Master's thesis

Double Master’s Degree in Industrial Engineering and Nuclear Engineering

Documentation, Design, Simulation and Implementation of an Electron Cyclotron (EC) control system for ITER

APPENDIX

VOLUME II / II

Author: David Soriano
Director: Alfredo de Blas
Examination session: June 2017

Escola Tècnica Superior d’Enginyeria Industrial de Barcelona
Disclaimer

This publication reflects the views only of the authors, and Fusion for Energy cannot be held responsible for any use which may be made of the information contained therein.
Contents

A HVPSs State Machines 3
A.1 Main Power Supply (MPS) State Machine 4
A.2 Body Power Supply (BPS) State Machine 5

B Simulations 7
B.1 Simulation 1: Fault in SCM PS during Pulse 8
B.2 Simulation 2: HVPSs Trip signal during Pulse 13

C Automatic test of the model 17
Appendix A

HVPSs State Machines

A.1 Main Power Supply (MPS) State Machine

A.2 Body Power Supply (BPS) State Machine
A.1 Main Power Supply (MPS) State Machine

This design has been done with the help of the technical report about the MPS [18].
A.2 Body Power Supply (BPS) State Machine

This design has been done with the help of the technical report about the BPS [19].
Appendix B

Simulations

B.1 Simulation 1: Fault in SCM PS during Pulse

B.2 Simulation 2: HVPSs Trip signal during Pulse
B.1 Simulation 1: Fault in SCM PS during Pulse

In this first simulation, the system will start from the opening of the model and the HMI, to show the first steps that need to be made before starting any operation.

On the first figure on the left, the user sets the operation mode he wants to use. In this case, the operation mode chosen is the Normal Operation, where all the systems are active (see Table 4.1 in the Report). On the right, all the parameters of the model are loaded by clicking the Load Excel button. Without this, the model cannot be run.

After this, the simulation can be started by pressing the Start button (next figure on the left). Once the simulation is started, the states of each subsystem will appear as well as some colors in the boxes. The simulation time is also displayed.

Then, the first set of auxiliaries is started by pressing the StartSet1 button, which will turn on the SCM Cryogenic system and the Ion Pumps. When both systems are operating in normal conditions, the system will move to SET1READY (figure on the right). As it can be seen in the picture, the SCM PS, the Collector DC and the Gun Coil states are in blue. This is because these three systems need to be configured before they can be turned on (putting the main parameters of the power supply as the output current). This is done at the same time the first set is started.

In all the sequence of figures that will follow, one can notice that the buttons linked to the commands of the Main State Machine (to start/stop or send to not ready the different sets) are only active when they can be used. For example, when starting the first set, the StartSet3 button cannot be pressed. This is to make the HMI easier to understand and avoid mistakes.
When the first set is ready, the second set is started. At the same time the second set is started, the QST Ground connection of the HVPSs is opened:

When the second set is ready, the third one is started:
And finally, when the third set is ready, the pulse can be initiated (left on next figure). After some time, during the pulse, a Fault is injected in the SCM PS (right figure):

The SCM PS belongs to set 2, so this fault will bring the system to SET2NOTREADY. This will automatically stop all the auxiliaries from the third set.
The HVPSs will move to QSTNotGrounded and their voltage will start decreasing slowly:

![Diagram](image)

After some time, the fault will be solved and a general reset will be done to the system. As it can be seen in the next figure, the SCM PS is not in fault state anymore by the Main State Machine will remain in SET2NOTREADY until the user sends the command to start or to stop the second set (left figure). In this case, the user decides to stop it (right figure):

![Diagram](image)
Once all the subsystems from the second set are off and the QST Ground connection of the HVPSs is closed again, the system will then go back to the SET1READY state. The simulation ends here.

When operating the system it is also important to check the evolution of the voltages of the HVPSs. As it can be seen in the next figure, the evolution is as it has been explained in the HVPSs Synchronization state machine: when the Start Pulse command is sent, the voltage of the MPS (yellow curve) starts increasing. When it reaches the 0.2 threshold, the MPS ON signal becomes high (purple curve), which will start the BPS (blue curve). When the BPS reaches its threshold of 0.01, the BPS ON signal becomes high (red curve). Both voltages increase progressively until getting to their references (normalized to 1 in this plot). Then, when the fault appears in the SCM PS is can be seen that the BPS voltage starts decreasing progressively. When it crosses again its threshold, the BPS ON signal becomes low again, making the voltage of the MPS start ramping down as well. When the MPS voltage crosses the 0.2 threshold, the MPS ON signal becomes low again.
### B.2 Simulation 2: HVPSs Trip signal during Pulse

The second simulation consists in observing the evolution of the system in case of sending a Trip signal to the HVPSs during a pulse. The first figure will be the system during a pulse (all the steps to reach this state have been described in Simulation 1) and then the trip signal will be sent from the Fault Injector HMI.

In case of a fault in the HVPSs, their state machines receive an alarm signal that will move their state machine directly to Fast Alarm state. At the same time, when there is a fault in the HVPSs, the Main State Machine will move to SET2NOTREADY, stopping automatically all the auxiliaries from the third set (next figure on the left).

In order to solve this fault, the HVPSs need the Fast Alarm protocol to be executed. This means that someone has been checking what happened to the system and what generated this fault. Once the fault has been identified and solved, the user can send the Fast Alarm Executed signal by pressing the Fast Alarm display in red. The box will change from red to orange and will display Fast Alarm Solved. In this simulation, the alarm has been executed first for the MPS (right figure).
And then for the BPS (next left figure). When both alarms are solved, the HVPSs need the user to send a Reset signal to get back to the main loop. This is done by pressing the Main Reset button (right figure). The Main State Machine will remain in SET2NOTREADY until the user decides what to do next.
In case of an alarm, the behaviour of the voltage of the HVPSs is different from the actuation in case of a fault in the auxiliaries. As it has been shown in Table 6.1 of the Report, the time for action in case of a fault in the HVPSs is much smaller than time in case of a fault in one of the auxiliaries (except for the arc detector). For this reason, the voltage from the HVPSs should drop instantly and abruptly after receiving the trip signal. This has been explained as well when describing Figure 4.17 of the Report about the setting of the voltage slope depending on the state of each of the HVPSs: in normal conditions, the voltage slope to stop the HVPSs is -0.5 whereas for fast shutdown requests in case of alarm, the voltage slope is -10.

All this can be seen in the following figure.

As for the first simulation:

- The yellow curve represents the MPS output voltage
- The purple curve represents the MPS\_ON signal
- The blue curve represents the BPS output voltage
- The red curve represents the BPS\_ON signal
Appendix C

Automatic test of the model

This script enables the user doing an automatic test of the FALCONSimulator model of the FALCON control system. In order to make it work, the user has to enter a sequence of the different states of the Main State Machine of the Simulator through which he wants the system to evolve. The list has to be entered in the form of 'cell array' \{1, 2 ,3 ,...,\}.

The states that have to be used are only the main ones:

- ST = 0: OFF state
- ST = 1: SET1READY
- ST = 2: SET2READY
- ST = 3: SET3READY
- ST = 4: PULSEON

It is not necessary to include the transitory states of "StartingSetX" or "StoppingSetX".

Depending on the transition, the script will actuate on the corresponding block that will allow the transition from one state to another.

The user can also inject faults in the middle of the sequence as a string (see list of possible faults below). After the name of the fault, the user has to write in the sequence the number of the state where the state machine will move after the fault. These states are:

- ST = 30: FAULTSET1
- ST = 31: SET2NOTREADY (fault in a system from set 2)
- ST = 32: SET3NOTREADY (fault in a system from set 3)

The script has two outputs:

- 'List states': gives the list of main states through which the model has gone to follow the sequence.
- 'Match': is a variable that gives the result of the comparison between the given and the obtained sequence. If the sequences do not match, this variable will be ’0’ and if they do, it will be ’1’

List of possible faults:

David Soriano Baguet
• Auxiliaries:
  SET 1
  – IPPS (Ion Pump)
  – SCMCS (SCM Cryogenic system)
  SET 2
  – OT (Oil Tank)
  – SCMPs (Superconducting magnets)
  – CFPS (Filament)
  – Cooling system
    * BPCS (Low pressure cooling)
    * HPCS (High pressure cooling)
    * CollCS (Collector cooling)
    * LPCS (Long pulse dummy load cooling)
  SET 3
  – CDCPS (Collector DC)
  – CSWPS (Collector Sweeping)
  – GCPS (Gun Coil)

• HVPS:
  – Trip (Trip signal to both HVPSs)
  – SlowAlarmMPS
  – SlowAlarmBPS
  – SlowAlarmMPSInternal
  – SlowAlarmBPSInternal
  – FastAlarmMPSInternal
  – FastAlarmBPSInternal
  – UnderVMPS (Low voltage MPS)
  – UnderVBPS (Low voltage BPS)

function [List_states, Match] = Automatic_test(States)

close_all

% First of all, the parameters need to be loaded in the MATLAB
% workspace. This can be done running the script 'References.m'
run(’References.m’);

open(’FALCONSimulator.slx’); % to open the Simulink model
open(’Falcon_Control.fig’); % to open the Main HMI (makes the
% evolution more visual and easy to follow)

% The operation mode is normal operation
set_param(’FALCONSimulator/Operation_mode’,’Value’,’6’);
set_param(’FALCONSimulator/AuxiliaryPS/Operation_mode’,’Value’,’6’);

% Set all the main commands to ‘0’
set_param(’FALCONSimulator/StartSet1’,’Value’,’0’);
set_param(’FALCONSimulator/StartSet2’,’Value’,’0’);
set_param(’FALCONSimulator/StartSet3’,’Value’,’0’);
set_param(’FALCONSimulator/StopSet1’,’Value’,’0’);
set_param(’FALCONSimulator/StopSet2’,’Value’,’0’);
set_param(’FALCONSimulator/StopSet3’,’Value’,’0’);
set_param(’FALCONSimulator/Start_Pulse’,’Value’,’0’);
set_param(’FALCONSimulator/ResetMain’,’Value’,’0’);
set_param(’FALCONSimulator/ResetHVPS’,’Value’,’0’);
set_param(’FALCONSimulator/RESETMPS’,’Value’,’0’);
set_param(’FALCONSimulator/RESETBPS’,’Value’,’0’);
Appendix C. Automatic test of the model

% Put all possible Faults to '0' by default
%Auxiliaries
set_param('FALCONSimulator/AuxiliaryPS/IonPumpPS/Ext_Fault', 'Value', '0');
set_param('FALCONSimulator/AuxiliaryPS/SCMCompressor/Ext_Fault', 'Value', '0');
set_param('FALCONSimulator/AuxiliaryPS/OilTankSensor/Flowsensor', 'Value', '1');
set_param('FALCONSimulator/AuxiliaryPS/SCMPS/Critical', 'Value', '0');
set_param('FALCONSimulator/AuxiliaryPS/FilamentPS/Ext_Fault', 'Value', '0');
set_param('FALCONSimulator/AuxiliaryPS/CollectorDCPS/Ext_Fault', 'Value', '0');
set_param('FALCONSimulator/AuxiliaryPS/CollectorSweepPS/Ext_Fault', 'Value', '0');
set_param('FALCONSimulator/AuxiliaryPS/GunCoilPS/Ext_Fault', 'Value', '0');

%Components of the cooling system
set_param('FALCONSimulator/AuxiliaryPS/CoolingSystem/Cooling_Aux/Fault_BP', 'Value', '0');
set_param('FALCONSimulator/AuxiliaryPS/CoolingSystem/Cooling_Aux/Fault_HP', 'Value', '0');
set_param('FALCONSimulator/AuxiliaryPS/CoolingSystem/Cooling_Coll/Fault_Coll', 'Value', '0');
set_param('FALCONSimulator/AuxiliaryPS/CoolingSystem/Cooling_LP/Fault_LP', 'Value', '0');

%Faults in the HVPSs
set_param('FALCONSimulator/Trip_signal', 'Value', '0');
set_param('FALCONSimulator/SlowAlarmMPS', 'Value', '0');
set_param('FALCONSimulator/SlowAlarmBPS', 'Value', '0');
set_param('FALCONSimulator/HVPS/SlowAlarmMPSInternal', 'Value', '0');
set_param('FALCONSimulator/HVPS/SlowAlarmBPSInternal', 'Value', '0');
set_param('FALCONSimulator/HVPS/FastAlarmMPSInternal', 'Value', '0');
set_param('FALCONSimulator/HVPS/FastAlarmBPSInternal', 'Value', '0');
set_param('FALCONSimulator/HVPS/V_less_VBCLTTHR', 'Value', '0');
set_param('FALCONSimulator/HVPS/V_greater_VBCLTTHR', 'Value', '0');
set_param('FALCONSimulator/HVPS/V_less_VBCLTTHRBPS', 'Value', '0');
set_param('FALCONSimulator/HVPS/V_greater_VBCLTTHRBPS', 'Value', '0');
set_param('FALCONSimulator', 'StopTime', '30') %if the user wants to
% set a maximum simulation time
set_param('FALCONSimulator', 'SimulationCommand', 'start');

states_array = {}; % cell array that will store all the states the
% system goes through
for i=2:length(States) % as long as there are elements in the input
% cell array
% if there are two consecutive numerical states, it means that a
% command needs to be sent to progress
if (isnumeric(States{i}) && isnumeric(States{i-1}))
    MainState = get_param('FALCONSimulator/MainState', '.RuntimeObject'); %Main State Machine state
    MainState = MainState.InputPort(1).Data;
    states_array(end+1) = MainState; % store the current state
end

% if the system has to move forward (start a set): the
% StopSetX button is set to zero and the StartSetX is
% activated
if ((States{i}-States{i-1})==1) && (States{i}˜=4) || ...
    ((States{i-1})>=30) && (States{i}-States{i-1}+30>0)) % moving to next state (starting new set)
    set_param(strcat('FALCONSimulator/StopSet',num2str(States{i})), 'Value','0');
    set_param(strcat('FALCONSimulator/StartSet',num2str(States{i})), 'Value','1');
    Action = strcat('FALCONSimulator/StartSet',num2str(States{i}));
    Value = 1;
end

% if the system has to move from SET3READY to PULSEON (from
% state 3 to 4): the Start_Pulse command is set to '1' to
% start the pulse
elseif (States{i}-States{i-1})==1 && (States{i}==4)
    set_param('FALCONSimulator/Start_Pulse', 'Value','1');
    Action = 'FALCONSimulator/Start_Pulse';
    Value = 1;
end

% if the system has to move backward (stop a set): the
% StopSetX button is activated and the StartSetX is set
% to '0'
elseif ((States{i}-States{i-1})==-1) && (States{i}==3) || ...
    ((States{i-1})>=30) && (States{i}-States{i-1}+30==0)) % moving to previous step (stopping set)
    set_param(strcat('FALCONSimulator/StartSet',num2str(States{i}+1)), 'Value','0');
    set_param(strcat('FALCONSimulator/StopSet',num2str(States{i}+1)), 'Value','1');
    Action = strcat('FALCONSimulator/StopSet',num2str(States{i}+1));
    Value = 1;
end

% if the system has to move from PULSEON to SET3READY (from
% state 4 to 3): the Start_Pulse command is set back to '0'
% to stop the pulse
elseif (States{i}-States{i-1})==-1 && (States{i}==3)
    set_param('FALCONSimulator/Start_Pulse', 'Value','0');
    Action = 'FALCONSimulator/Start_Pulse';
end
Value = 0; % the Start_Pulse command will be set to '0'
end
pause(0.02) % the changes of parameters in the simulink
% environment need some time to be done

% As long as the simulator has not reached the requested
% state, the command will be repeated. The state is
% permanently updated in order to know when it reaches the
% right state
while MainState˜=States{i}
    set_param(‘Action’,’Value’,num2str(Value)); % repeat the
    action
    pause(0.02)
    MainState = get_param(‘FALCONSimulator/MainState’,’
    RuntimeObject’);
    MainState = MainState.InputPort(1).Data; % update the
    state
    states_array{end+1} = MainState; % add state to the array
end
% if the object in the array is a string (corresponding to a
% fault in any of the subsystems), the fault command has to
% be set.
elseif iscellstr(States{i})
    Value = 1; % the value of the fault block will be by default
    ’1’
    Fault_HVPS = 0; % indicates if the fault is in the HVPS. This
    % is important to know which systems need to be resetted
    % after solving the fault
    if strcmp(States{i},’IPPS’) % Ion Pump
        Fault = ‘FALCONSimulator/AuxiliaryPS/IonPumpPS/Ext_Fault’;
    elseif strcmp(States{i},’SCMCS’) % SCM Compressor
        Fault = ‘FALCONSimulator/AuxiliaryPS/SCMCompressor/
        Ext_Fault’;
    elseif strcmp(States{i},’OT’) % Oil Tank
        Fault = ‘FALCONSimulator/AuxiliaryPS/OilTankSensor/
        Flowsensor’;
        Value = 0; % the flow sensor works the opposite way: ’1’
        % means normal operation and ’0’ indicates a fault in the
        % flow sensor
    elseif strcmp(States{i},’SCMPS’) % SCM
        Fault = ‘FALCONSimulator/AuxiliaryPS/SCMPS/Critical’;
    elseif strcmp(States{i},’CFPS’) % Filament
        Fault = ‘FALCONSimulator/AuxiliaryPS/FilamentPS/Ext_Fault’;
    elseif strcmp(States{i},’CDCPS’) % Collector DC
        Fault = ‘FALCONSimulator/AuxiliaryPS/CollectorDCPS/
        Ext_Fault’;
    elseif strcmp(States{i},’CSWPS’) % Collector Sweeping
        Fault = ‘FALCONSimulator/AuxiliaryPS/CollectorSweepPS/
Appendix

else if strcmp(States{i}, 'GCPS') % Gun Coil
    Fault = 'FALCONSimulator/AuxiliaryPS/GunCoilPS/Ext_Fault';
end %Components of the cooling system
else if strcmp(States{i}, 'BPCS') % Low pressure auxiliaries
    Fault = 'FALCONSimulator/AuxiliaryPS/CoolingSystem/Cooling_Aux/Fault_BP';
else if strcmp(States{i}, 'HPCS') % High pressure auxiliaries
    Fault = 'FALCONSimulator/AuxiliaryPS/CoolingSystem/Cooling_Aux/Fault_HP';
else if strcmp(States{i}, 'CollCS') % Collector cooling
    Fault = 'FALCONSimulator/AuxiliaryPS/CoolingSystem/Cooling_Coll/Fault_Coll';
else if strcmp(States{i}, 'LPSC') % Long pulse cooling
    Fault = 'FALCONSimulator/AuxiliaryPS/CoolingSystem/Cooling_LP/Fault_LP';
end %Faults in the HVPSs. The Fault_HVPS will be set to '1' in
% this case
else if strcmp(States{i}, 'Trip') % Trip HVPSs
    Fault = 'FALCONSimulator/Trip_signal';
    Fault_HVPS = 1;
else if strcmp(States{i}, 'SlowAlarmMPS') % Slow External Alarm
    Fault = 'FALCONSimulator/SlowAlarmMPS';
    Fault_HVPS = 1;
else if strcmp(States{i}, 'SlowAlarmBPS') % Slow External Alarm
    Fault = 'FALCONSimulator/SlowAlarmBPS';
    Fault_HVPS = 1;
else if strcmp(States{i}, 'SlowAlarmMPSInternal') % Slow
    Fault = 'FALCONSimulator/HVPS/SlowAlarmMPSInternal';
    Fault_HVPS = 1;
else if strcmp(States{i}, 'SlowAlarmBPSInternal') % Slow
    Fault = 'FALCONSimulator/HVPS/SlowAlarmBPSInternal';
    Fault_HVPS = 1;
else if strcmp(States{i}, 'FastAlarmMPSInternal') % Fast
    Fault = 'FALCONSimulator/HVPS/FastAlarmMPSInternal';
    Fault_HVPS = 1;
else if strcmp(States{i}, 'FastAlarmBPSInternal') % Fast
    Fault = 'FALCONSimulator/HVPS/FastAlarmBPSInternal';
    Fault_HVPS = 1;
else if strcmp(States{i}, 'UnderVMPS') % Low voltage MPS
    Fault = 'FALCONSimulator/HVPS/V_less_VBCLTTHR';
Appendix C. Automatic test of the model

Fault_HVPS = 1;
else if strcmp(States{i}, 'UnderVBPS') % Low voltage BPS
    Fault = 'FALCONSimulator/HVPS/V_less_VBCLTTHRBPS';
    Fault_HVPS = 1;
end
states_array{end+1} = States{i}; % add to the sequence the % string corresponding to the inserted fault
set_param(Fault, 'Value', num2str(Value)); % To inject the % fault in the corresponding system
pause(0.02)

% if the element is numeric and the previous one was a string %
%(which means the element has to correspond to states 30, 31 % or 32)
elseif (isnumeric(States{i})) && (iscellstr(States{i-1}))
    MainState = get_param( 'FALCONSimulator/MainState', 'RuntimeObject'); %Main State Machine state
    MainState = MainState.InputPort(1).Data;
    % While the system does not reach the corresponding fault % state, continue updating the state
    while MainState~=States{i}
        pause(0.02)
        MainState = get_param( 'FALCONSimulator/MainState', 'RuntimeObject'); %Main State Machine state
        MainState = MainState.InputPort(1).Data;
    end
    states_array{end+1} = MainState; % store the fault state reached

% if the fault state is the last element in the list, the % fault is not cleared and the system is not resetted.
if i<length(States)
    % Once the system reached the fault state, the fault can % be cleared to continue
    set_param(Fault, 'Value', num2str(~Value));
    pause(0.02)
    % After clearing the fault, reset the corresponding % system

% If the fault is in the HVPSs, the alarm protocol needs % to be executed first (solved) before doing the reset
if Fault_HVPS==1
    set_param( 'FALCONSimulator/Manual Switch7', 'sw','1'); % MPS Alarm Slow Executed
    set_param( 'FALCONSimulator/Manual Switch8', 'sw','1'); % MPS Alarm Fast Executed
    set_param( 'FALCONSimulator/Manual Switch15', 'sw','1'); % BPS Alarm Slow Executed
    set_param( 'FALCONSimulator/Manual Switch16', 'sw','1'); % BPS Alarm Fast Executed

David Soriano Baguet
pause (.02)
set_param ( ’FALCONSimulator/ResetHVPS’ , ’Value’ , ’1’) % to reset the HVPPs Synchronization state machine
set_param ( ’FALCONSimulator/RESETMPS’ , ’Value’ , ’1’) % to reset the MPS state machine
set_param ( ’FALCONSimulator/RESETBPS’ , ’Value’ , ’1’) % to reset the BPS state machine
else % If fault in the auxiliaries, use the main reset
set_param ( ’FALCONSimulator/ResetMain’ , ’Value’ , ’1’);
end
pause (.02)
% The reset blocks are automatically set back to zero
% (thanks to the ’UpdateSim.m’ file (as are the
% Start/Stop sets blocks)
end
end

end

end
% if the sequence is finished, stop the simulation
set_param ( ’FALCONSimulator’ , ’SimulationCommand’ , ’stop’);
pause (.02)

%To remove consecutive repetitions of states and transitory states
j=2;
while j<=length ( states_array )
    if isnumeric ( states_array{j} )
        if (( states_array{j}>=10) && ( states_array{j}<30)) || ...
            ... ( isequal ( states_array{j} , states_array{j-1} ) )
            states_array{j} = [];
        else
            j=j+1;
        end
    else
        j=j+1;
    end
end
List_states = states_array;
Match = isequal ( states_array , States ); % ’1’ if they are equal, ’0’ if not
end