMERGENCY_RUN_REQ :R_TRIG;
EMERGENCY_IDLE_REQ :R_TRIG;
EMERGENCY_INIT_REQ :R_TRIG;
EMERGENCY_STOP_REQ :R_TRIG;



END_VAR

VAR_OUTPUT

 CurrentSubState :EMCS_SUB_MODE;
 Init_State :BOOL;
 Run_State :BOOL;
 Stop_State :BOOL;

END_VAR


EMERGENCY_INIT_REQ(CLK:=INIT_REQUEST);
EMERGENCY_IDLE_REQ(CLK:=IDLE_REQUEST);
EMERGENCY_RUN_REQ(CLK:=RUN_REQUEST);
EMERGENCY_STOP_REQ(CLK:=STOP_REQUEST);
```

```
IF EMERGENCY_IDLE_REQ.Q THEN
 Mode_State:=0;
 IDLE_REQUEST:=FALSE;
 END_IF;

IF EMERGENCY_INIT_REQ.Q THEN
 Mode_State:=1;
 INIT_REQUEST:=FALSE;
 END_IF;

IF EMERGENCY_RUN_REQ.Q THEN
 Mode_State:=2;
 RUN_REQUEST:=FALSE;
 END_IF;


IF EMERGENCY_STOP_REQ.Q THEN
 Mode_State:=3;
 STOP_REQUEST:=FALSE;
 END_IF;


CASE Mode_State OF

    1: (*Init *)

           currentSubState.CURRENT_SUB_MODE_STATE :=Mode_State;

        Azioni_INIT := TRUE;
           Azioni_Idle := FALSE;

           PCS_EMCS_EMERGENCY_REQUEST();

           INIT_REQUEST:=FALSE;

           IF (PCS_Private.PermEnter_Emergency) THEN
                 Init_State:=1;
           END_IF;
```

```
2: (* Run *)
  Azioni_RUN := TRUE;
    Init_State:=0;
    Azioni_INIT := FALSE;
    RUN_REQUEST:=FALSE;

    IF (SolarFarm_Healthy) THEN

         Run_State:=1;
         Electrolyzer_ForceStop();
         BESS_Ensure_Is_Started();
         // Load Shed Stage 1
         PCS_LoadShedStage1();

         IF(BESS_SocAboveLow() AND BESS_SocBelowHi()) THEN
              Emergency_BESS_MEDIUM_SOC();
         ELSIF(NOT BESS_SocBelowHi()) THEN  // Above High level
              Emergency_BESS_HIGH_SOC();
         ELSE                                          // Below High level
              Emergency_BESS_LOW_SOC();
         END_IF;

         IF(BESS_AvailableEnergyPerc <= BESS_LIMIT_FOR_ADV_LOAD_SHEDDING) THEN
              PCS_LoadShedStage2();
         END_IF;

         CurrentSubState.CURRENT_SUB_MODE_STATE := Running;
         EMCS_OUT.CURRENT_MODE:= EMCS_MODE_ENUM.EMERGENCY;

    END_IF;

3: (* Stato Stop *)

        Run_State:=0;
        Mode_State :=3;
      Azioni_STOP := TRUE;
        STOP_REQUEST:=FALSE;
        CurrentSubState.CURRENT_SUB_MODE_STATE :=Stopping;
```

```
            IF (TRUE) THEN
            Stop_state:=1;
            Mode_state :=0;
            END_IF;

      0: (* Stato Idle *)

             Stop_State :=FALSE;
             Run_State :=FALSE;
             Init_State :=FALSE;
             Azioni_run := FALSE;
             Azioni_STOP := FALSE;
             Azioni_INIT := FALSE;

         Azioni_Idle := TRUE;
             CurrentSubState.CURRENT_SUB_MODE_STATE :=Idle_SUB;

END_CASE

IF ((Mode_State<>2) AND ModeActivatorEmergency.Precondition=FALSE) THEN
      Mode_State:=0;
      //Stop_SubMode(8);
END_IF
```

4.13 POWEROUT MODE EXECUTOR

```
FUNCTION_BLOCK EMCS_MODE_EXECUTOR_POWEROUT
VAR
Mode_State :INT :=0;    (*  0= Idle 1 = Init, 2 = Run, 3 = Stop *)

Azioni_INIT : BOOL := FALSE;
Azioni_RUN : BOOL := FALSE;
Azioni_STOP : BOOL := FALSE;
Azioni_Idle : BOOL := FALSE;

END_VAR
```

```
VAR_INPUT

IDLE_REQUEST :BOOL;
INIT_REQUEST :BOOL;
RUN_REQUEST :BOOL;
STOP_REQUEST :BOOL;

POWEROUT_RUN_REQ :R_TRIG;
POWEROUT_IDLE_REQ :R_TRIG;
POWEROUT_INIT_REQ :R_TRIG;
POWEROUT_STOP_REQ :R_TRIG;

END_VAR

VAR_OUTPUT

 CurrentSubState :EMCS_SUB_MODE;
 Init_State :BOOL;
 Run_State :BOOL;
 Stop_State :BOOL;

END_VAR


POWEROUT_INIT_REQ(CLK:=INIT_REQUEST);
POWEROUT_IDLE_REQ(CLK:=IDLE_REQUEST);
POWEROUT_RUN_REQ(CLK:=RUN_REQUEST);
POWEROUT_STOP_REQ(CLK:=STOP_REQUEST);

IF POWEROUT_IDLE_REQ.Q THEN
 Mode_State:=0;
 IDLE_REQUEST:=FALSE;
 END_IF;

IF POWEROUT_INIT_REQ.Q THEN
 Mode_State:=1;
 INIT_REQUEST:=FALSE;
 END_IF;
```

```
IF POWEROUT_RUN_REQ.Q THEN
 Mode_State:=2;
 RUN_REQUEST:=FALSE;
 END_IF;


IF POWEROUT_STOP_REQ.Q THEN
 Mode_State:=3;
 STOP_REQUEST:=FALSE;
 END_IF;


CASE Mode_State OF

    1: (* Init *)

            currentSubState.CURRENT_SUB_MODE_STATE :=Mode_State;

            PCS_EMCS_POWEROUT_REQUEST();

        Azioni_INIT := TRUE;
            Azioni_Idle := FALSE;
            INIT_REQUEST:=FALSE;

            IF (TRUE) THEN
            Init_State:=1;

            END_IF;

    2: (* Stato Run *)
      Azioni_RUN := TRUE;
        Init_State:=0;
        Azioni_INIT := FALSE;
        RUN_REQUEST:=FALSE;

      IF (TRUE) THEN

            Run_State:=1;
            OpenAllCircuitBreakersBusMV();
```

```
        FuelCell_Ensure_Sleep();
        SolarFarm_EnsureStop();
        Electrolyzer_Stop();
        BESS_Ensure_Is_Stopped();

        CurrentSubState.CURRENT_SUB_MODE_STATE := Running;
        EMCS_OUT.CURRENT_MODE:= EMCS_MODE_ENUM.POWER_OUT;
    END_IF;



3: (* Stop *)

        Run_State:=0;
        Mode_State :=3;
       Azioni_STOP := TRUE;
        STOP_REQUEST:=FALSE;
        CurrentSubState.CURRENT_SUB_MODE_STATE :=Stopping;

        IF (TRUE) THEN

        //EMCS_Internal.AutoManualMode:=2; //Research
        //CloseAllCircuitBreakersBusMV();


        Stop_state:=1;
        Mode_state :=0; (*ricevuto conferma che tutto sia in stop*)
        END_IF;

  0: (* Idle *)

         Stop_State :=FALSE;
         Run_State :=FALSE;
         Init_State :=FALSE;
         Azioni_run := FALSE;
         Azioni_STOP := FALSE;
         Azioni_INIT := FALSE;
```

```
        Azioni_Idle := TRUE;
            CurrentSubState.CURRENT_SUB_MODE_STATE :=Idle_SUB;

END_CASE
```

4.14 MAINTENANCE MODE EXECUTOR

```
FUNCTION_BLOCK EMCS_MODE_EXECUTOR_MAINTENANCE
VAR
Mode_State :INT :=0;    (* 1 = Init, 2 = Run, 3 = Stop  0= Idle*)

Azioni_INIT : BOOL := FALSE;
Azioni_RUN : BOOL := FALSE;
Azioni_STOP : BOOL := FALSE;
Azioni_Idle : BOOL := FALSE;

FirstRunExecution : R_TRIG;
FUEL_CELL_STOPPPED : BOOL := FALSE;

END_VAR

VAR_INPUT

IDLE_REQUEST :BOOL;
INIT_REQUEST :BOOL;
RUN_REQUEST :BOOL;
STOP_REQUEST :BOOL;

MAINTENANCE_RUN_REQ :R_TRIG;
MAINTENANCE_IDLE_REQ :R_TRIG;
MAINTENANCE_INIT_REQ :R_TRIG;
MAINTENANCE_STOP_REQ :R_TRIG;



END_VAR

VAR_OUTPUT
```

```
 CurrentSubState :EMCS_SUB_MODE;
 Init_State :BOOL;
 Run_State :BOOL;
 Stop_State :BOOL;

END_VAR


MAINTENANCE_INIT_REQ(CLK:=INIT_REQUEST);
MAINTENANCE_IDLE_REQ(CLK:=IDLE_REQUEST);
MAINTENANCE_RUN_REQ(CLK:=RUN_REQUEST);
MAINTENANCE_STOP_REQ(CLK:=STOP_REQUEST);

IF MAINTENANCE_IDLE_REQ.Q THEN
 Mode_State:=0;
 IDLE_REQUEST:=FALSE;
 END_IF;

IF MAINTENANCE_INIT_REQ.Q THEN
 Mode_State:=1;
 INIT_REQUEST:=FALSE;
 END_IF;

IF MAINTENANCE_RUN_REQ.Q THEN
 Mode_State:=2;
 RUN_REQUEST:=FALSE;
 END_IF;


IF MAINTENANCE_STOP_REQ.Q THEN
 Mode_State:=3;
 STOP_REQUEST:=FALSE;
 END_IF;


CASE Mode_State OF

    1: (* Init *)
```

```
            currentSubState.CURRENT_SUB_MODE_STATE :=Mode_State;

            PCS_EMCS_MAINTENANCE_REQUEST();

        Azioni_INIT := TRUE;  (* Esegue delle azioni preliminari per avviare la modalità *)
            Azioni_Idle := FALSE;
            FUEL_CELL_STOPPPED := FALSE;

            IF (TRUE) THEN
                  Init_State:=TRUE;        (*ricevuto il consenso dopo le azioni preliminari si avvia
effettivamente la modalità*)
            END_IF;

    2: (* Run *)
      Azioni_RUN := TRUE;
        Init_State:=0;
        Azioni_INIT := FALSE;
        RUN_REQUEST :=FALSE;

        IF (TRUE) THEN  (*ricevuto il consenso dopo le azioni di run *)

            Run_State:=1;
            CurrentSubState.CURRENT_SUB_MODE_STATE := Running;
            EMCS_OUT.CURRENT_MODE:= EMCS_MODE_ENUM.MAINTENANCE;

            FirstRunExecution(CLK:=Run_State);

            IF(FirstRunExecution.Q) THEN
                  BESS_Ensure_Is_Stopped();
                  Electrolyzer_Stop();

              PCS_Private.PermRemovedHotStanby := FALSE;
              PCS_Private.PermRemovedExport := FALSE;

            END_IF;

            IF(NOT FUEL_CELL_STOPPPED) THEN
                  FUEL_CELL_STOPPPED := FuelCEll_Ensure_Sleep();
            END_IF
```

```
        END_IF;

    3: (* Stop *)

          Run_State:=0;
          Mode_State :=3;
         Azioni_STOP := TRUE;
          CurrentSubState.CURRENT_SUB_MODE_STATE :=Stopping;

          IF (TRUE) THEN
          Stop_state:=1;
          Mode_state :=0; (*ricevuto conferma che tutto sia in stop*)
          END_IF;

      0: (* Idle *)

           Stop_State :=FALSE;
           Run_State :=FALSE;
           Init_State :=FALSE;
           Azioni_run := FALSE;
           Azioni_STOP := FALSE;
           Azioni_INIT := FALSE;

        Azioni_Idle := TRUE;
           CurrentSubState.CURRENT_SUB_MODE_STATE :=Idle_SUB;

END_CASE
```

4.15 RESEARCH MODE EXECUTOR

```
FUNCTION_BLOCK EMCS_MODE_EXECUTOR_RESEARCH
VAR
Mode_State :INT :=0;    (* 1 = Init, 2 = Run, 3 = Stop  0= Idle*)

Azioni_INIT : BOOL := FALSE;
Azioni_RUN : BOOL := FALSE;
Azioni_STOP : BOOL := FALSE;
```

```
Azioni_Idle : BOOL := FALSE;
FirstRunExecution : R_TRIG;
END_VAR

VAR_INPUT

IDLE_REQUEST :BOOL;
INIT_REQUEST :BOOL;
RUN_REQUEST :BOOL;
STOP_REQUEST :BOOL;

RESEARCH_RUN_REQ :R_TRIG;
RESEARCH_INIT_REQ :R_TRIG;
RESEARCH_STOP_REQ :R_TRIG;
RESEARCH_IDLE_REQ :R_TRIG;

FUEL_CELL_STOPPPED : BOOL := FALSE;

END_VAR


VAR_OUTPUT

 CurrentSubState :EMCS_SUB_MODE;
 Init_State :BOOL;
 Run_State :BOOL;
 Stop_State :BOOL;

END_VAR

RESEARCH_INIT_REQ(CLK:=INIT_REQUEST);
RESEARCH_RUN_REQ(CLK:=RUN_REQUEST);
RESEARCH_STOP_REQ(CLK:=STOP_REQUEST);
RESEARCH_IDLE_REQ(CLK:=IDLE_REQUEST);

IF RESEARCH_INIT_REQ.Q THEN
  Mode_State:=1;
  INIT_REQUEST:=FALSE;
 END_IF;
```

```
 IF RESEARCH_IDLE_REQ.Q THEN
  Mode_State:=0;
  IDLE_REQUEST:=FALSE;
 END_IF;

IF RESEARCH_RUN_REQ.Q THEN
  Mode_State:=2;
  RUN_REQUEST:=FALSE;
 END_IF;


IF RESEARCH_STOP_REQ.Q THEN
 Mode_State:=3;
 STOP_REQUEST:=FALSE;
 END_IF;

CASE Mode_State OF
    1: (* Init *)
            currentSubState.CURRENT_SUB_MODE_STATE :=Mode_State;

            PCS_EMCS_RESEARCH_REQUEST();

        Azioni_INIT := TRUE;
            Azioni_Idle := FALSE;
            FUEL_CELL_STOPPPED := FALSE;
            IF (TRUE) THEN
                  Init_State:=TRUE;
            END_IF;

    2: (* Run *)
      Azioni_RUN := TRUE;
        Init_State :=FALSE;
        Azioni_INIT := FALSE;
        RUN_REQUEST :=FALSE;

            FirstRunExecution(CLK:=Run_State);
            IF(FirstRunExecution.Q) THEN
                  // Stop all CPI (Company Provided Items)
```

```
            BESS_Ensure_Is_Stopped();
            Electrolyzer_Stop();
            SolarFarm_EnsureStop();

            PCS_Private.PermRemovedHotStanby := FALSE;
            PCS_Private.PermRemovedExport := FALSE;
      END_IF;

      IF(NOT FUEL_CELL_STOPPPED) THEN
            FUEL_CELL_STOPPPED := FuelCEll_Ensure_Sleep();
      END_IF

   IF (TRUE) THEN  (*ricevuto il consenso dopo le azioni di run *)
       Run_State :=TRUE;
        currentSubState.CURRENT_SUB_MODE_STATE := Running;
       EMCS_OUT.CURRENT_MODE:= EMCS_MODE_ENUM.RESEARCH;
  END_IF;



3: (* Stop *)

      Mode_state :=3;
       Run_State :=FALSE;
      Azioni_STOP := TRUE;
      currentSubState.CURRENT_SUB_MODE_STATE :=Stopping;

      IF (TRUE) THEN
      Stop_State :=TRUE;
      Mode_state :=0; (*ricevuto conferma che tutto sia in stop*)
      END_IF;

 0: (* Idle *)

     Stop_State :=FALSE;
        Run_State :=FALSE;
        Init_State :=FALSE;
        Azioni_run := FALSE;
         Azioni_STOP := FALSE;
```

```
            Azioni_INIT := FALSE;

        Azioni_Idle := TRUE;
           currentSubState.CURRENT_SUB_MODE_STATE :=Idle_SUB;


END_CASE
```

4.16 NONE MODE EXECUTOR

```
FUNCTION_BLOCK EMCS_MODE_EXECUTOR_NONE
VAR
Mode_State :INT :=0;    (* 1 = Init, 2 = Run, 3 = Stop  0= Idle *)

Azioni_INIT : BOOL := FALSE;
Azioni_RUN : BOOL := FALSE;
Azioni_STOP : BOOL := FALSE;
Azioni_Idle : BOOL := FALSE;

END_VAR

VAR_INPUT

IDLE_REQUEST :BOOL;
INIT_REQUEST :BOOL;
RUN_REQUEST :BOOL;
STOP_REQUEST :BOOL;

NONE_RUN_REQ :R_TRIG;
NONE_IDLE_REQ :R_TRIG;
NONE_INIT_REQ :R_TRIG;
NONE_STOP_REQ :R_TRIG;

END_VAR


VAR_OUTPUT

 CurrentSubState :EMCS_SUB_MODE;
```

```
 Init_State :BOOL;
 Run_State :BOOL;
 Stop_State :BOOL;

END_VAR



NONE_INIT_REQ(CLK:=INIT_REQUEST);
NONE_IDLE_REQ(CLK:=IDLE_REQUEST);
NONE_RUN_REQ(CLK:=RUN_REQUEST);
NONE_STOP_REQ(CLK:=STOP_REQUEST);

IF NONE_INIT_REQ.Q THEN
  Mode_State:=1;
  INIT_REQUEST:=FALSE;
 END_IF;

 IF NONE_IDLE_REQ.Q THEN
 Mode_State:=0;
 IDLE_REQUEST:=FALSE;
 END_IF;

IF NONE_RUN_REQ.Q THEN
  Mode_State:=2;
  RUN_REQUEST:=FALSE;
 END_IF;


IF NONE_STOP_REQ.Q THEN
 Mode_State:=3;
 STOP_REQUEST:=FALSE;
 END_IF;

CASE Mode_State OF
    1: (* Init *)
            currentSubState.CURRENT_SUB_MODE_STATE :=Mode_State;

        Azioni_INIT := TRUE;  (* Esegue delle azioni preliminari per avviare la modalità *)
```

```
        Azioni_Idle := FALSE;
        IF (TRUE) THEN
              Init_State :=TRUE;
         (*ricevuto il consenso dopo le azioni preliminari si avvia effettivamente la modalità*)
        END_IF;

2: (* Run *)
  Azioni_RUN := TRUE;
    Init_State :=FALSE;
    Azioni_INIT := FALSE;

    IF (TRUE) THEN
         Run_State :=TRUE;
          currentSubState.CURRENT_SUB_MODE_STATE := Running;
         EMCS_OUT.CURRENT_MODE:= EMCS_MODE_ENUM.NONE;
   END_IF;


3: (* Stop *)
        Mode_state :=3;
         Run_State :=FALSE;
        Azioni_STOP := TRUE;
        currentSubState.CURRENT_SUB_MODE_STATE :=Stopping;

        IF (TRUE) THEN
        Stop_State :=TRUE;
        Mode_state :=0; (*ricevuto conferma che tutto sia in stop*)
        END_IF;
  0: (* Idle *)

         Stop_State :=FALSE;
         Run_State :=FALSE;
         Init_State :=FALSE;
         Azioni_run := FALSE;
          Azioni_STOP := FALSE;
          Azioni_INIT := FALSE;

      Azioni_Idle := TRUE;
        currentSubState.CURRENT_SUB_MODE_STATE :=Idle_SUB;
```

```
END_CASE
```

This document undergoes an issuance cycle within EDMS Fusionlive". Its distribution is electronically controlled. Once printed shall be considered as "NOT CONTROLLED" copy