Agenzia nazionale per le nuove tecnologie, l'energia
e lo sviluppo economico sostenibile

Ministero dello Sviluppo Economico

RICERCA DI SISTEMA ELETTRICO

Sistema di simulazione di logiche di controllo per il miglioramento delle prestazioni e della sicurezza di impianti nucleari di nuova concezione

S. Di Gennaro, B. Castillo–Toledo, F. Memmi

RdS/2012/161

SISTEMA DI SIMULAZIONE DI LOGICHE DI CONTROLLO PER IL MIGLIORAMENTO DELLE PRESTAZIONI E DELLA SICUREZZA DI IMPIANTI NUCLEARI DI NUOVA CONCEZIONE

S. Di Gennaro, B. Castillo–Toledo, F. Memmi Università dell'Aquila

Settembre 2012

Report Ricerca di Sistema Elettrico
Accordo di Programma Ministero dello Sviluppo Economico - ENEA
Area: Governo, Gestione e Sviluppo**,** del Sistema Elettrico Nazionale
Progetto: Nuovo Nucleare da Fissione: Collaborazioni Internazionali e sviluppo Competenze in Materia Nucleare
Responsabile del Progetto: Massimo Sepielli,ENEA

CENTER OF EXCELLENCE DEWS

DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING

UNIVERSITY OF L'AQUILA, V. G. GRONCHI 18, 67100, L'AQUILA, ITALY

DELIVERABLE 2

Development of a Simulation Environment for the Analysis of Control System Performance for Performance and Safety Improvements of Novel Nuclear Plants

Sviluppo di un Sistema di Simulazione per l'Analisi delle Prestazione di Sistemi di Controllo per il Miglioramento della Prestazioni e della Sicurezza di Impianti Nucleari di Nuova Concezione

Authors:
Stefano DI GENNARO, Bernardino CASTILLO–TOLEDO, Fabrizio MEMMI

Principal Investigator:
Prof. Stefano DI GENNARO

Project PAR 2011

July 31, 2012

Abstract

In this deliverable, the digital controllers designed in [3] have been tested in the Simulink© simulation environment, to check their performance, and to verify if the behavior of the controlled system is correct and satisfies the control specifications, for various values of the sampling time. To better check the real behavior of the closed loop system, the controllers designed and tested on the basis of such a simulation environment need to be further checked inserting some hardware in the loop. In particular, it has been considered the implementation of the control law from real process measurements. It has been analyzed an appropriate real–time environment where is possible to acquire and store data, compute the control algorithm, and apply the control action to actuators. Even though not certified for nuclear applications, it has been analyzed the popular LabVIEW© as a potential solution to interface Simulink with real–time environments. This solution allows checking the methodological steps for the real–time prototyping of the controllers, and can be used for future non–nuclear industrial applications, while for nuclear applications more costly nuclear–certified softwares, ensuring the same real–time performances of LabVIEW, have to be considered. Finally, it will be presented an experimental set–up used to show the benefit of the FPGA features and that has to be integrated with the Simulink/LabVIEW simulation environment, in order to better validate the designed control laws.

Riassunto

In questo documento sono stati testati i controllori digitali progettati in [3] nell'ambiente di simulazione Simulink©, per controllare la loro prestazione e verificare se il comportamento del sistema controllato è corretto e soddisfa le specifiche di controllo, per vari valori del tempo di campionamento. Per meglio controllare il reale comportamento del sistema a ciclo chiuso, i controllori progettati e testati sulla base di tale ambiente di simulazione devono essere ulteriormente testati inserendo nell'anello dei dispositivi fisici. In particolare è stata considerata l'implementazione della legge di controllo a partire da misure di un processo reale. È stato analizzato un ambiente a tempo reale appropriato ove è possibile acquisire e immgazzinate dati, calcolare l'algoritmo di controllo, e applicare l'azione di controllo agli attuatori. Sebbene non certificato per applicazioni nucleari, è stato analizzato il popolare LabVIEW© come potenziale soluzione per interfacciare Simulink con ambienti a tempo–reale. Questa soluzione permette di controllare i passi metodologici per una prototipizzazione a tempo–reale dei controllori, e può essere usata per future applicazioni industriali non nucleari, mentre per applicazioni nucleari devono essere considerati più costosi programmi certificati in campo nucleare, che assicurino le stesse prestazioni in tempo reale di LabVIEW. Infine sarà presentato uno schema sperimentale, usato per mostrare i vantaggi delle caratteristiche degli FPGA e che deve essere integrato con l'ambiente di simulazione in Simulink/LabVIEW, per meglio validare le leggi di controllo progettate.

1 A Simulation Environment for Performance Evaluation of Digital Controllers

The digital controllers designed in [3] need to be tested in simulation environment to check their performance, to verify if the behavior of the controlled system is correct and if it satisfies the control specifications. This environment is Matlab© (Matrix Laboratory), also used in [1] to test the continuous time controllers. More specifically, the toolbox Simulink© has been used, which has the advantage of an easy graphical visualization.

1.1 Some Recalls on The Simulink Environment

There exists a variety of softwares able to simulate dynamical systems, both commercial and non commercial. One of the most popular choice for simulating control systems, also in the industrial context, is Matlab© (Matrix Laboratory) of Mathworks, and its toolbox Simulink©.

Simulink is an environment for multi–domain simulation and Model–Based Design for dynamic and embedded systems. Simulink provides an interactive graphical environment and a customizable set of block libraries that allow the design, simulation, implementation and test of a large number of systems arising in communication, control, signal, video and image processing, just to mention a few fields.

Simulink provides an extensive and expandable library of predefined blocks and a graphical editor for arranging these intuitive blocks into block diagrams. The user composes the block diagram of the system to be simulated by means of the interconnections among the elementary blocks, and Simulink automatically generates the implementation code.

Simulink is capable of interacting with Matlab, enabling full access to Matlab workspace for analyzing and visualizing results, customizing the modeling environment, as well as defining signal, parameter, and test data. Moreover, Matlab Function blocks fully exploit the powerful Matlab algorithms. These characteristics allow describing the behavior of complex dynamics thanks to control statements, cycles and other facilities, thus making the model design easier.

Additional key features are:

1. Model Explorer to navigate, create, configure, and search all signals,parameters, properties, and

generated code associated with the model;

2. Application Programming Interfaces (APIs) that allow the connection with other simulation programs and incorporate hand–written codes;

3. Simulation modes (Normal, Accelerator, and Rapid Accelerator) for running simulations interpretively or at compiled C–code speeds using fixed– or variable–step solvers;

4. Graphical debugger and profiler to examine simulation results and then diagnose performance and unexpected behavior in the model;

5. Model analysis and diagnostics tools to ensure model consistency and identify modeling errors.

Simulink provides a graphical user interface (GUI) for building models as block diagrams. The interactive graphical environment simplifies the modeling process, making the formulation of differential and difference equations dispensable.

In developing complex dynamical systems is often convenient, if not necessary, to split models into hierarchies of designed components, and make them communicate through input and output. Simulink models are hierarchical, thus being perfectly suited to such an approach. In this way, the models may be analyzed at different levels and according to their structural organization.

1.2 Implementation in Simulink of the Digital Controllers

A mathematical model of the primary circuit of a PWR has been implemented in Simulink in [1], where the reader can find the details of this implementation. In this section, the implementation of the digital controllers developed in [3] is described. As already discussed in [3], most of the modern controllers are realized by microprocessor–based digital circuits, or process–control computers. These devices can be characterized by discrete operations, where the control algorithms are implemented on a digital device.

The values of the sampled variables of the model have been used to implement the digital controllers for the pressurizer level and pressure. Figure 1 shows the Simulink diagram block representing the digital control system.

We first introduce and briefly describe the Simulink blocks used to build the control simulation system. Then, the single blocks composing the control system are analyzed and commented.

The library blocks used in the system are

1. Embedded Matlab Function: is the main library used in the model, it allows you to implement a MATLAB function (with input and output) in the Simulink environment.

2. Integrator: foundation library for the simulation of dynamic models. Returns the integral of the output signal which receives in input, at the time instant current. One can use variety of methods of numericalintegration for the calculation of the output.

3. Unit Delay: it delays its input by the specified sample period. This block is equivalent to the z-1 discrete-time operator. The block accepts one input and generates one output, which can be either both scalar or both vector. If the input is a vector, all elements of the vector are delayed by the same sample period.

4. Subsystem block: represents a subsystem of the system that contains it. The Subsystem block can represent a virtual subsystem or a nonvirtual subsystem.

5. Zero–order hold: this block samples and holds the input for the sample period you specify. The block accepts one input and generates one output. Each signal can be scalar or vector. If the input is a vector, the block holds all elements of the vector for the same sample period.

6. From: this block picks up the signal from the relative block ?Goto? and passes it on in output. Allows the passage of signals between blocks without connecting them.

7. Goto: transfers the input signal to the corresponding block From.

8. Input port: creates a port for subsystem or external inputs. It represents a link inside and outside of the system.

9. Output port: creates an output port for subsystem or external output. It represents a link between inside and outside the system.

10. Switch: this block performs a switching of out between the first and the third signal input places, via the directives of the control signal (2nd signal).

11. Signal to Workspace: writes the data obtained from simulations, within a structure in the main MATLAB workspace.

12. Scope: graphics the variable value at the time of simulation. Allows viewing multi-axis with respect to the same time range.

With respect to the simulation environment considered in [1], the scheme of figure 1 presents the following differences

- A digital inventory mass controller;

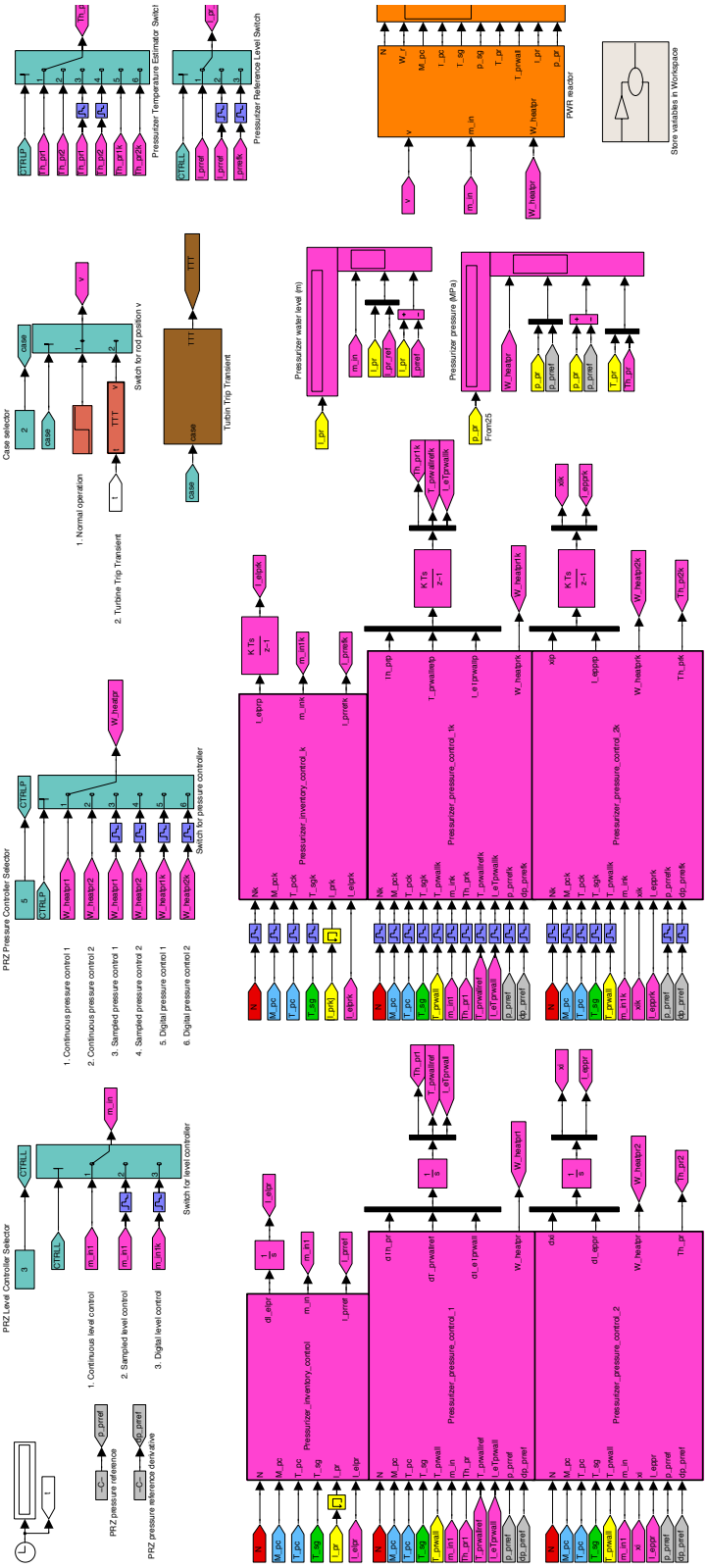

Figure 1: Digital control scheme for the primary circuit of a NPP

- Two digital pressurizer pressure controllers;
- Switches and displays.

An exhaustive description of the previous simulation environment is given in [1], which analyzes in detail the blocks corresponding to the PWR reactor dynamics, the continuous–time pressurizer water level control, the continuous–time pressurizer pressure controllers, the rod position control, and the various function performed by switches and displays visible in the scheme. Here we will describe the new blocks added to implement the digital controllers. Figure 5 shows the block `Pressurizer_inventory_control_k`, which represents the implementation of the digital inventory mass controller

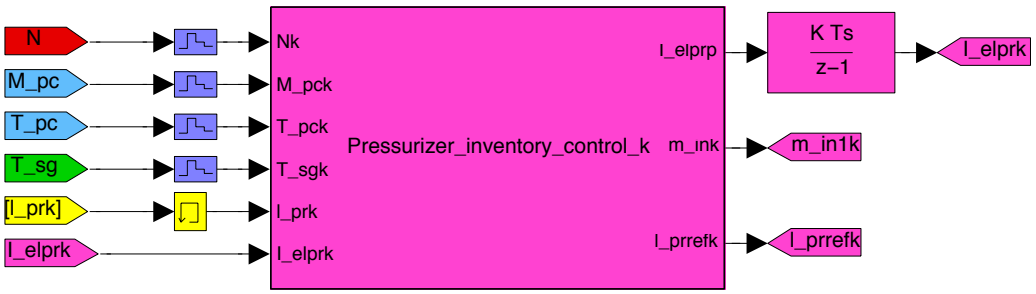

Figure 2: Pressurizer Discrete Inventory Mass Control

The corresponding EMF code is given in Table 1.

Table 1 – Digital Inventory Control (EMF code)

```
function [I_elprp,m_ink,l_prrefk]=Pressurizer_inventory_control_k(Nk,M_pck,
                                                    T_pck,T_sgk,l_prk,I_elprk)
%#eml
%-----------------------------------------------------------------------
% Initialization of the variables

% System parameters
Delta=0;
A_pr=0;
c_ppc=0;
c_psi=0;
c_phi0=0;
c_phi1=0;
c_phi2=0;
n_sg=0;
k_tsg=0;

% Reference parameters
c_r1=0;
c_r2=0;

% Perturbation parameters
m_out0=0;
T_pci0=0;
Delta0=0;
W_losspc0=0;

% Controller parameters
k_p=0;
k_i=0;
```

```
% Actuator parameter
minmax=0;

% Sampling period
delta=0;

%--------------------------------------------------------------------
% Loading current parameters from workspace
eml.extrinsic('evalin');
%eml.extrinsic('assignin');

% Load parameters from workspace
% System parameters
Delta=evalin('base','Delta');
A_pr=evalin('base','A_pr');
c_ppc=evalin('base','c_ppc');
c_psi=evalin('base','c_psi');
c_phi0=evalin('base','c_phi0');
c_phi1=evalin('base','c_phi1');
c_phi2=evalin('base','c_phi2');
n_sg=evalin('base','n_sg');
k_tsg=evalin('base','k_tsg');

% Reference parameters
c_r1=evalin('base','c_r1');
c_r2=evalin('base','c_r2');

% Perturbation parameters
m_out0=evalin('base','m_out0');
T_pci0=evalin('base','T_pci0');
Delta0=evalin('base','Delta0');
W_losspc0=evalin('base','W_losspc0');
```

```
% Controller parameters
k_p=evalin('base','k_p');
k_i=evalin('base','k_i');

% Actuator parameter
minmax=evalin('base','minmax');

% Sampling period
delta=evalin('base','delta');

%-----------------------------------------------------------------------
% Digital Pressurizer Level Control
% Cold and hot leg temperatures
T_pcclk=T_pck-Delta;
T_pchlk=T_pck+Delta;

% Level reference (see Pisa's report)
lprrefk=c_r1*(T_pcclk+T_pchlk)-c_r2;
if lprrefk<0
    l_prrefk=0;
else
    l_prrefk=lprrefk;
end

% Function \phi and its derivative
phik=c_phi0+c_phi1*T_pck-c_phi2*T_pck^2;
dphik=c_phi1-2*c_phi2*T_pck;

% \psi function
psik=phik-(T_pci0-T_pck)*dphik-2*c_r1*A_pr*(T_pci0-T_pck)*phik^2/M_pck;

% Integral term
I_elprp=l_prk-l_prrefk;
```

```
% Input m_{in}
mink=A_pr*((2*c_r1*phik^2/M_pck+dphik)*(c_ppc*m_out0*Delta0+c_psi*Nk
     -n_sg*k_tsg*(T_pck-T_sgk)-W_losspc0)/c_ppc-(k_p*(l_prk-l_prrefk)
     +k_i*I_elprk)*phik^2+m_out0*phik)/psik;
if mink<=0,
    m_ink=0;
elseif mink>=minmax;
    m_ink=minmax:
el
en
```

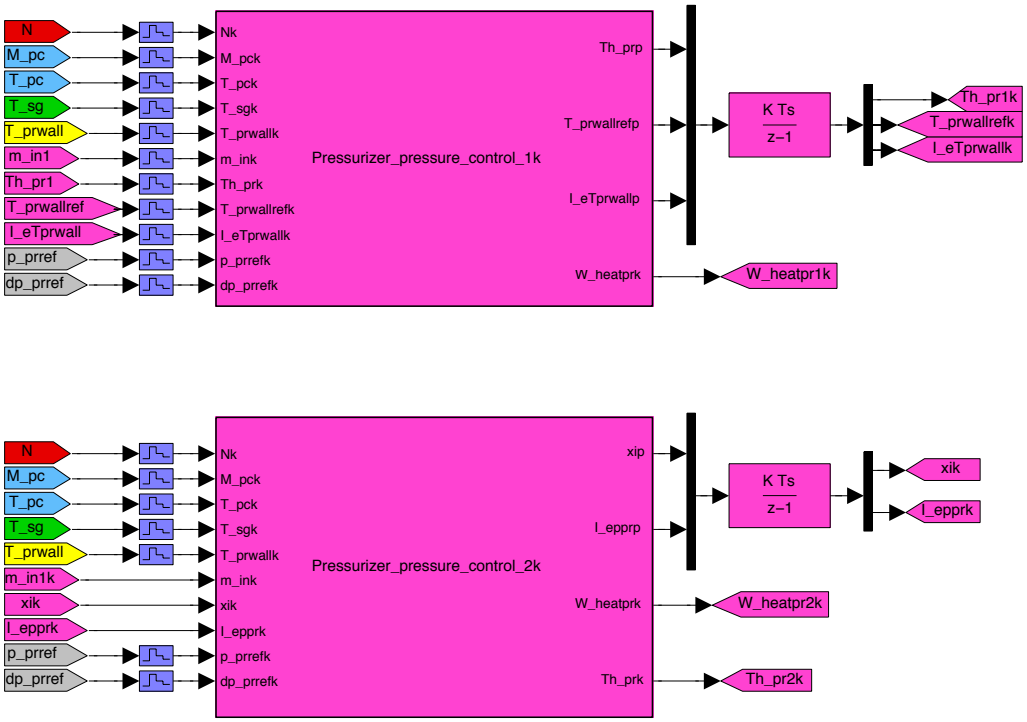

Figure 3: Digital pressurizer pressure controllers

In Figure 3 the `Pressurizer_pressure_controller_1k` and the `Pressurizer_pressure`

_controller_2k blocks are shown. These blocks simulate the implementation of the digital controllers

$$\hat{T}_{pr,k+1} = \hat{T}_{pr,k} + \delta\Bigg[-\left(\frac{m^{\circ}_{pr,k}}{M_{pr,k}} + \frac{k_{wall}}{c_{p,pr}M_{pr,k}}\right)\hat{T}_{pr,k} + \frac{k_{wall}}{c_{p,pr}M_{pr,k}}T_{pr,wall,k} + \frac{1}{c_{p,pr}M_{pr,k}}W_{heat,pr,k} + \frac{c_{p,pc}m^{\circ}_{pr,k}}{c_{p,pr}M_{pr,k}}(T_{pc,k} + \Delta^{\circ})\Bigg]$$

$$T_{pr,wall,\mathrm{ref},k+1} = T_{pr,wall,\mathrm{ref},k} + \delta\left(\frac{k_{wall}}{c_{p,wall}}T_{pr,\mathrm{ref},k} - \frac{k_{wall}}{c_{p,wall}}T_{pr,wall,\mathrm{ref},k} + k_i I_{e_{T_{pr,wall}},k} - \frac{1}{c_{p,wall}}W^{\circ}_{loss,pr}\right)$$

$$I_{e_{T_{pr,wall}},k+1} = I_{e_{T_{pr,wall}},k} + \delta\left(T_{pr,wall,k} - T_{pr,wall,\mathrm{ref},k}\right)$$

$$W_{heat,pr,k} = -k_{wall}(T_{pr,wall,k} - T_{pr,wall,\mathrm{ref},k}) - \frac{c_{p,pr}}{c_{p,wall}}k_{wall}M_{pr,k}B^T P\begin{pmatrix} I_{e_{T_{pr,wall}},k} \\ T_{pr,wall,k} - T_{pr,wall,\mathrm{ref},k}\end{pmatrix} + c_{p,pr}M_{pr,k}\Bigg[\dot{T}_{pr,\mathrm{ref}}\Big|_k + \left(\frac{m^{\circ}_{pr,k}}{M_{pr,k}} + \frac{k_{wall}}{c_{p,pr}M_{pr,k}}\right)T_{pr,\mathrm{ref},k} - \frac{k_{wall}}{c_{p,pr}M_{pr,k}}T_{pr,wall,\mathrm{ref},k} - \frac{c_{p,pc}m^{\circ}_{pr,k}}{c_{p,pr}M_{pr,k}}(T_{pc,k} + \Delta^{\circ})\Bigg]$$

$$T_{pr,\mathrm{ref},k} = \frac{c_1 + \sqrt{c_1^2 - 4c_2(c_0 - p_{pr,\mathrm{ref},k})}}{2c_2}, \qquad \dot{T}_{pr,\mathrm{ref}}\Big|_k = \frac{2\dot{p}_{pr,\mathrm{ref},k}}{\sqrt{c_1^2 - 4c_2(c_0 - p_{pr,\mathrm{ref},k})}}$$

and

$$I_{e_{p_{pr}},k+1} = I_{e_{p_{pr}},k} + \delta\left(c_0 - c_1\hat{T}_{pr} + c_2\hat{T}_{pr}^2 - p_{pr,\mathrm{ref}}\right)$$

$$\xi_{k+1} = \xi_k + \delta\left(\hat{T}_{pr} - T_{pr,wall} - \frac{1}{k_{wall}}W^{\circ}_{loss,pr} - \frac{1}{k}\frac{1}{c_{p,pr}M_{pr}}C_{pr}\right)$$

$$\hat{T}_{pr,k} = \kappa\left(\frac{c_{p,wall}}{k_{wall}}T_{pr,wall,k} - \xi_k\right)$$

$$C_{pr,k} = \frac{c_{p,pr}M_{pr,k}}{-c_1 + 2c_2\hat{T}_{pr,k}}\left(\dot{p}_{pr,\mathrm{ref},k} - K_p\left(c_0 - c_1\hat{T}_{pr,k} + c_2\hat{T}_{pr,k}^2 - p_{pr,\mathrm{ref},k}\right) - K_i I_{e_{p_{pr}},k}\right)$$

$$W_{heat,pr,k} = k_{wall}(\hat{T}_{pr,k} - T_{pr,wall,k}) + C_{pr,k} + \delta_{pr}\left(c_{p,pr}m^{\circ}_{pr,k}\hat{T}_{pr,k} - c_{p,pc}m^{\circ}_{pr,k}(T_{pc,k} + \Delta^{\circ})\right)$$

respectively. Their EMF codes are reported in Tables 2 and 3.

Table 2 – Digital pressurizer pressure control 1k (EMF code)

```
function [Th_prp,T_prwallrefp,I_eTprwallp,W_heatprk]
=Pressurizer_pressure_control_1k(Nk,M_pck,T_pck,T_sgk,T_prwallk,m_ink,Th_prk,
                                 T_prwallrefk,I_eTprwallk,p_prrefk,dp_prrefk)
%#eml
```

```
%-------------------------------------------------------------------------
%Inizialization of variables

% System parameters
c_ppr=0;
c_ppc=0;
c_psi=0;
c_phi0=0;
c_phi1=0;
c_phi2=0;
c0=0;
c1=0;
c2=0;
n_sg=0;
k_tsg=0;
c_pwall=0;
k_wall=0;
V_pc0=0;

% Perturbation parameters
W_losspr0=0;
W_losspc0=0;
m_out0=0;
T_pci0=0;
Delta0=0;

% Actuator parameter
Wheatmax=0;

% Controller parameter
k_di1=0;
Pd=zeros(2,2);
```

```
% Sampling period
delta=0;

%-------------------------------------------------------------------------
% Loading current parameters from workspace
eml.extrinsic('evalin');
%eml.extrinsic('assignin');

% Load parameters from workspace
% System parameters
c_ppr=evalin('base','c_ppr');
c_ppc=evalin('base','c_ppc');
c_psi=evalin('base','c_psi');
c_phi0=evalin('base','c_phi0');
c_phi1=evalin('base','c_phi1');
c_phi2=evalin('base','c_phi2');
c0=evalin('base','c0');
c1=evalin('base','c1');
c2=evalin('base','c2');
n_sg=evalin('base','n_sg');
k_tsg=evalin('base','k_tsg');
c_pwall=evalin('base','c_pwall');
k_wall=evalin('base','k_wall');
V_pc0=evalin('base','V_pc0');

% Perturbation parameters
W_losspr0=evalin('base','W_losspr0');
W_losspc0=evalin('base','W_losspc0');
m_out0=evalin('base','m_out0');
T_pci0=evalin('base','T_pci0');
Delta0=evalin('base','Delta0');

% Actuator parameter
```

```
Wheatmax=evalin('base','Wheatmax');

% Controller parameter
k_di1=evalin('base','k_di1');
Pd=evalin('base','Pd');

% Sampling period
delta=evalin('base','delta');

%-------------------------------------------------------------------------
% Digital Pressurizer Pressure Controller 1
T_prrefk=(c1+sqrt(c1^2-4*c2*(c0-p_prrefk)))/(2*c2);
dT_prrefk=2*dp_prrefk/(sqrt(c1^2-4*c2*(c0-p_prrefk)));
density_pck=c_phi0+c_phi1*T_pck-c_phi2*T_pck^2;
derdensity_pck=c_phi1-2*c_phi2*T_pck;
M_prk=M_pck-density_pck*V_pc0;
m_pr0k=m_ink-m_out0-derdensity_pck*V_pc0*(c_ppc*m_ink*(T_pci0-T_pck)
      +c_ppc*m_out0*Delta0+c_psi*Nk-n_sg*k_tsg*(T_pck-T_sgk)
      -W_losspc0)/(c_ppc*M_pck);
if m_pr0k>0,
    dprk=1;
else
    dprk=0;
end
W_heatprrefk=c_ppr*M_prk*dT_prrefk+k_wall*(T_prrefk-T_prwallrefk)
           -dprk*(c_ppc*m_pr0k*(T_pck+Delta0)-c_ppr*m_pr0k*T_prrefk);
BPdxk=[0 1]*Pd*[I_eTprwallk;T_prwallk-T_prwallrefk];
W_hk=-k_wall*(T_prwallk-T_prwallrefk)-c_ppr*k_wall*M_prk*BPdxk+W_heatprrefk;
if W_hk<0
    W_heatprk=0;
elseif W_hk>Wheatmax,
    W_heatprk=Wheatmax;
else
```

```
    W_heatprk=W_hk;
end
Th_prp=(-k_wall*(Th_prk-T_prwallk)+W_heatprk+dprk*(c_ppc*m_pr0k*(T_pck+Delta0)
       -c_ppr*m_pr0k*Th_prk))/(c_ppr*M_prk);
T_prwallrefp=k_wall*(T_prrefk-T_prwallrefk)/c_pwall+k_di1*I_eTprwallk
       -W_losspr0/c_pwall;
I_eTprwallp=T_prwallk-T_prwallrefk;
```

Table 2 – Digital pressurizer pressure control 1k (EMF code)

Table 3 – Digital pressurizer pressure control 2k (EMF code)

```
function [xip,I_epprp,W_heatprk,Th_prk]=Pressurizer_pressure_control_2k(Nk,
          M_pck,T_pck,T_sgk,T_prwallk,m_ink,xik,I_epprk,p_prrefk,dp_prrefk)
%#eml
%-------------------------------------------------------------------------
%Inizialization of variables

% System parameters
c_ppr=0;
c_ppc=0;
c_psi=0;
c_phi0=0;
c_phi1=0;
c_phi2=0;
c0=0;
c1=0;
c2=0;
n_sg=0;
k_tsg=0;
c_pwall=0;
k_wall=0;
```

```
V_pc0=0;

% Perturbation parameters
W_losspr0=0;
W_losspc0=0;
m_out0=0;
T_pci0=0;
Delta0=0;

% Controller parameters
Kdp=0;
Kdi=0;
kd=0;

% Actuator parameter
Wheatmax=0;

% Sampling period
delta=0;

%--------------------------------------------------------------------------
% Loading current parameters from workspace
eml.extrinsic('evalin');
%eml.extrinsic('assignin');

% Load parameters from workspace
% System parameters
c_ppr=evalin('base','c_ppr');
c_ppc=evalin('base','c_ppc');
c_psi=evalin('base','c_psi');
c_phi0=evalin('base','c_phi0');
c_phi1=evalin('base','c_phi1');
c_phi2=evalin('base','c_phi2');
```

```
c0=evalin('base','c0');
c1=evalin('base','c1');
c2=evalin('base','c2');
n_sg=evalin('base','n_sg');
k_tsg=evalin('base','k_tsg');
c_pwall=evalin('base','c_pwall');
k_wall=evalin('base','k_wall');
V_pc0=evalin('base','V_pc0');

% Perturbation parameters
W_losspr0=evalin('base','W_losspr0');
W_losspc0=evalin('base','W_losspc0');
m_out0=evalin('base','m_out0');
T_pci0=evalin('base','T_pci0');
Delta0=evalin('base','Delta0');

% Controller parameters
Kdp=evalin('base','Kdp');
Kdi=evalin('base','Kdi');
kd=evalin('base','kd');

% Actuator parameter
Wheatmax=evalin('base','Wheatmax');

% Sampling period
delta=evalin('base','delta');

%-------------------------------------------------------------------------
% Digital Pressurizer Pressure Controller 2
density_pck=c_phi0+c_phi1*T_pck-c_phi2*T_pck^2;
derdensity_pck=c_phi1-2*c_phi2*T_pck;
M_prk=M_pck-density_pck*V_pc0;
m_pr0k=m_ink-m_out0-derdensity_pck*V_pc0*(c_ppc*m_ink*(T_pci0-T_pck)
```

```
       +c_ppc*m_out0*Delta0+c_psi*Nk-n_sg*k_tsg*(T_pck-T_sgk)
       -W_losspc0)/(c_ppc*M_pck);
if m_pr0k>0,
    dprk=1;
else
    dprk=0;
end
Th_prk=kd*(c_pwall*T_prwallk/k_wall-xik);
ph_prk=c0-c1*Th_prk+c2*Th_prk^2;
Dhk=-c1+2*c2*Th_prk;
Cprk=c_ppr*M_prk*(dp_prrefk-Kdp*(ph_prk-p_prrefk)-Kdi*I_epprk)/Dhk;
xip=Th_prk-T_prwallk-W_losspr0/k_wall-Cprk/(kd*c_ppr*M_prk);
I_epprp=ph_prk-p_prrefk;
W_hk=k_wall*(Th_prk-T_prwallk)+Cprk+dprk*(c_ppr*m_pr0k*Th_prk
     -c_ppc*m_pr0k*(T_pck+Delta0));
if W_hk<0
    W_heatprk=0;
elseif W_hk>Wheatmax,
    W_heatprk=Wheatmax;
else
    W_heatprk=W_hk;
end
```

Table 3 – Digital pressurizer pressure control 2k (EMF code)

Numerical and analogical displays show the behavior of the controlled variables, make possible their check during the simulation, as long as further variables of interest, such as the tracking errors.

Switches allow the selection of the desired control law. They have various possible choices, for the various possible control laws (continuous–time, sampled, digital).

1.3 Simulations Results

In this section, the results obtained in the simulations of the model will be presented and discussed. The Simulink scheme, in Figure 1, allows varying several simulation conditions, such as the operation

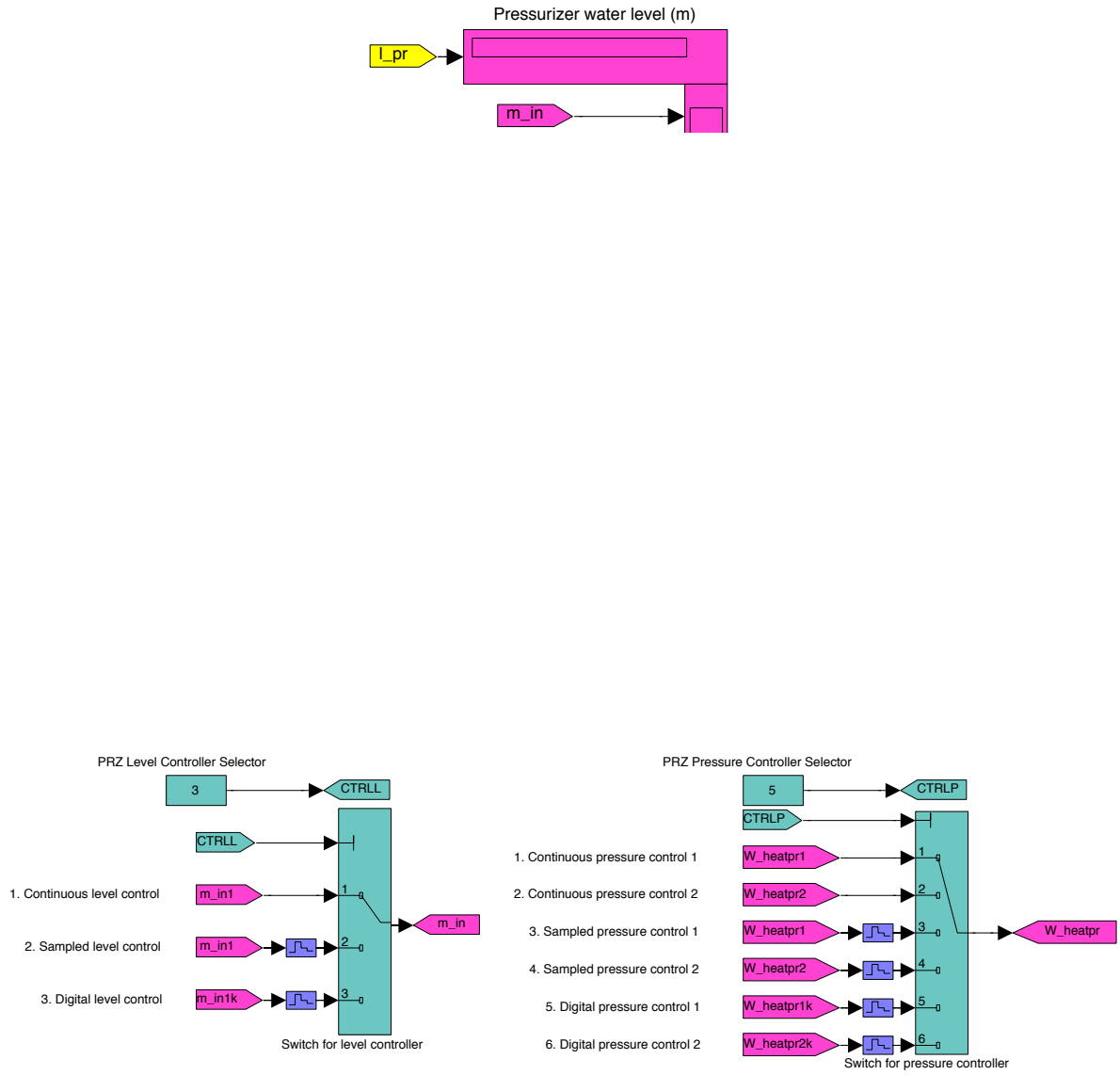

Figure 5: Switches for control selection

condition of the plant (normal conditions, turbine trip transient), the pressurizer inventory and pressure controllers.

The parameters influencing the controller's performance, such as the sampling time for the sampled and the digital controllers, or those describing the turbine trip transient, can be changed modifying the values in the initialization data file (see Table 4).

For the sake of conciseness, we report here the most interesting case, namely that corresponding to the digital inventory control `Pressurizer_inventory_control_k` and the first digital pressure controller `Pressurizer_pressure_controller_1k` during a stop valve fault, and the consequent turbine trip transient. In fact, this event allows checking whether the digital control laws are robust with respect

to faults. The simulation study have been conducted considering growing sampling times, as further test of robustness of the control laws with respect to delays. Indeed, the sampling time is one of the most important factor when dealing with implementations on digital electronics devices. The performance of these devices are growing quickly, but are still limited in speed. Finally, in the simulations parameter perturbations have been considered in order to achieve more realistic conditions.

We consider a turbine trip due to faulty closure of the turbine stop valve. During normal operation, the main steam flows from the steam generators through parallel pipes to a header, from where the steam is led to main steam stop and control valves by individual pipes of the high pressure turbine. Branch lines provide the possibility to bypass the turbine during transient operations. In normal conditions, the bypass station remains closed and the steam passes through the main stop and control valves and expands in the high pressure turbine.

The main steam stop valves have dual function. They isolate the turbine from the main steam line or from the steam generator. They rapidly interrupt the supply of steam to the turbine after being triggered by monitors if a dangerous condition arises. Therefore they have been designed for quick closing and maximum reliability. The control valves, on the other hand, regulate the flow of steam to the turbine according to the prevailing and provide a second means of isolation for the turbine in case of emergency. The control valve is operated by the piston of the servo–motor which is subjected to the spring force in the closing direction and the pressure of the control fluid in the opening direction. The position of the valve is determined by the secondary fluid pressure which is controlled by the governor.

In case of undue operating conditions within the turbine–generator plant the turbine trip system is released by means of protective devices for turbine and generator (turbine protection system). Hereby the main steam stop and control valves are closed. The steam produced in the steam generators is bypassed via bypass stop and control valves and dumped into the condenser. When turbine trip is initiated, the pressure drop in the trip oil circuit also causes the secondary fluid pressure to collapse because it is fed from it. The result is that both stop valve and the control valve close rapidly. The time for closing the stop valve is about 150 ms and for the control valve about 200 ms. In events like excessive load reductions, load rejection or turbine trip, the main steam maximum pressure limitation opens valves in the main steam bypass station and the main steam is passed into the condenser. The main steam pressure in the header is used as actual value for the control. The set–point is a few bars above the main steam operating pressure. Main steam relief station may also be used for controlling the main steam pressure.

Besides manual trip or spurious actuation, turbine trip initiation may be caused by steam turbine protection system components, like overspeed protection, overspeed trip selection, high condenser pressure protection, thrust bearing trip, low lube oil pressure trip, fire protection, main steam minimum pressure

signalin, electrical or mechanical generator protection.

After the turbine stop valves have closed, main steam pressure increases challenging the steam generator secondary side heat removal capability. Coolant temperature and pressure will increase and, unless adequate countermeasures are timely provided, heat removal from the core may be challenged too.

On turbine trip, after the turbine stop valves have closed, secondary side heat removal is abruptly interrupted leading to a sudden increase in the main steam pressure. However, the pressure excursion is rather limited by the prompt response of the turbine bypass station. The main steam bypass valves open immediately because the MS pressure goes above the maximum pressure set–point, which is reduced on turbine trip and then raised to a maximum at which it is held. In the first 10 s after the turbine trip, degraded heat removal conditions in the secondary side of the steam generators cause an increase in the coolant average temperature and a consequent expansion of coolant volume. Because of the high reactor minimum load, the overall temperature changes in coolant and moderator are rather small. Consequently, the volumetric changes are also limited, as can be seen in the behavior of the pressurizer water level. Closed–loop control and limitation systems are called upon to deal with the effects of power mismatch between reactor and the electric generator, keeping process variables within acceptable limits. Partial rod dropping is initiated by comparing reactor and generator power and, after 1.1 s delay time, the reactor is run back to a minimum load of approximately 80%, as a consequence of rod movement. A minor transient is observed in the coolant pressure which demands the intervention of pressurizer heating power. With the available main steam bypass system, the secondary relief station stays closed. Due to the main steam pressure rise, the steam generator water level initially slightly decreases and it is brought back to normal by the main feed–water control. Due to the effective response of control and limitation systems, promptly reducing the reactor power and early opening of the main steam bypass station, the process variables differ only insignificantly from their set–points, keeping reasonable margins to the limits of the reactor protection system, which are not reached.

Figures 6–36 show the dynamics associated to the principal variables describing the primary circuit dynamics, the reference values of the controlled variables and output, as well as the controlled input. As already commented, the simulations have been carried out for different values of the sampling time: $\delta = 0.0001$ s, $\delta = 0.001$ s, $\delta = 0.01$ s, $\delta = 0.1$ s, and $\delta = 1$ s. The simulation results that follow will show the satisfactory behavior, although for $\delta = 1$ the control input starts to be too active.

Table 4 – Initialization data file

```
clear all, clc
disp('Loading simulation data ...')
```

```
disp('(see help for details)')
disp(' ')

%=========================================================================
% Reactor parameters
%-------------------------------------------------------------------------
Lambda=1e-5;          % generation time; s
S=2830.05;            % flux of the constant neutron source; %/s
p0=2.85e-4;           % rod reactivity coefficients; m
p1=6.08e-5;           %                              m^(-1)
p2=1.322e-4;          %                              m^(-2)

%-------------------------------------------------------------------------
% Primary Ciruit parameters
%--------------------------
c_ppc=5355;           % specific heat at 280 C; J/(kg*K)
c_psi=13.75e6;        % power reactor constant; W/%
n_sg=6;               % number of steam generatots in Paks Nuclear Power Plant

% Perturbations
m_out0=2.11;          % nominal outlet mass flow rate; kg/s
m_out=2.0678;         % real outlet mass flow rate: -2% of m_out0; kg/s
T_pci0=258.85;        % nominal inlet temperature; C
T_pci=256.2615;       % real inlet temperature: -1% of T_pci0; C
W_losspc0=2.996e7;    % nominal heat loss; J/s
W_losspc=3.07976e7;   % real heat loss: +3% of W_losspc0; J/s
Delta0=15;            % nominal difference between T_pc and T_pc,cl; C
Delta=15.6;           % real difference between T_pc and T_pc,cl: +4% of Delta0; C

%-------------------------------------------------------------------------
% Steam Generator parameters
%--------------------------
m_sg=119.31;          % inlet secondary water mass flow rate = outlet secondary
```

```
                                                    steam mass flow rate; kg
c_psgl=3809.9;       % second. circuit liquid water specific heat at 260 C;
                                                                  J/(kg K)
c_psgv=3635.6;       % second. circuit  steam water specific heat at 260 C;
                                                                  J/(kg K)
T_sgsw= 220.85;      % second. circuit inlet temperature; C
E_evapsg=1.658e6;    % evaporation energy at 260 C; J/kg
k_tsg=9.5296e6;      % steam generator heat transfer coefficient; J/(K s)
M_sg=34920;          % water mass; kg

% Perturbations
W_losssg0=1.8932e7;  % nominal heat loss; J/s
W_losssg=1.9689e7;   % real heat loss: +4% of W_losssg0; J/s

%-------------------------------------------------------------------------
% Pressurizer parameters
%---------------------------
c_ppr=6873.1;        % specific heat of the water; J/(kg*K)
V_pc0=242;           % water nominal volume; m^3
c_phi0=581.2;        % coefficients of the density quadratic function; []
c_phi1=2.98;
c_phi2=0.00848;
c0=28884.78;         % coefficients of the saturated vapor; kPa
c1=258.01;           %                                     kPa/C
c2=0.63455;          %                                     kPa/C^2
A_pr=4.52;           % vessel cross section; m^2
k_wall=1.9267e8;     % wall heat transfer coefficient; W/C
c_pwall=6.4516e7;    % wall heat capacity; J/C

% Perturbations
W_losspr0=1.6823e5;  % nominal heat loss; J/s
W_losspr=1.7159e5;   % real heat loss: +2% of W_losspr; J/s
```

```
%------------------------------------------------------------------------
% Nominal inputs
%---------------------------
v0=0;                % input: nominal rod position; cm
m_in0=2.11;          % input: nominal inlet mass flow rate; kg/s
W_heatpr0=168000;    % input: nominal heating power; W

%------------------------------------------------------------------------
% Steady state conditions
%---------------------------
% Reactor initial Condition
N0=Lambda*S/p0;      % =99.3 The neutron flux N is measured in percent

% Primary Ciruit initial conditions
M_pc0=2e5;           % water mass in the primary circuit; kg

% Primary circuit and Steam generator initial conditions
A=[c_ppc*m_in0+n_sg*k_tsg   -n_sg*k_tsg;
    -k_tsg             m_sg*c_psgv+k_tsg];
B=[c_ppc*m_in0*T_pci+c_ppc*m_out*Delta+c_psi*N0-W_losspc;
   m_sg*(c_psgl*T_sgsw-E_evapsg)-W_losssg];
C=inv(A)*B;
T_pc0=C(1,1);
T_sg0=C(2,1);
clear A B C

% Pressurizer initial conditions
T_pr0=326.51;                      % C
T_prwall0=T_pr0-W_losspr/k_wall;

%------------------------------------------------------------------------
% Pressurizer water level reference parameters
%---------------------------
```

```
% pressurizer water level at nominal conditions
l_pr0=(M_pc0/(c_phi0+c_phi1*T_pc0-c_phi2*T_pc0^2)-V_pc0)/A_pr;

c_r1=0.093;                        % m/C
c_r2=2*c_r1*T_pc0-l_pr0;           % m

%-------------------------------------------------------------------------
% Turbine trip transient (TTT)
%---------------------------
tTTT=2;               % instant of occurrence of the turbine trip transient; s
DeltaTTTN=1.1;        % delay for reducing reactor power; s

a=p2;                % determination of rod position correspondig to
b=p1;                % N= 80% of N0
c=p0-Lambda*S/(N0*0.80);
vTTT=(-b+sqrt(b^2-4*a*c))/(2*a);
clear a b c

%-------------------------------------------------------------------------
% Pressurizer water level controller parameters
%---------------------------
k_p=100;
k_i=50;

% Initial condition (integral action)
I_elpr0=0;

% Actuator parameter (saturation)
minmax=20;                                % kg/s

%-------------------------------------------------------------------------
% Pressurizer pressure controller
%---------------------------
```

```
% Pressure reference and derivative
p_prref=12300;                          % kPa
dp_prref=0;

% Actuator parameter (saturation)
Wheatmax=3.6e6;                         % W

% Temperature observer initial conditions
Th_pr0=324;                             % C

%-------------------------------------------------------------------------
% Continuous controller 1
%--------------------------
% Initial conditions
T_prref0=326.51;                        % C
T_prwallref0=T_prref0-W_losspr0/k_wall; % C
I_eTprwall0=0;                          % C s (integral action)

% Controller parameter
k_i1=200;                               % Integral gain

% Lyapunov matrix equation
q1=1e-5;
q2=1e-10;
Q=[q1 0; 0 q2];
A=[0 1; -k_i1 -k_wall/c_pwall];
P=lyap(A',Q);  % P=lyap(A',Q) solves the Lyapunov matrix equation: P*A + A'*P = -Q
clear A Q

%-------------------------------------------------------------------------
% Continuous controller 2
%--------------------------
% Parameters
```

```
zeta=0.707;                              % damping
wn=3e3;                                  % natural frequency
Kp=2*zeta*wn;
Ki=wn^2;

% Observer gain
k=20;

% Integrator initial conditions
xi0=-Th_pr0/k+c_pwall*T_prwall0/k_wall;
I_eppr0=0;

%--------------------------------------------------------------------------
% Digital Controllers
%--------------------------
% Sampling time
delta=.1;                                % s

%--------------------------------------------------------------------------
% Digital controller 1
%--------------------------
% Initial conditions as for the continuous controller

% Controller parameter
k_di1=k_i1;                             % Integral gain

% Lyapunov matrix equation for the digital case
qd1=q1;
qd2=q2;
Qd=[qd1 0; 0 qd2];
Ad=[0 1; -k_di1 -k_wall/c_pwall];
Pd=lyap(Ad',Qd);  % Pd=lyap(Ad',Qd) solves the Lyapunov matrix equation:
                                                    Pd*Ad + Ad'*Pd = -Qd
```

```
clear Ad Qd

%------------------------------------------------------------------------
% Digital controller 2
%--------------------------
% Initial conditions as for the continuous controller

% Parameters
zetad=0.707;                            % damping
wnd=50;                                 % natural frequency
Kdp=2*zetad*wnd;
Kdi=wnd^2;

% Observer gain
kd=20;

%========================================================================
disp('... data loaded!')
disp('Starting simulation.')
```

Table 4 – Initialization data file

1.3.1 *Digital Inventory* `Pressurizer_inventory_controller_1k` *and Pressure Control* `Pressurizer_pressure_controller_1k` *with Sampling Time* $\delta = 10^{-3}$ *s*

The simulation results are summarized in Figures 6–12.

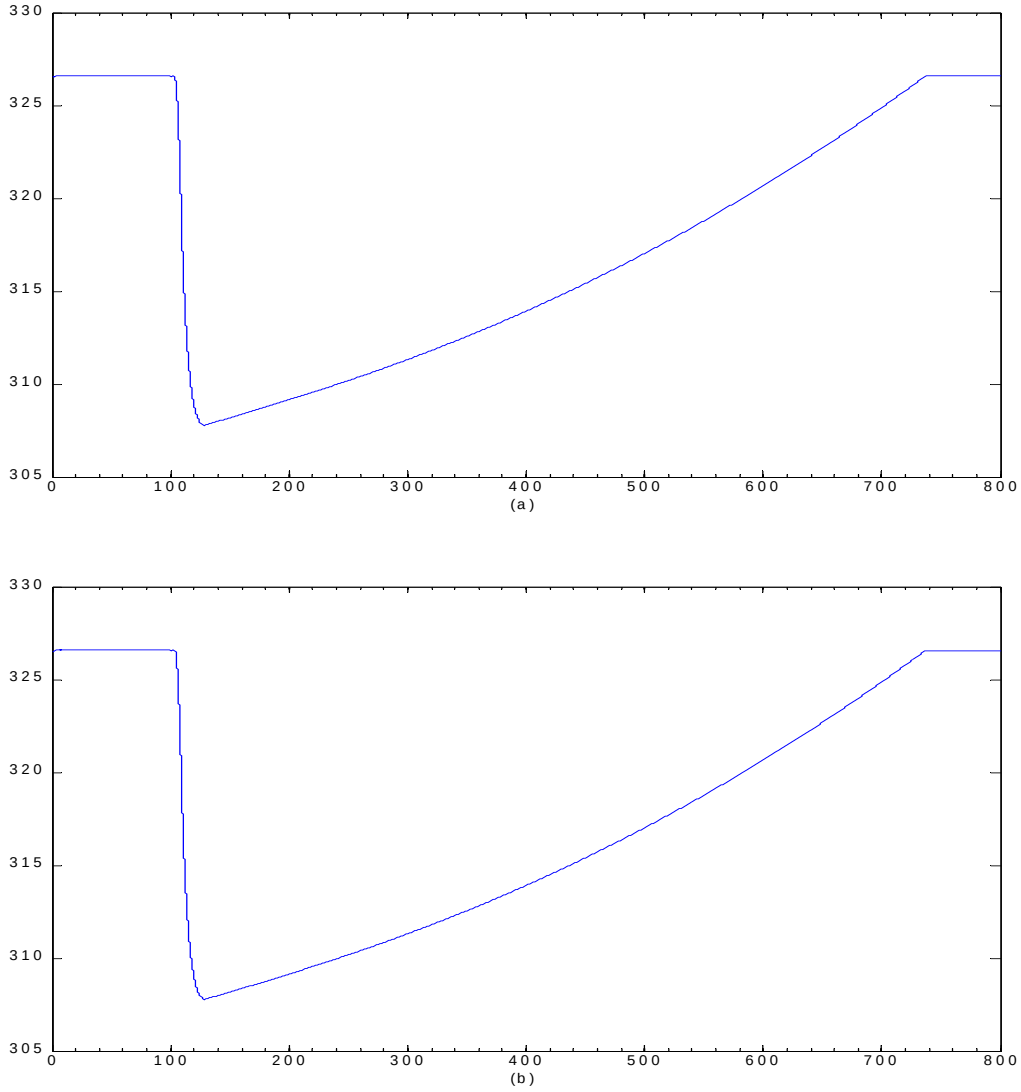

Figure 6: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-3}$ s. (a) Pressurizer temperature [°C]; (b) Pressurizer wall temperature [°C]

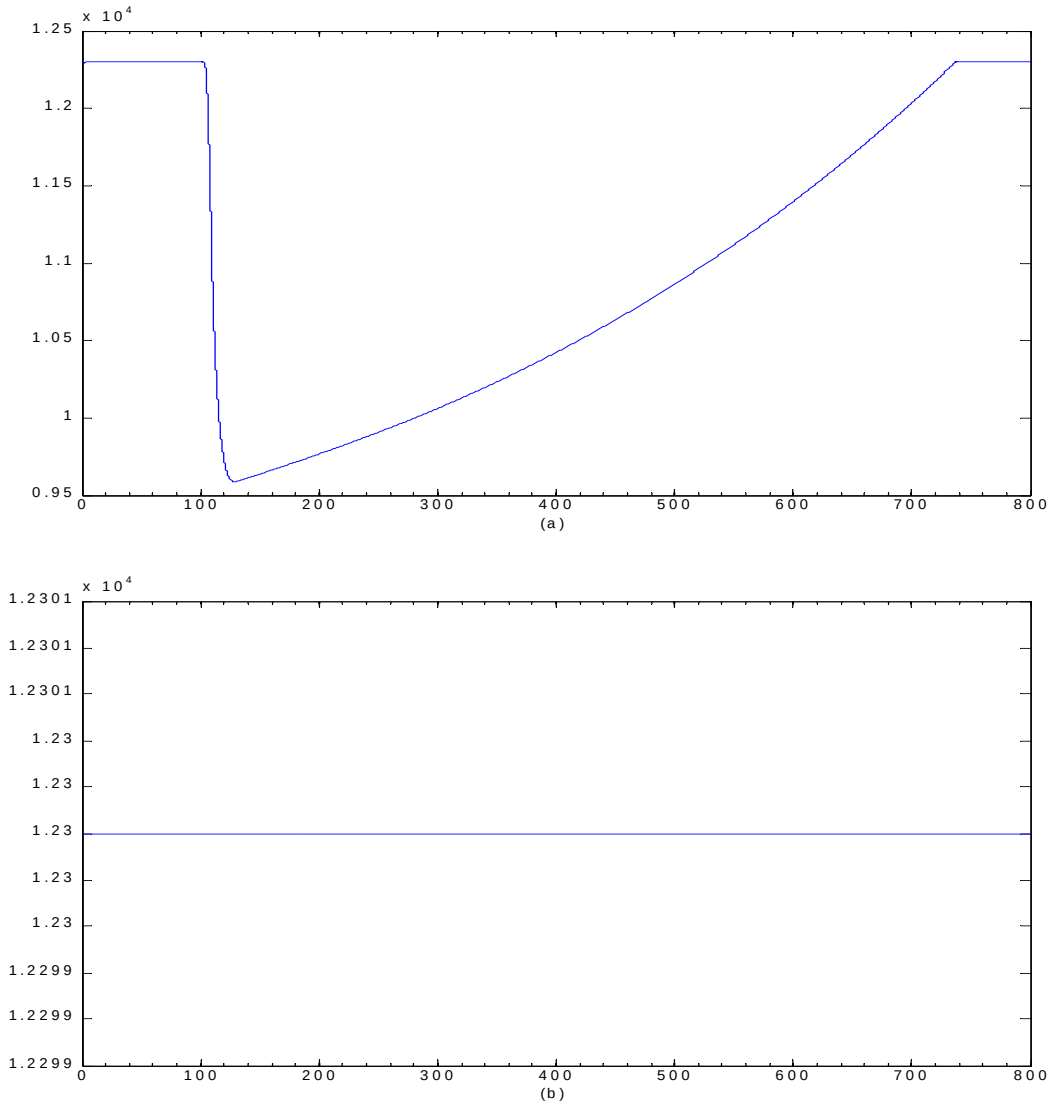

Figure 7: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-3}$ s. (a) Pressurizer pressure [Pa]; (b) Pressurizer pressure reference [Pa]

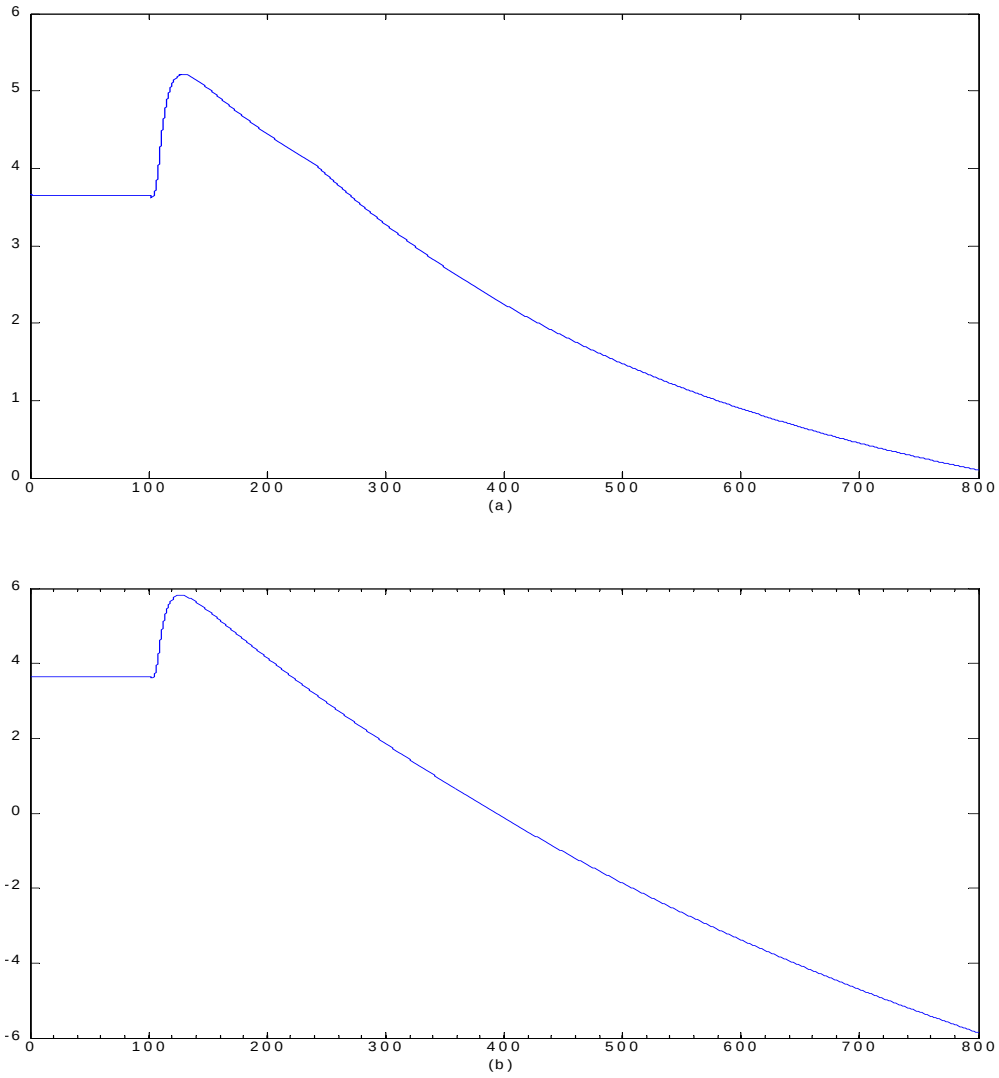

Figure 8: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-3}$ s. (a) Pressurizer water level [m]; (b) Pressurizer water level reference [m]

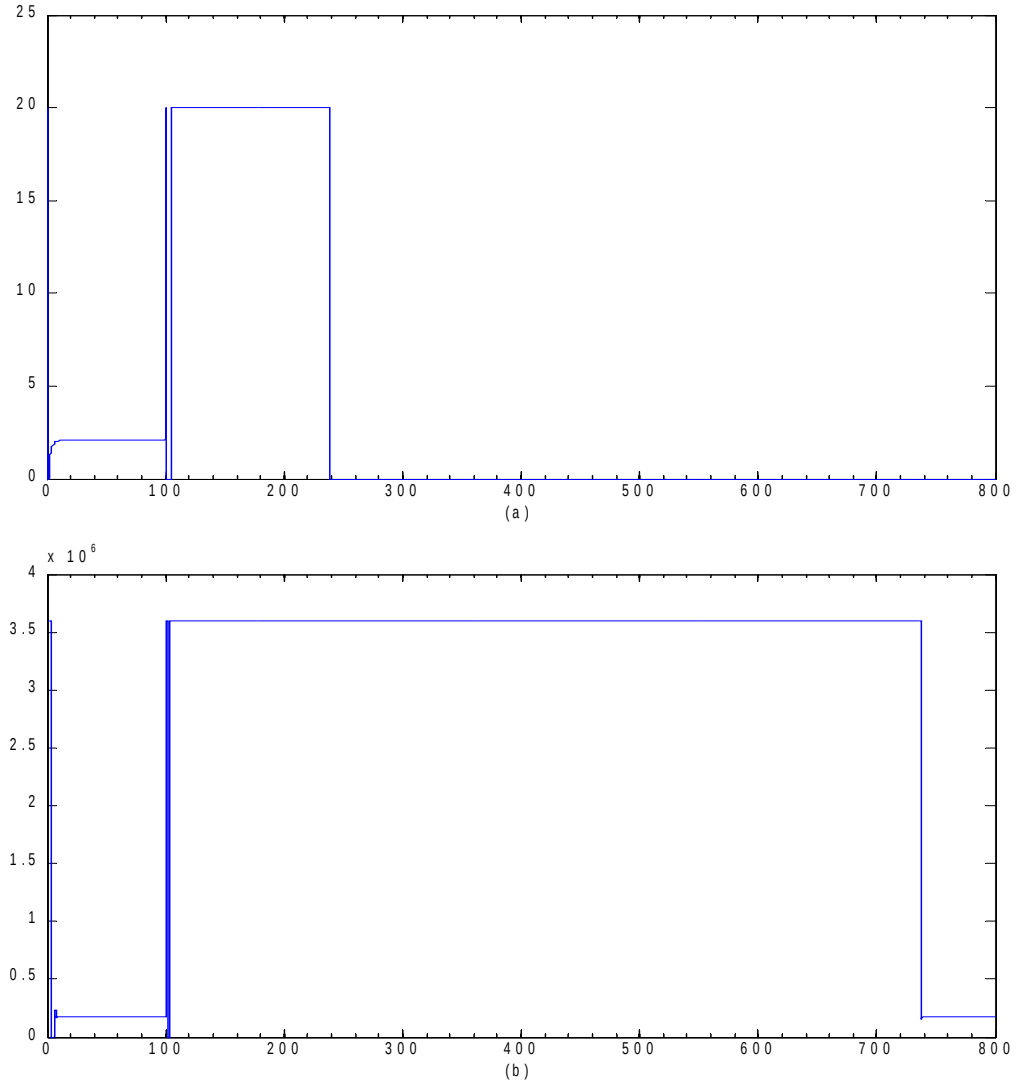

Figure 9: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-3}$ s. (a) Inlet mass flow rate [kg/s]; (b) Pressurizer heating power [W]

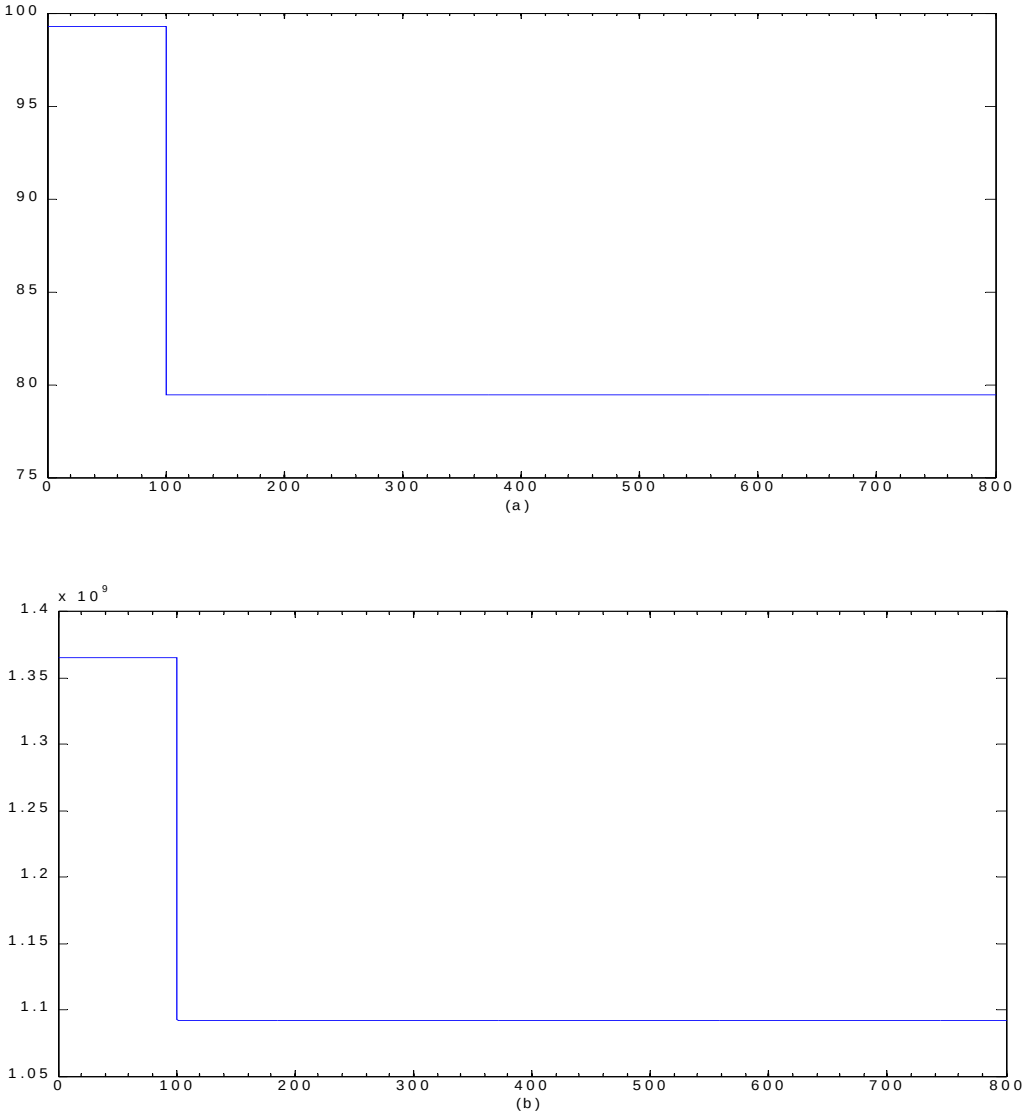

Figure 10: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-3}$ s. (a) Neutron flux [%]; (b) Reactor power [W]

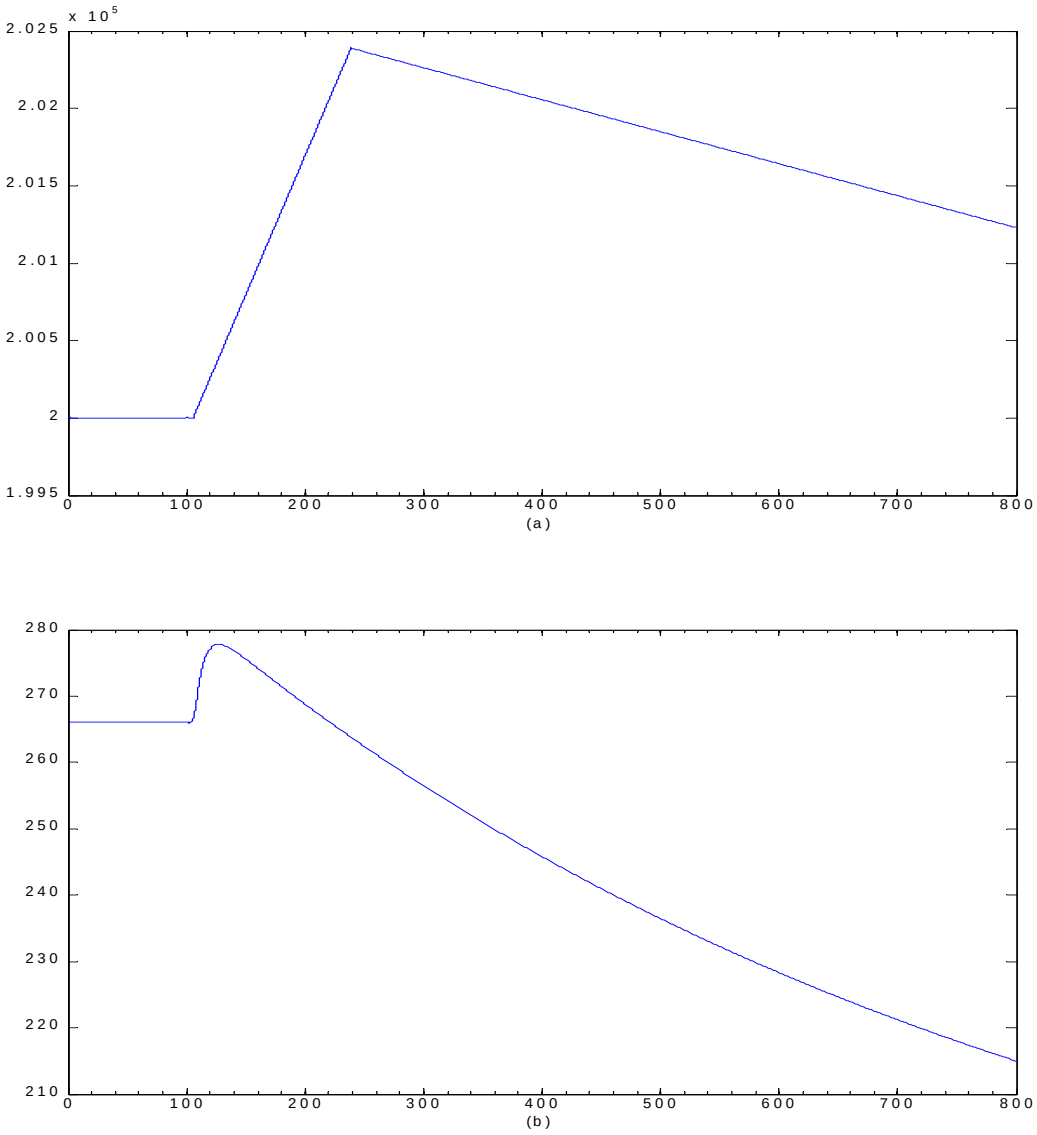

Figure 11: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-3}$ s. (a) Primary circuit water mass [kg]; (b) Primary circuit avarege temperature [°C]

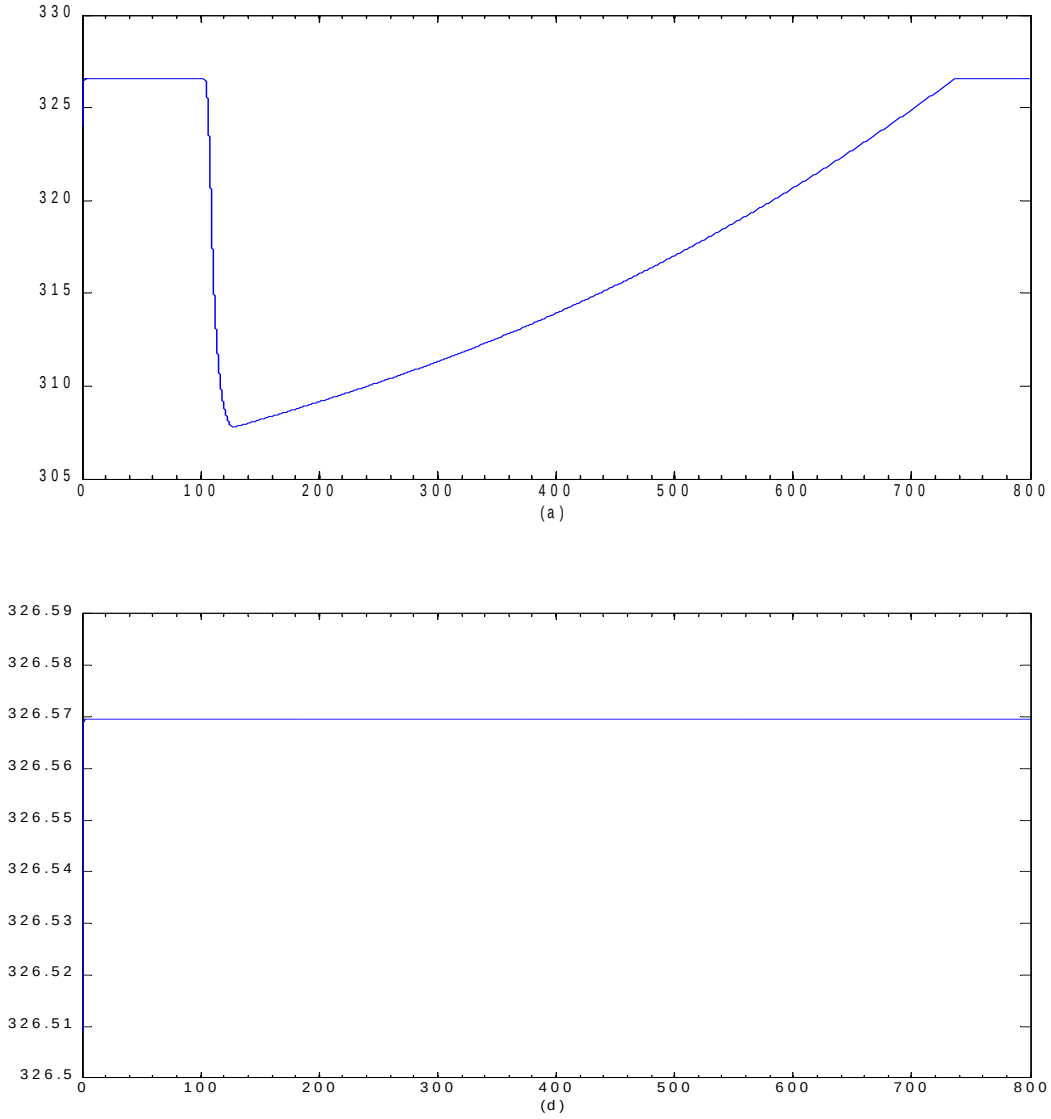

Figure 12: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-3}$ s. (a) Estimated pressurizer temperature [°C]; (b) Pressurizer reference temperature [°C]

1.3.2 *Digital Inventory* `Pressurizer_inventory_controller_1k` *and Pressure Control* `Pressurizer_pressure_controller_1k` *with Sampling Time* $\delta = 10^{-2}$ *s*

The simulation results are summarized in Figures 13–19.

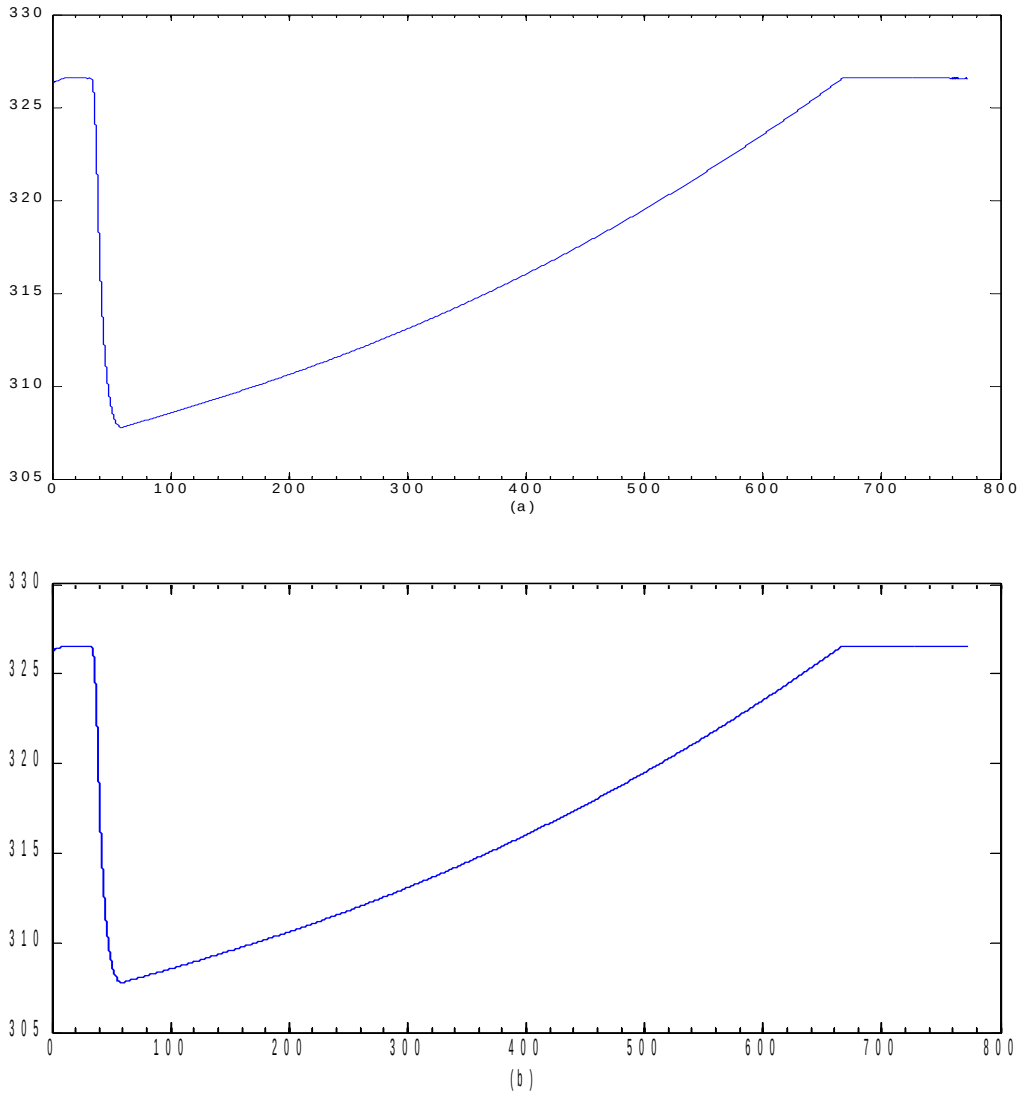

Figure 13: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-2}$ s. (a) Pressurizer temperature [°C]; (b) Pressurizer wall temperature [°C]

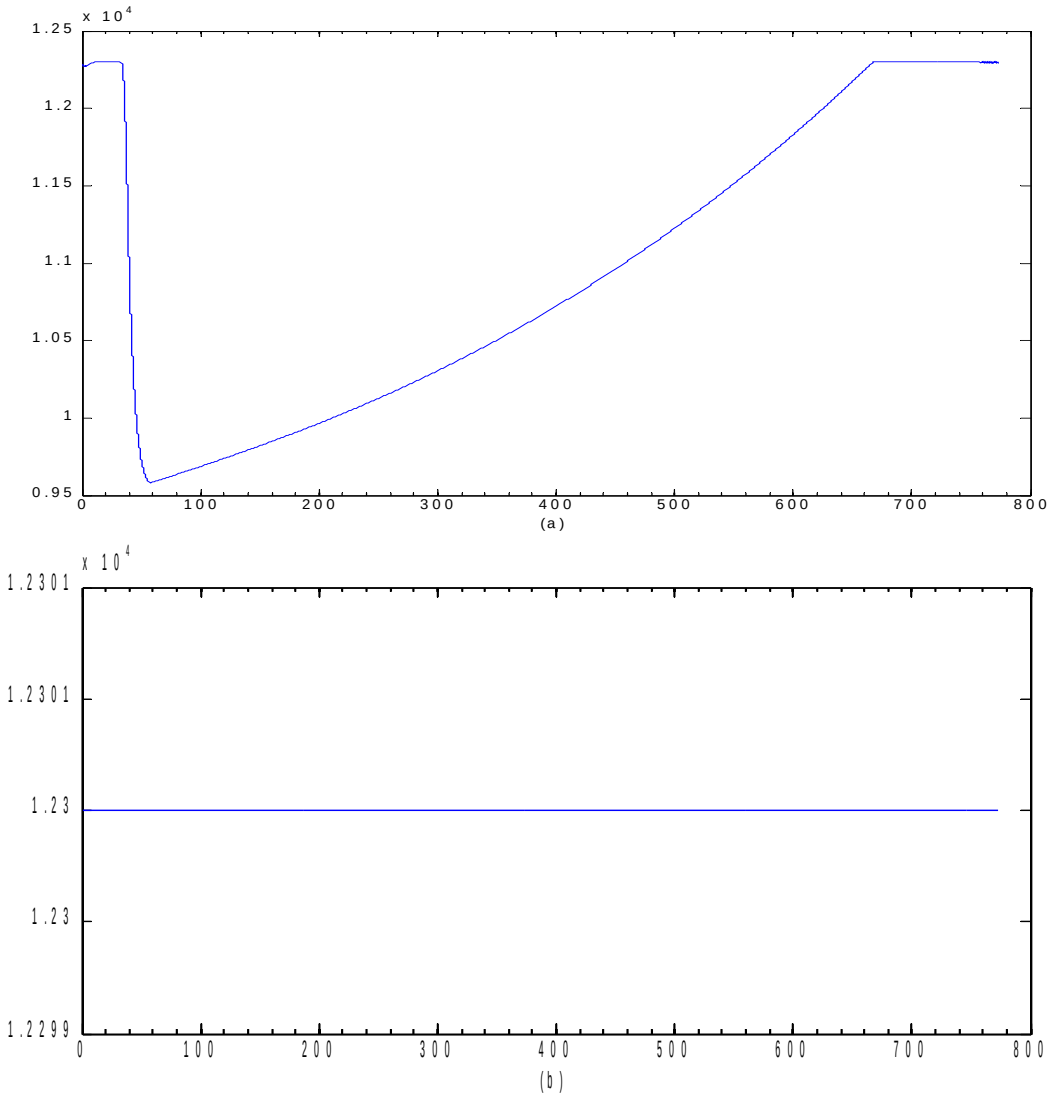

Figure 14: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-2}$ s. (a) Pressurizer pressure [Pa]; (b) Pressurizer pressure reference [Pa]

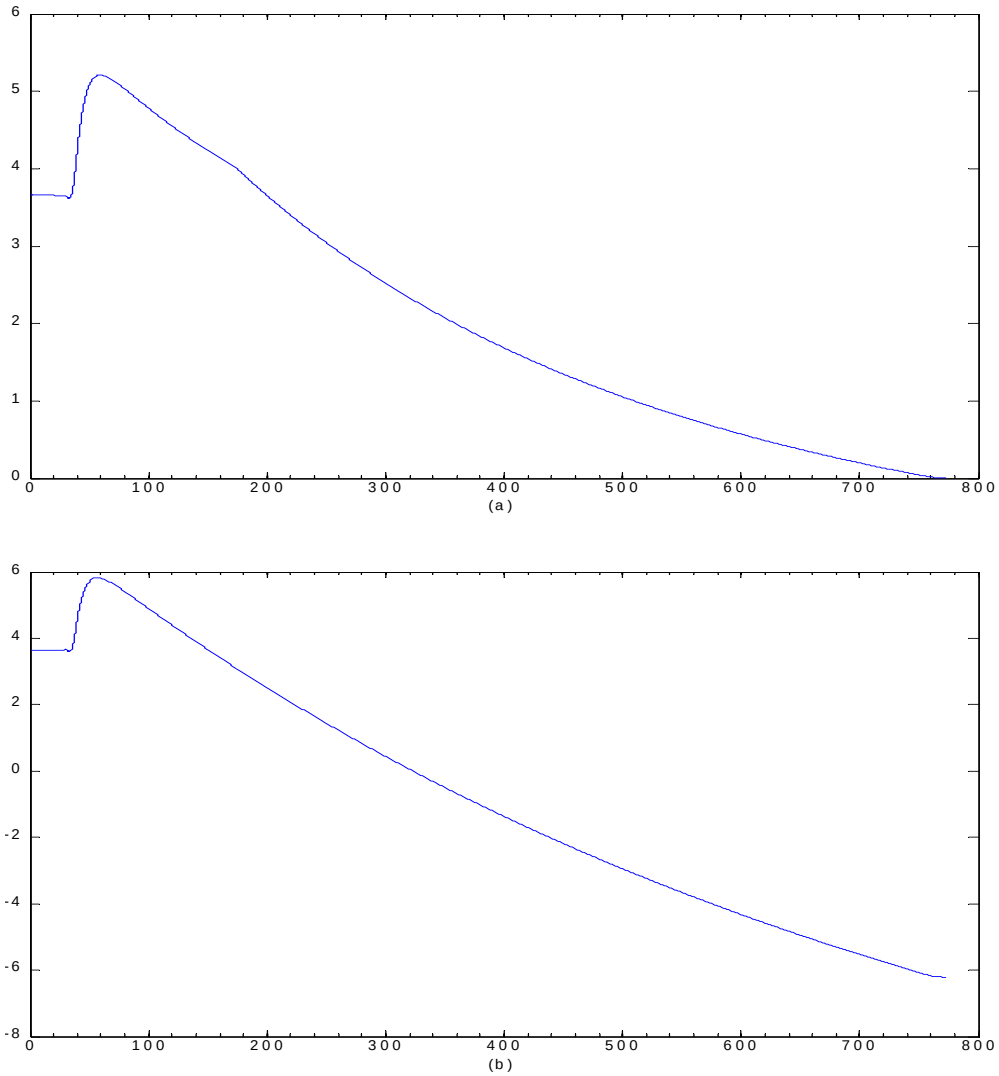

Figure 15: Controllers (6)-(7), $\Delta_t = 0.01$. (a) Pressurizer water level [m]; (b) Pressurizer water level reference [m]

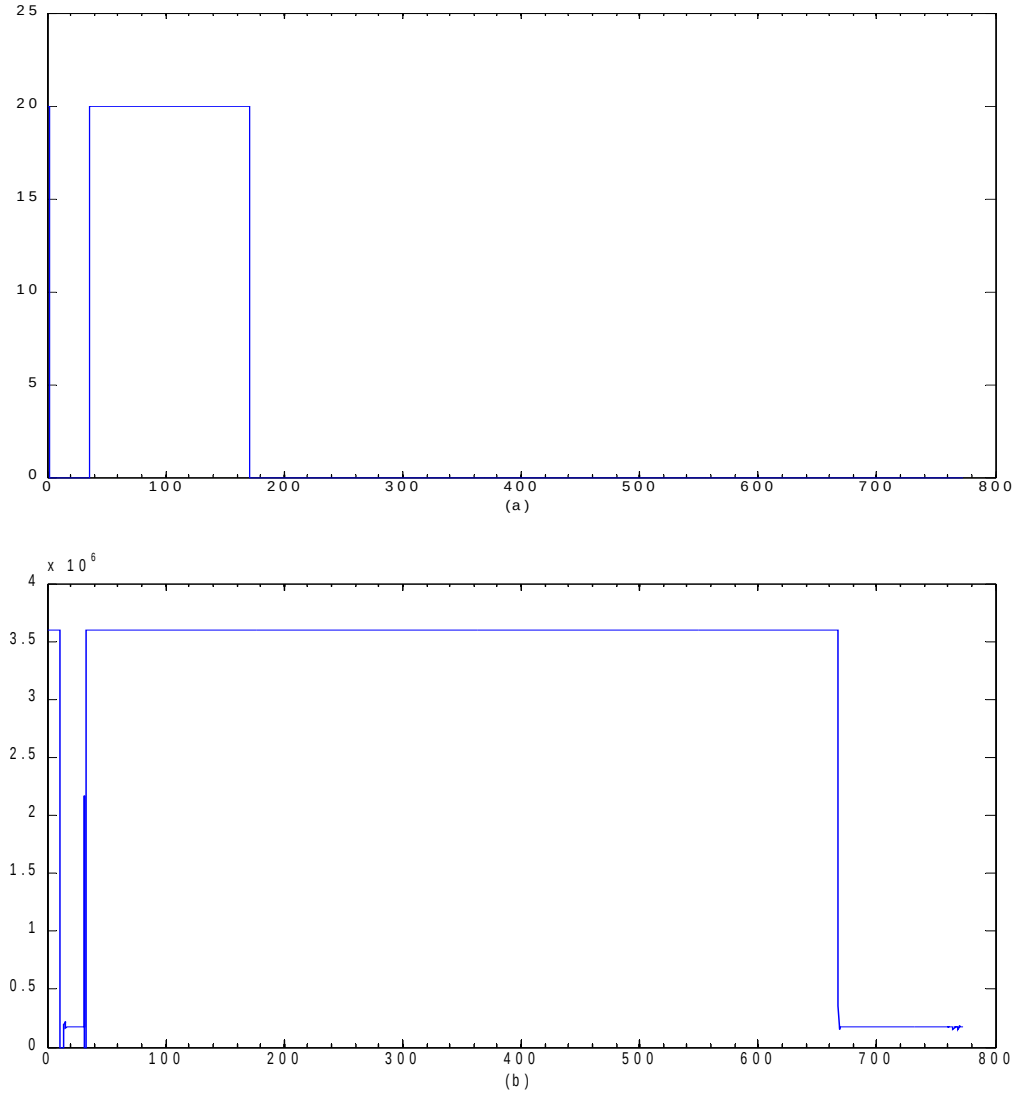

Figure 16: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-2}$ s. (a) Inlet mass flow rate [kg/s]; (b) Pressurizer heating power [W]

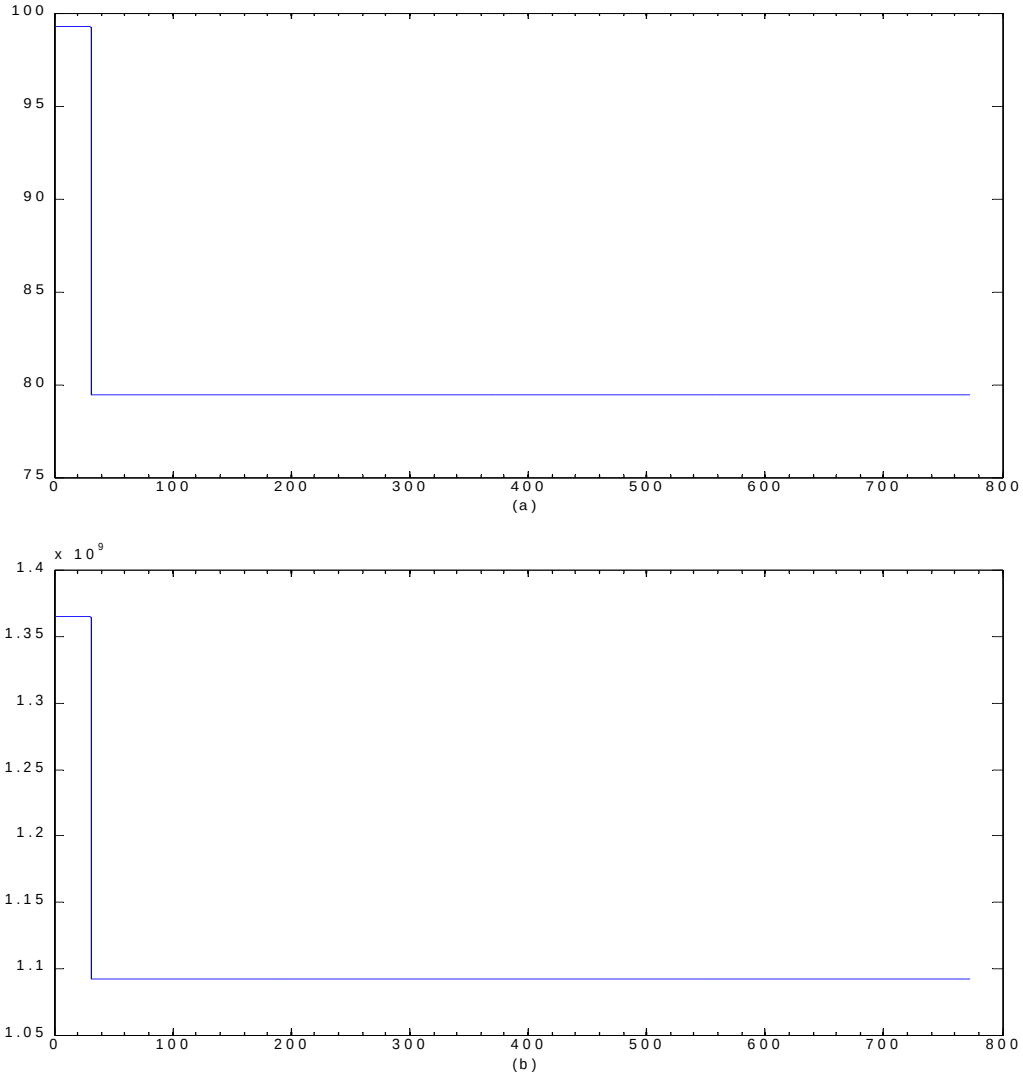

Figure 17: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-2}$ s. (a) Neutron flux [%]; (b) Reactor power [W]

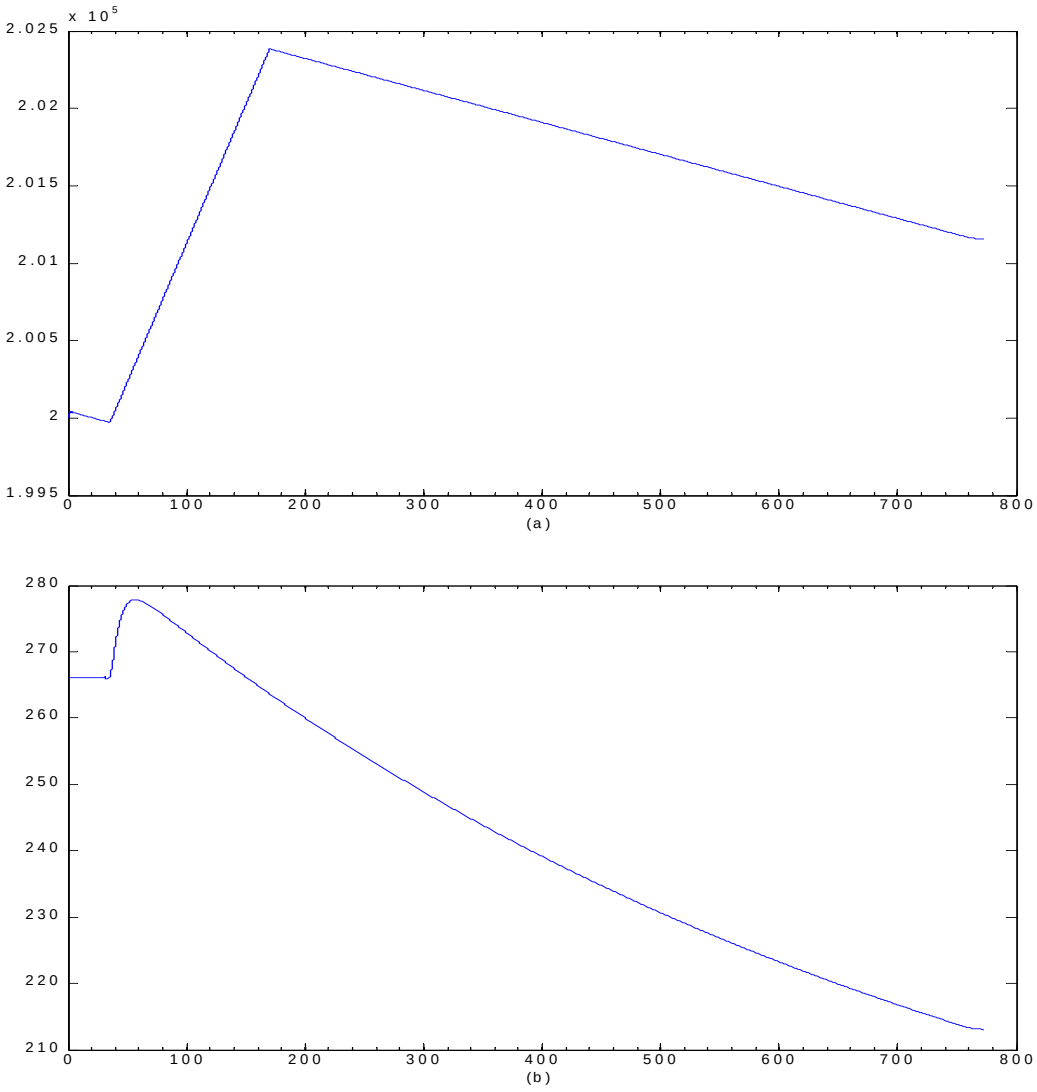

Figure 18: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-2}$ s. (a) Primary circuit water mass [kg]; (b) Primary circuit avarege temperature [°C]

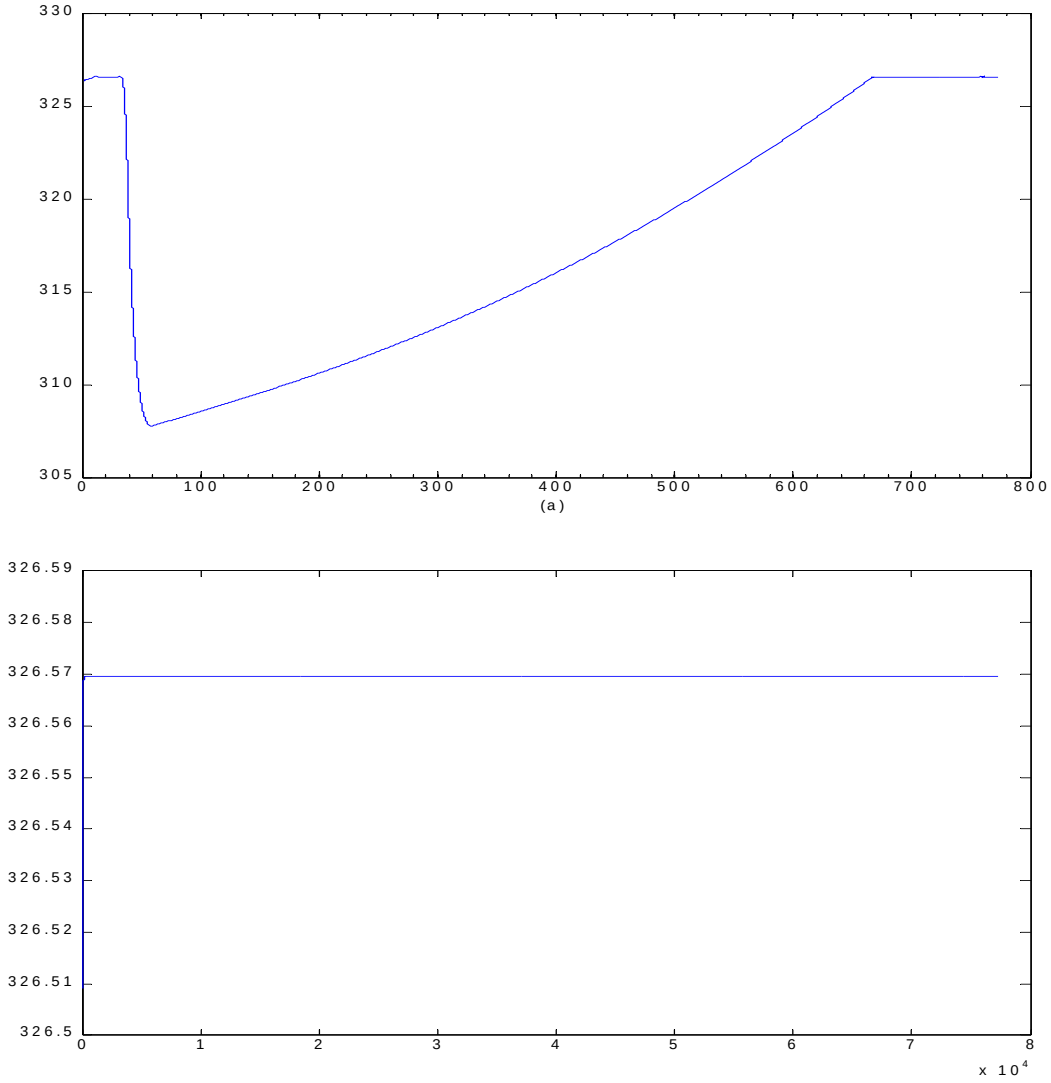

Figure 19: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-2}$ s. (a) Estimated pressurizer temperature [°C]; (b) Pressurizer reference temperature [°C]

1.3.3 *Digital Inventory* `Pressurizer_inventory_controller_1k` *and Pressure Control* `Pressurizer_pressure_controller_1k` *with Sampling Time* $\delta = 10^{-1}$ *s*

The simulation results are summarized in Figures 20–26.

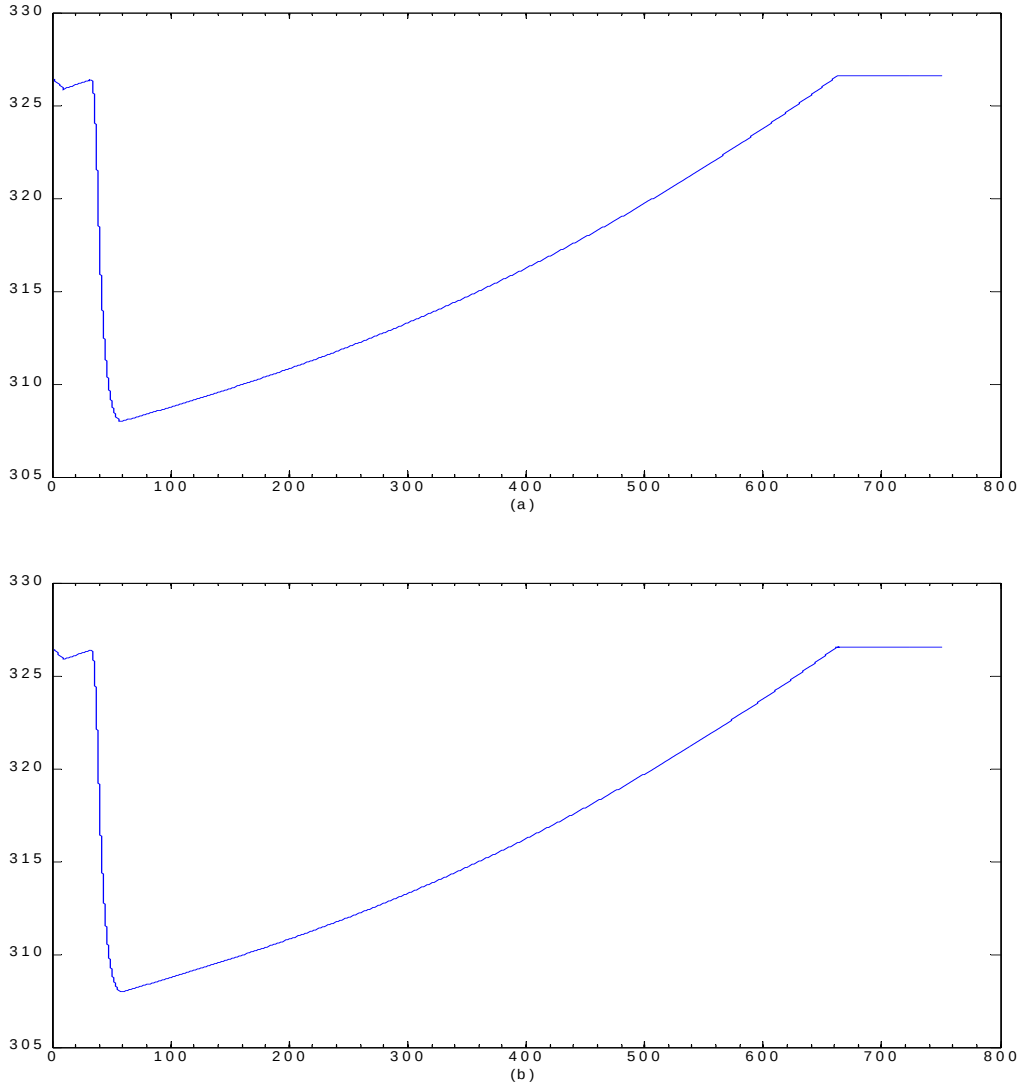

Figure 20: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-1}$ s. (a) Pressurizer temperature [°C]; (b) Pressurizer wall temperature [°C]

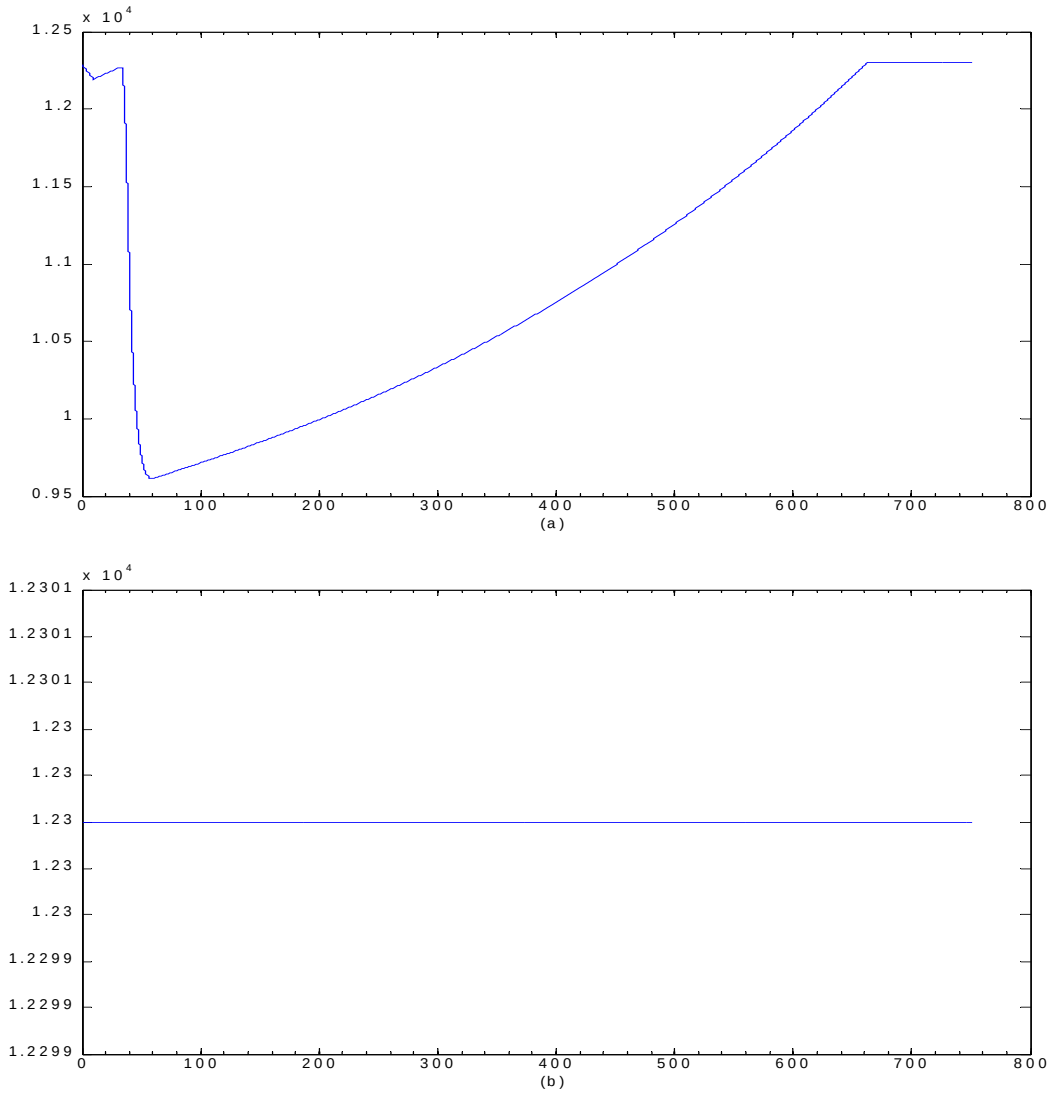

Figure 21: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-1}$ s. (a) Pressurizer pressure [Pa]; (b) Pressurizer pressure reference [Pa]

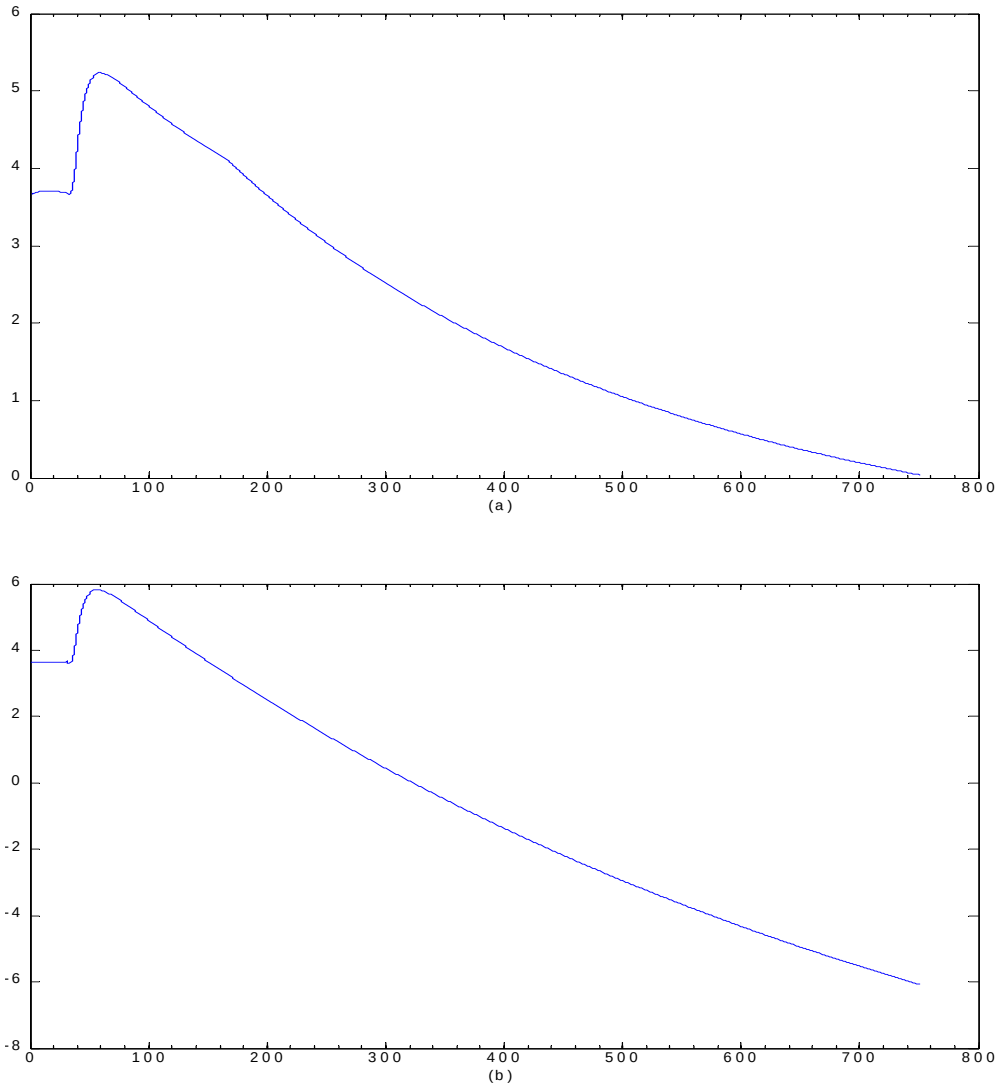

Figure 22: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-1}$ s. (a) Pressurizer water level [m]; (b) Pressurizer water level reference [m]

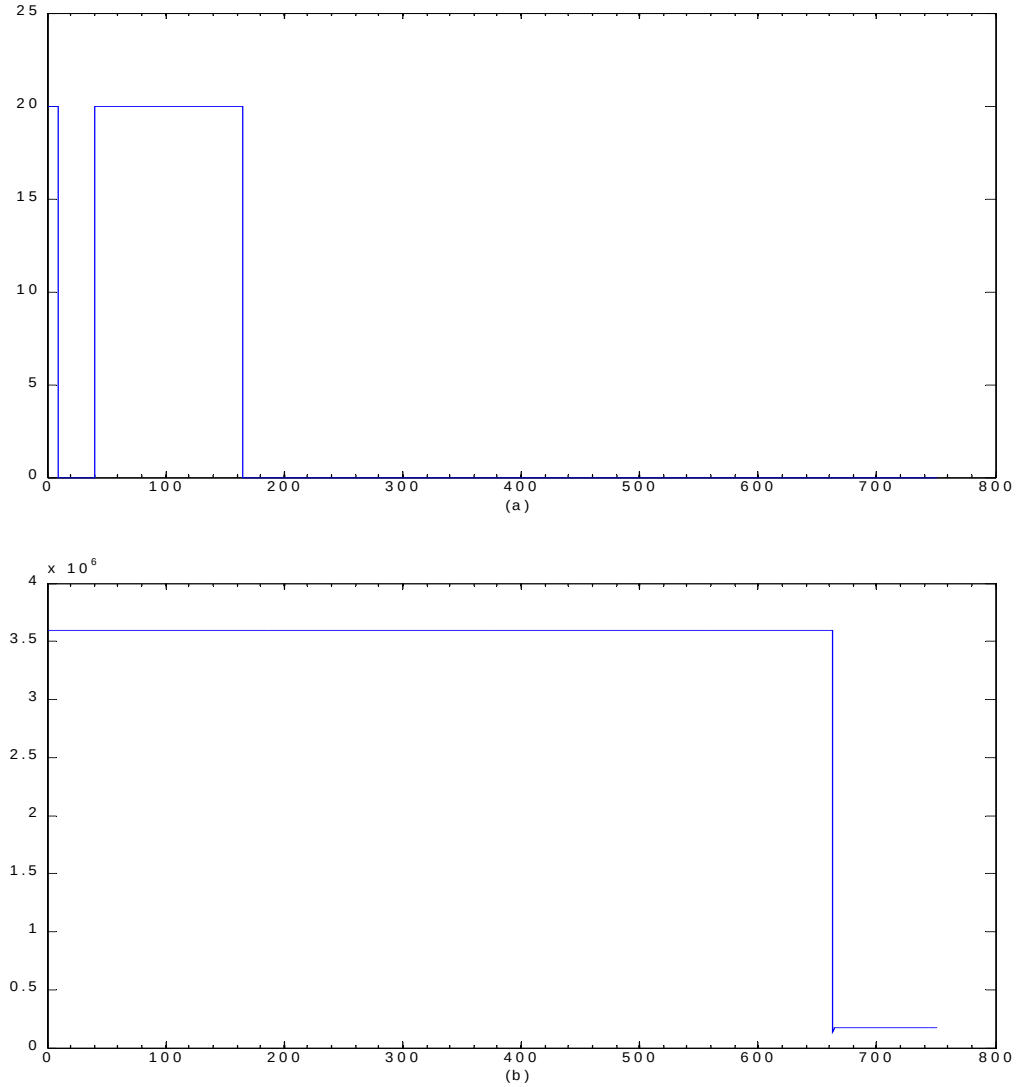

Figure 23: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-1}$ s. (a) Inlet mass flow rate [kg/s]; (b) Pressurizer heating power [W]

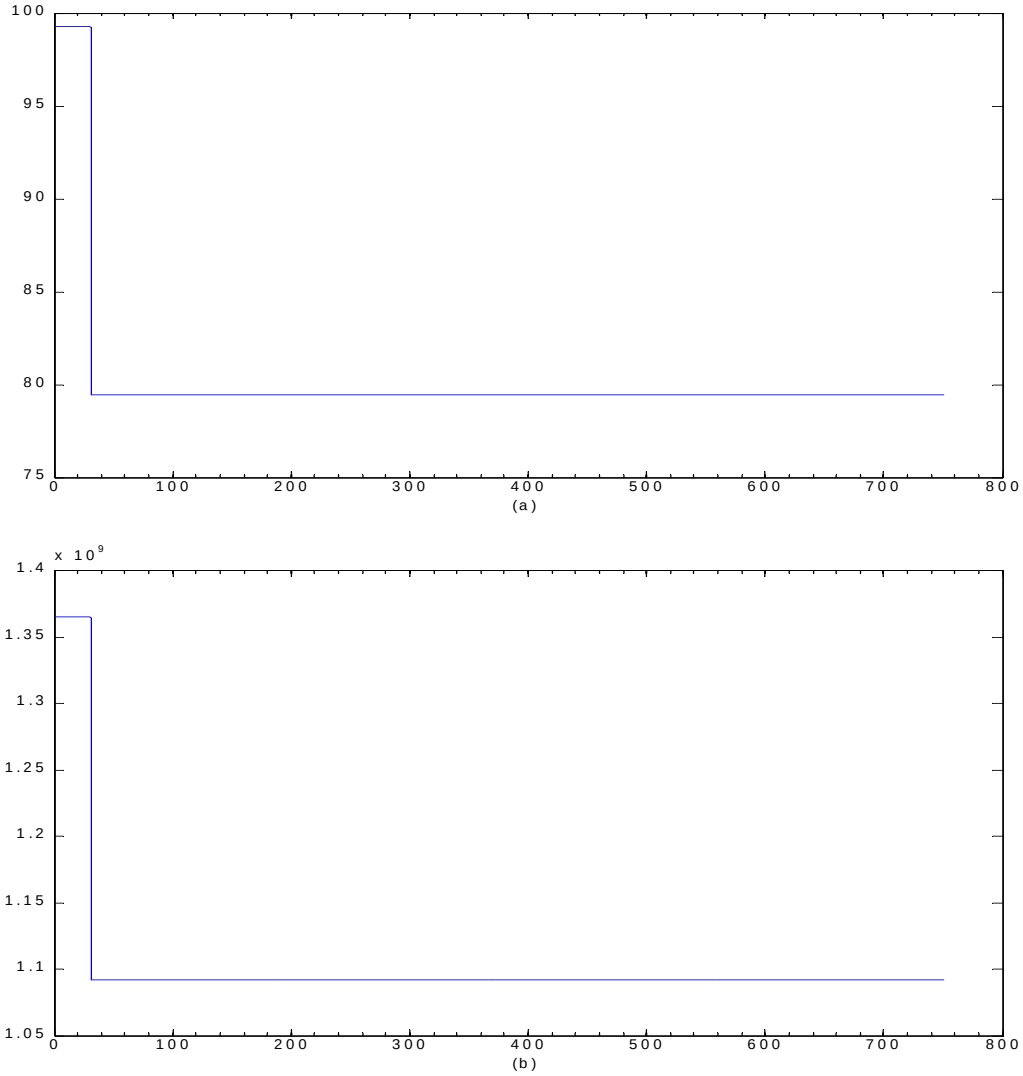

Figure 24: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-1}$ s. (a) Neutron flux [%]; (b) Reactor power [W]

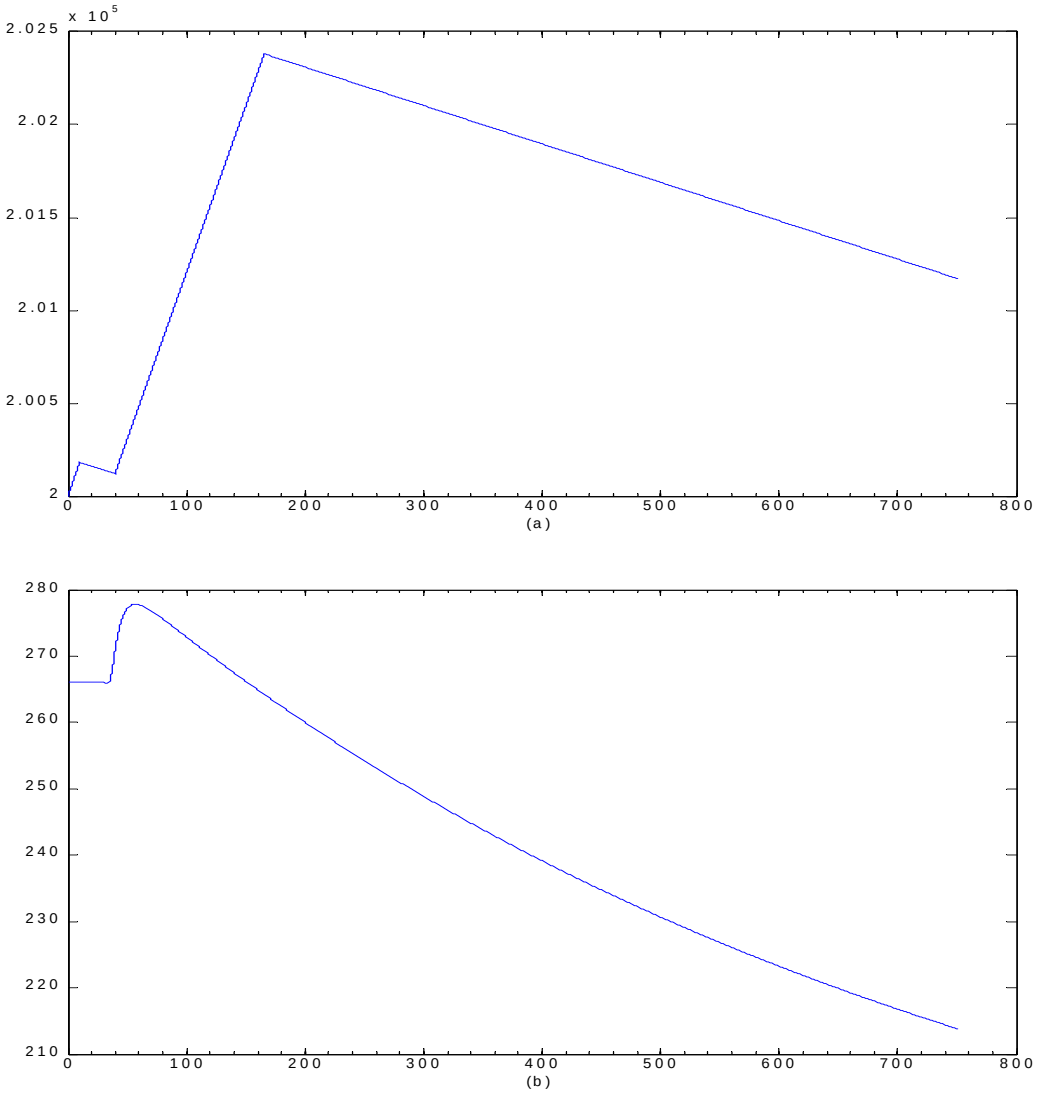

Figure 25: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-1}$ s. (a) Primary circuit water mass [kg]; (b) Primary circuit avarege temperature [°C]

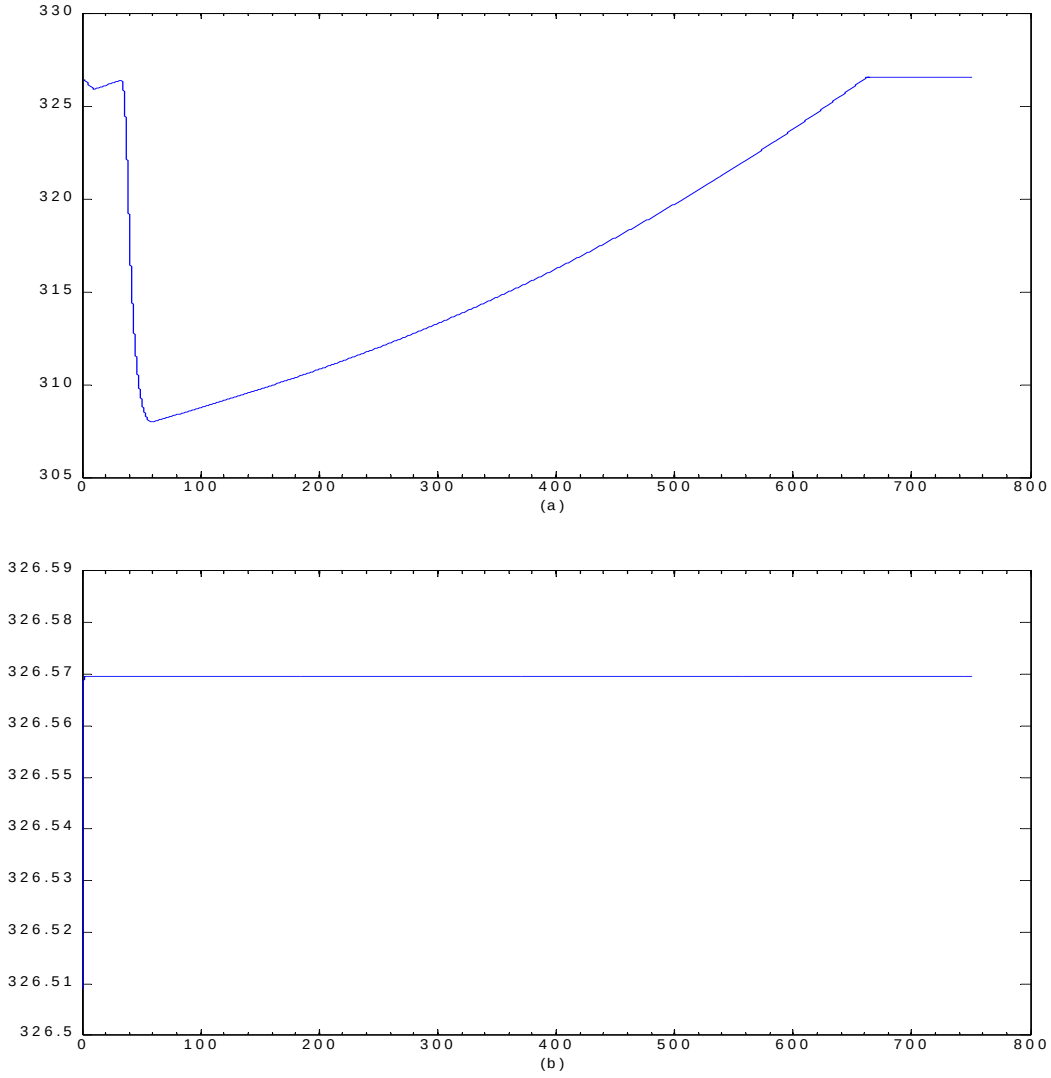

Figure 26: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 10^{-1}$ s. (a) Estimated pressurizer temperature [°C]; (b) Pressurizer reference temperature [°C]

1.3.4 *Digital Inventory* `Pressurizer_inventory_controller_1k` *and Pressure Control* `Pressurizer_pressure_controller_1k` *with Sampling Time* $\delta = 1$ *s*

The simulation results are summarized in Figures 27–33.

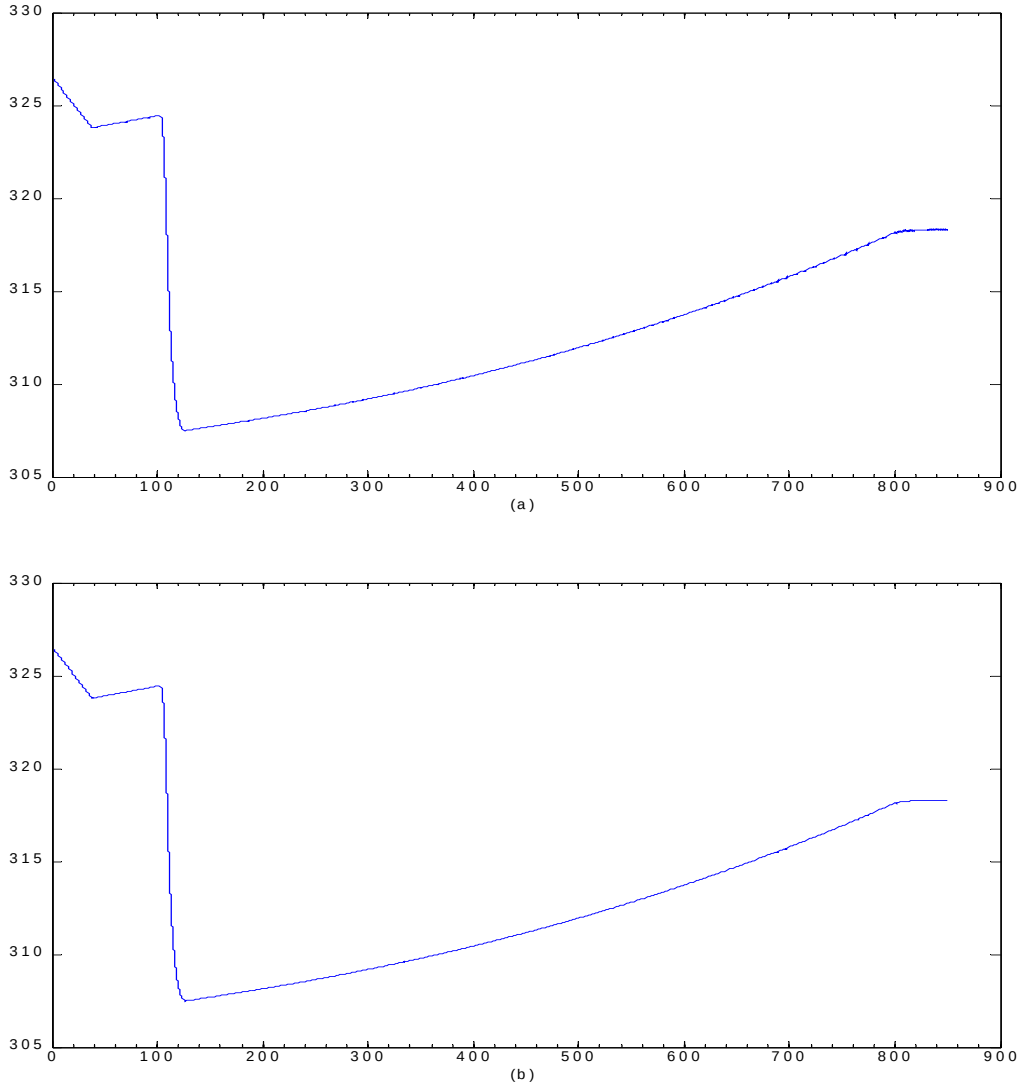

Figure 27: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 1$ s. (a) Pressurizer temperature [°C]; (b) Pressurizer wall temperature [°C]

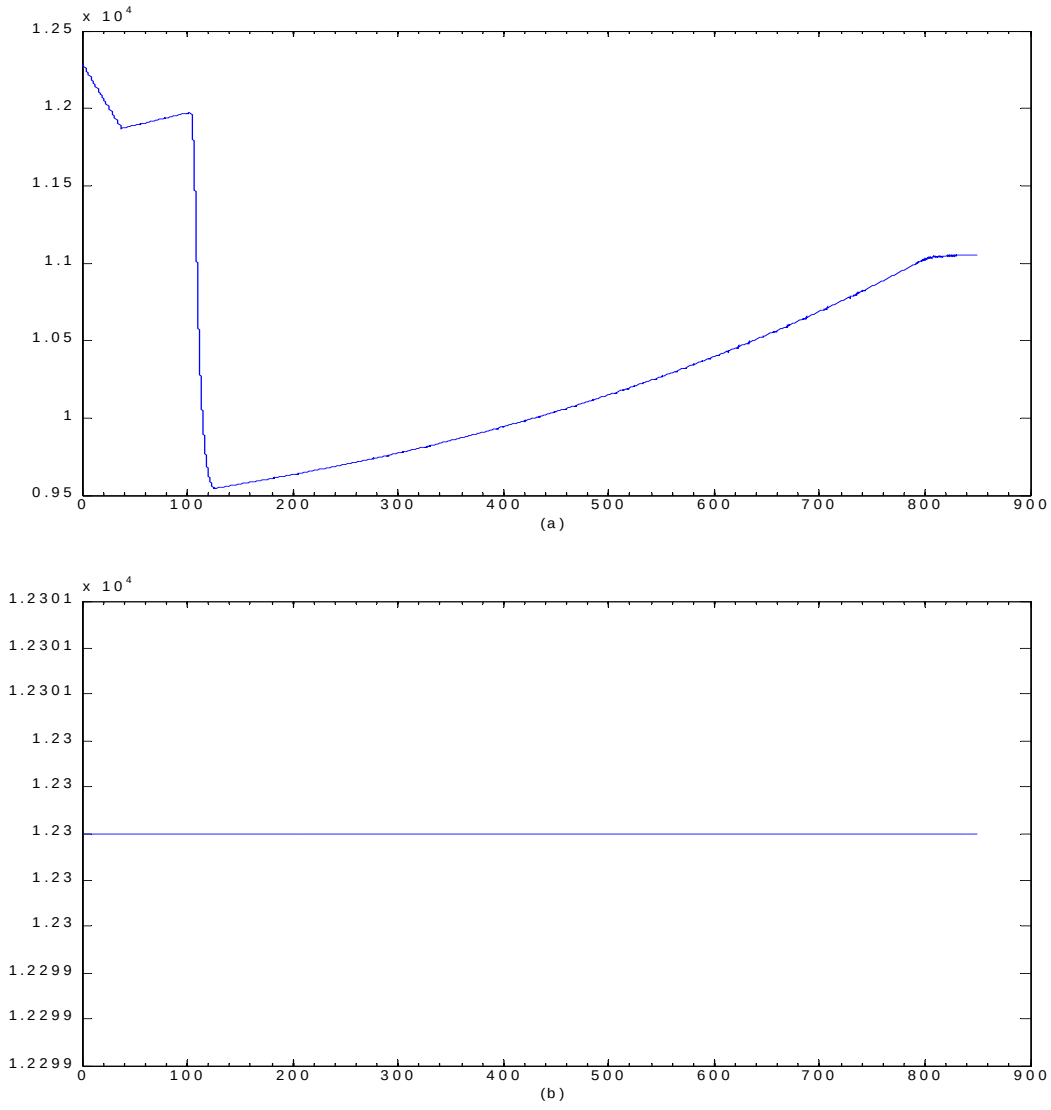

Figure 28: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 1$ s. (a) Pressurizer pressure [Pa]; (b) Pressurizer pressure reference [Pa]

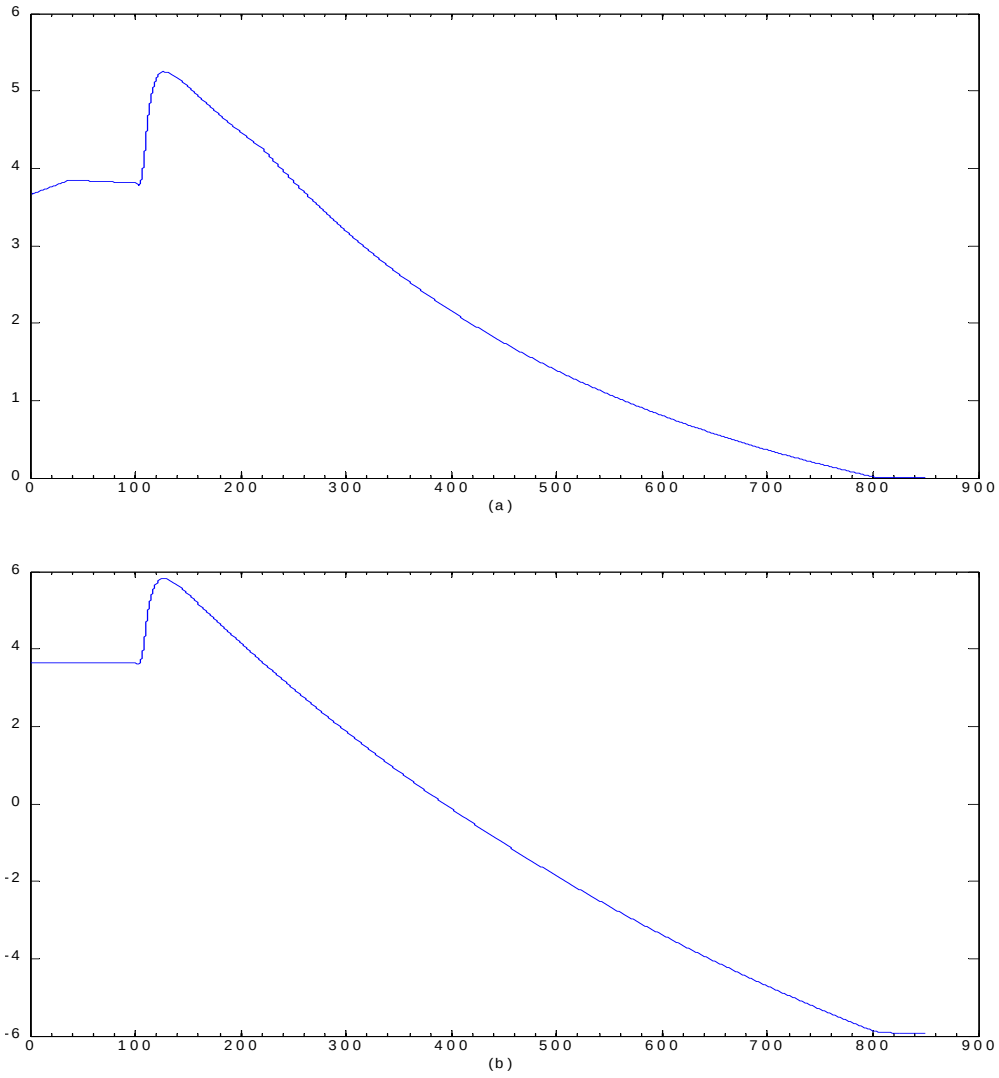

Figure 29: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 1$ s. (a) Pressurizer water level [m]; (b) Pressurizer water level reference [m]

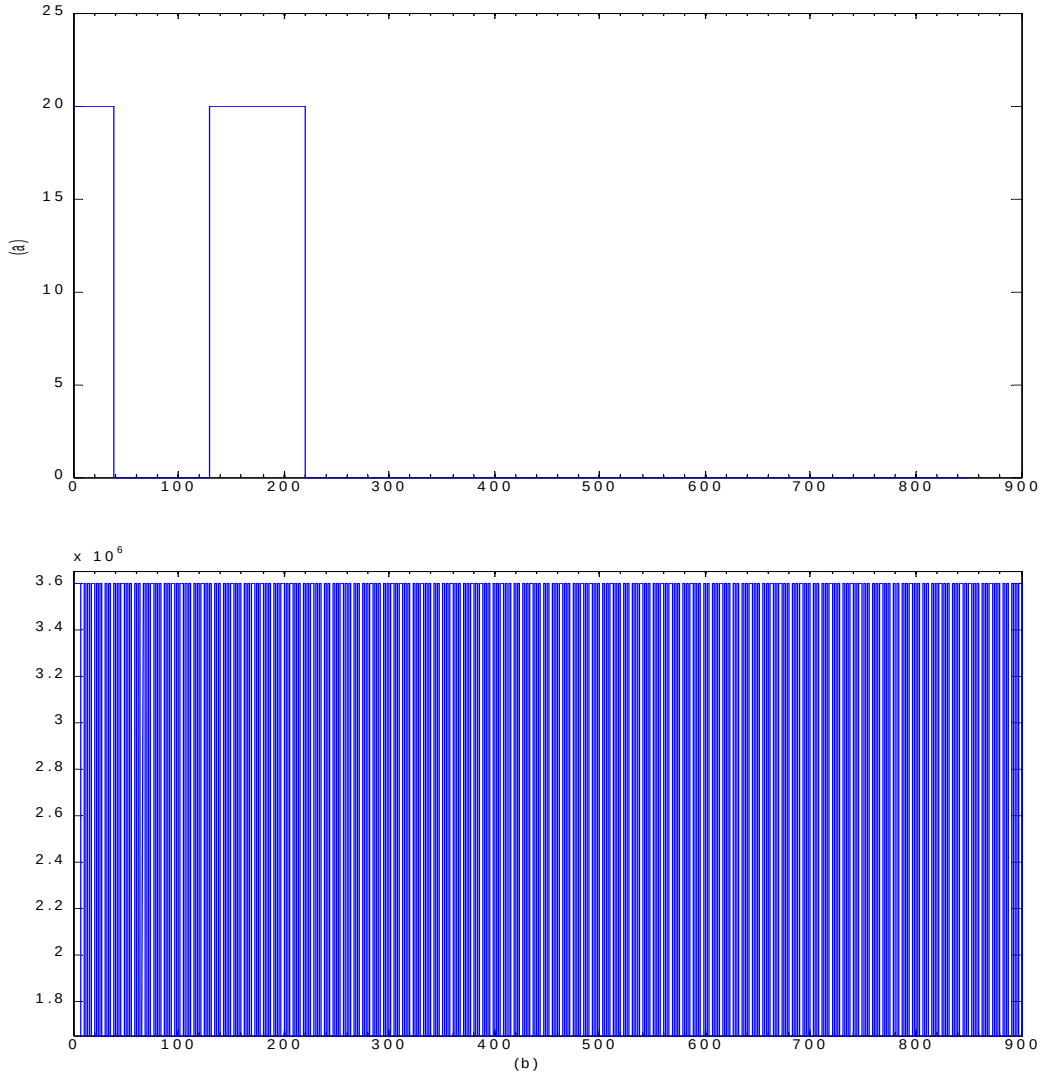

Figure 30: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 1$ s. (a) Inlet mass flow rate [kg/s]; (b) Pressurizer heating power [W]

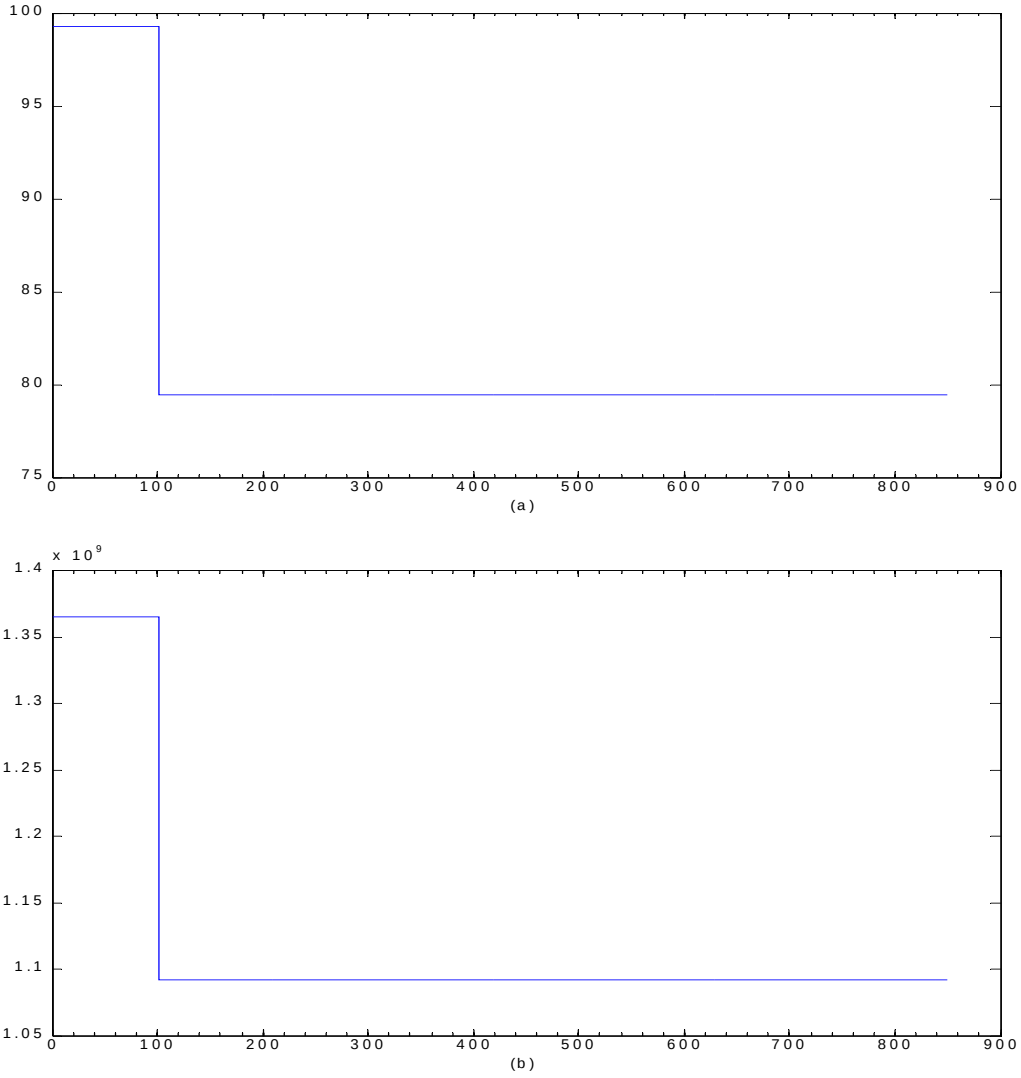

Figure 31: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 1$ s. (a) Neutron flux [%]; (b) Reactor power [W]

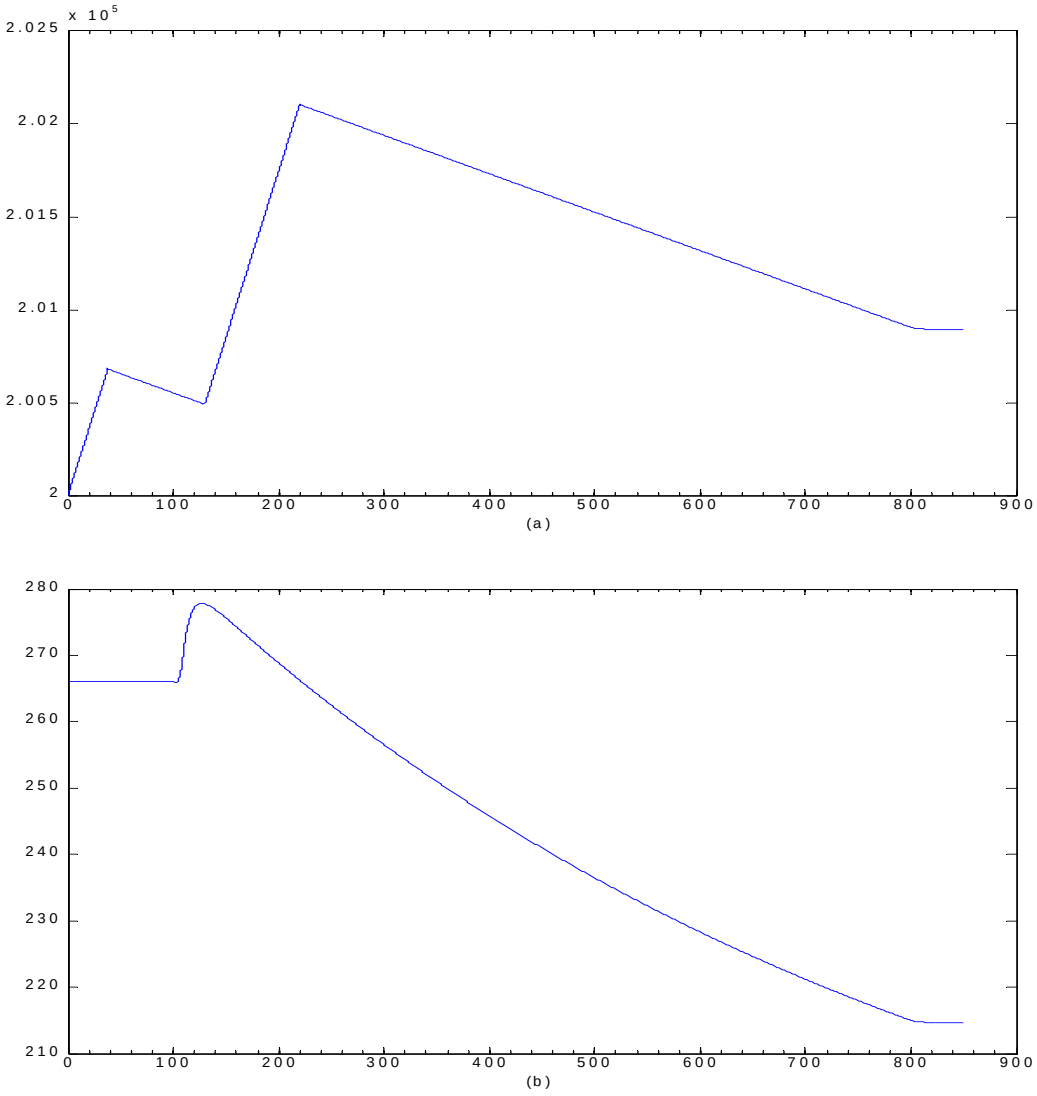

Figure 32: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 1$ s. (a) Primary circuit water mass [kg]; (b) Primary circuit avarege temperature [°C]

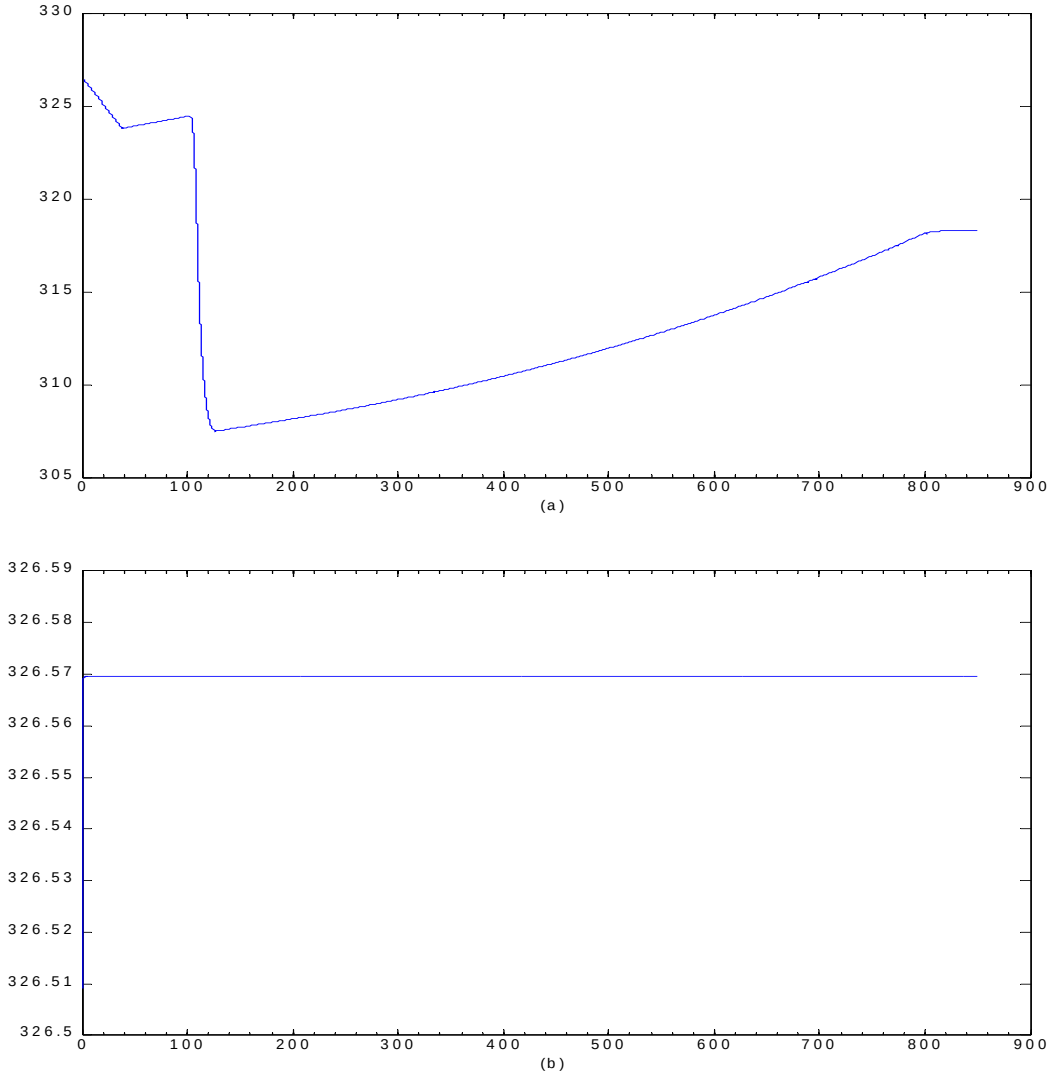

Figure 33: Digital inventory and pressure control `Pressurizer_inventory_controller_1k`, `Pressurizer_pressure_controller_1k` with sampling time $\delta = 1$ s. (a) Estimated pressurizer temperature [°C]; (b) Pressurizer reference temperature [°C]

2 National Instruments LabVIEW: Software and Hardware

The controllers designed and tested on the basis of the simulation environment have to be implemented on a digital device. Before this final step, it is common to insert some hardware in the loop to better check the real behavior of the closed loop system. In particular, it is possible to apply the controller to the real process under study and to calculate the control law considering measurements from the real process. For, it is necessary to use an appropriate real–time environment where is possible to acquire and store data, compute the control algorithm, apply the control action to actuators. Therefore, once the controller has been designed in Simulink, it is necessary to interface Simulink with this real–time environment for further checks of the control algorithm.

One possible choice is LabVIEW©, described in the following. Even though LabVIEW is not certified for nuclear applications, this choice is motivated by the fact that it is a very popular software for real–time interfaces, simple enough for the purpose of the present project. LabVIEW, hence, allows checking the methodological steps for the real–time prototyping of the controllers. For future industrial applications this software can still be used, while in the case of applications in nuclear environment, more costly nuclear–certified softwares, ensuring the same real–time performances of LabVIEW, can be considered.

2.1 Programming Language: LabVIEW RT

LabVIEW is a graphic programming language that makes use of icon instead of code lines to create a specific application. This is a software language based on data flow, that determines a program execution. In LabVIEW environment is possible to implement a user interface using objects and tools. This mentioned interface is known as front panel. Code is added adopting a graphic representation of different functions to manage front panel objects: the block diagram checks this elements. LabVIEW is totally integrated to communicate through GPIB, VXI, PXI, RS–232, RS-485 interface and plug–in DAQ. This National Instruments property language is useful to create test and measurement application, to acquire data, instruments control, data storage and analysis.

LabVIEW programs are called Virtual Instruments and they contain three fundamental elements:

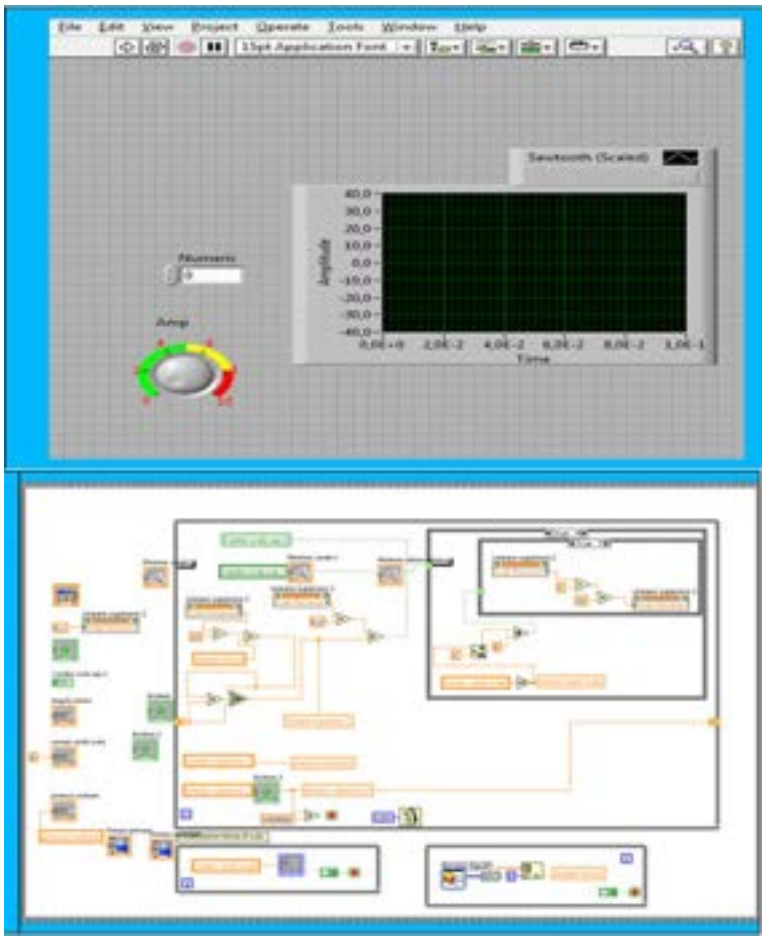

Figure 34: LabView Programming Language Example of Back–End Layer and Applicative User Interface

a front panel, a block diagram and one icon with the connector box. The front panel has indicators, controls, buttons and other elements. Indicators are graphs, LED, etc. Controls simulate instrument input and provide data to the block diagram. Indicators simulate device output and display data that the block diagram generates and acquires. Primary potentiality of LabVIEW resides in its hierarchical nature of programming: it is possible to create subroutines, that can be recalled and reused in other projects. In addition to the LabVIEW real time module, an FPGA one is available to create a lower level application to program the FPGA. This kind of module requests some important rules and tricks to optimize the FPGA resources usage and exploit its potentials.

2.2 Introduction to the CompactRIO Architecture

The hardware architecture employed in this project includes a National Instruments product known as CompactRIO, a reconfigurable, embedded system implementing data acquisition and manipulation [27]. This kind of architecture presents I/O modules, a FPGA chassis and an embedded controller, pro-

grammable through LabVIEW programming language. The main reason leading to choose this system typology is the ability to make advanced analysis, but its potentialities also cover elevated signals elaboration, control algorithms from immediate PID systems to dynamic ones like model predictive controls (MPCs). All of these features are well–defined to achieve an adequate level of determinism. Another benefit is represented by the modularity and flexibility guaranteed by this platform: as a matter of fact a large range of controller, reconfigurable chassis and I/O modules can be included to easily change over from prototype to production.

The real–time controller is a microprocessor that executes LabVIEW real–time code in a reliable and deterministic way offering control, datalogging and peripheral communication. Because each acquisition module is connected directly to the FPGA rather than through a bus, there is almost no control latency for system response compared to other controller architectures.

By default, this FPGA automatically communicates with I/O modules and provides deterministic I/O to the real–time processor through a PCI bus communication. Out of the box, the FPGA enables programs on the real–time controller to access I/O with less than 500 ns of jitter between loops. You can also directly program this FPGA to further customize the system. Because of the FPGA speed, this chassis is frequently used to create controller systems that incorporate high-speed buffered I/O, very fast control loops, or custom signal filtering. For instance, using the FPGA, a single chassis can execute more than 20 analog proportional integral derivative (PID) control loops simultaneously at a rate of 100 kHz. Additionally, because the FPGA runs all code in hardware, it provides the highest reliability and determinism that is ideal for hardware-based interlocks, custom timing and triggering, or eliminating the custom circuitry normally required with nonstandard sensors and buses.

Lastly I/O modules comprise isolation stages, conversion and signal conditioning for a direct connection to sensors and motor units. Modules easily available on the market includes several signal typologies inputs and actuators: thermocouple inputs, current or voltage, strain gauges, or digital inputs/outputs. Additionally, you can build your own modules or purchase modules from third-party vendors.

A similar platform has been chosen because with LabVIEW FPGA is possible to

1. Collect analog waveform at rates of hundreds of kilohertz;
2. Create custom digital pulse trains at up to 40 MHz;
3. Implement custom digital communication protocols;
4. Run control loops at rates in the hundreds of kilohertz;
5. Use modules not supported by the scan mode including CAN and PROFIBUS communication;

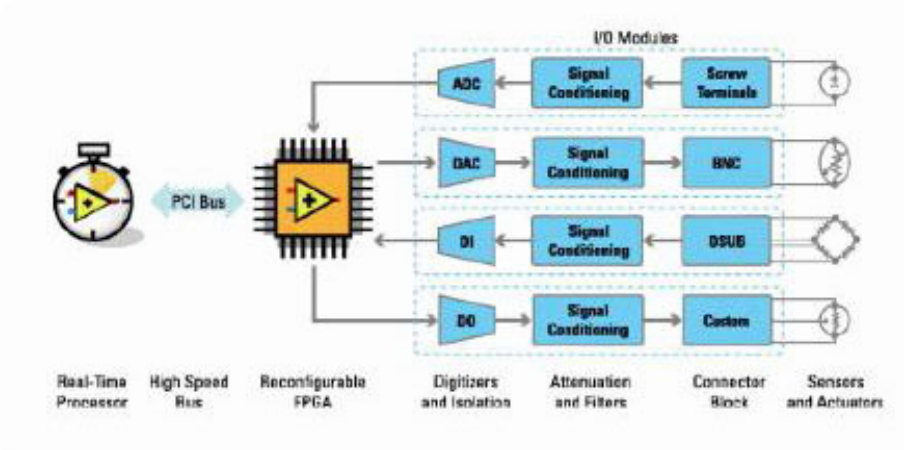

Figure 35: Scheme of CompactRIO Embedded System, Exploiting Processor and FPGA Technologies

6. Implement custom timing, triggering, and filtering.

With the combination of a real–time processor and programmable FPGA on CompactRIO, we take advantage of the strengths of each computing platform. The real–time processor excels at floating–point math and analysis and peripheral communication such as network-published shared variables and Web services. The FPGA excels at smaller tasks that require very high-speed logic and precise timing. A scenario for which the FPGA is programmed directly can include

1. High–Speed Waveform Acquisition /Generation (greater than 1 kHz).

   If an aquisition or generation at speeds higher than 1 kHz is needed to take full advantage of these module features, LabVIEW FPGA can acquire at a user-defined rate tailored to a specific application.

2. Custom Triggering/Timing/Synchronization.

   With the reconfigurable FPGA, a programmer can create simple, advanced, or otherwise custom implementations of triggers, timing schemes, and I/O or chassis synchronization. These can be as elaborate as triggering a custom CAN message based on the rise of an analog acquisition exceeding a threshold or as simple as acquiring input values on the rising edge of an external clock source.

3. Hardware–Based Analysis/Generation and Coprocessing.

   Many sensors output more data than can be reasonably processed on the real–time processor alone. The FPGA can be used as a valuable coprocessor to analyze or generate complex signals while freeing the processor for other critical threads.

This type of FPGA based coprocessing is commonly used in applications such as

Encoding/decoding sensors

(a) Tachometers

(b) Standard and/or custom digital protocols

Signal processing and analysis

(a) Spectral analysis (fast Fourier transforms and windowing)

(b) Filtering, averaging, and so on

(c) Data reduction

(d) Third-party IP integration

Sensor simulation

(a) Linear-voltage differential transformers (LVDTs)

(b) Cam and crank

Hardware-in-the-loop simulation.

4. Highest–Performance Control

   Not only the FPGA can realize high-speed acquisition and generation, but also can implement many control algorithms on the FPGA. You can use single–point I/O with multichannel, tunable PID or other control algorithms to implement deterministic control with loop rates up to hundreds of kilohertz.

5. Unsupported Modules

   Several C Series modules do not feature scan mode support. For these modules, you need to use LabVIEW FPGA to build an interface between the I/O and your real–time application. For a list of modules that feature scan mode support, see C Series Modules Supported by CompactRIO Scan Mode. Unsupported Targets CompactRIO targets with 1M gate FPGAs cannot fully support the scan mode. You can implement some scan mode features on unsupported targets, but you must use LabVIEW FPGA. The knowledge base article Using CompactRIO Scan Mode with Unsupported Backplanesi describes how to use LabVIEW FPGA to build a custom scan mode interface for an unsupported FPGA target.

Because LabVIEW FPGA code runs directly on hardware, the main advantages of FPGA–based design are:

1. *High Reliability*

   LabVIEW FPGA code running on a FPGA is highly reliable because the logic is compiled into a physical hardware design. Once the FPGA is programmed, it becomes a hardware chip with all of the associated reliability.

2. *High Determinism*

   Processor-based systems often involve several abstraction layers to help schedule tasks and share resources among multiple processes. The driver layer controls hardware resources and the operating system manages memory and processor bandwidth. For any given processor core, only one instruction can execute at a time. Real-time operating systems reduce jitter to a finite maximum when programmed with good priority hierarchy. FPGAs do not use any operating systems. This minimizes reliability concerns with true parallel execution and deterministic hardware dedicated to every task. For NI FPGA–based hardware, you can achieve 25 ns timing accuracy of critical components within your design.

3. *True Parallelism*

   Multithreaded applications break down into multiple parallel sections of code which are executed in a round-robin fashion, giving the appearance of parallel execution. Multicore processors expand on that idea by allowing multithreaded applications to truly execute multiple parallel code at one time. The number of parallel pieces of code executing concurrently is limited to the number of cores available in the specific processor. Because an FPGA implements parallel code as parallel circuits in hardware, you are not limited by processor cores; therefore, every piece of parallel code in an entire FPGA application can execute concurrently. Even traditionally serial operations can improve throughput on FPGAs by implementing pipelining.

4. *Reconfigurability*

   Being reconfigurable, FPGA chips are able to keep up with any future modifications you might need. As a product or system matures, functional enhancements are always feasable without spending time redesigning hardware or modifying a board layout. This is especially applicable to industrial communication protocols. As communication protocols evolve and improve over

time, you can modify the implementation of that protocol within an FPGA to support the latest technology features and changes.

5. *Instant Boot Up*

   Because LabVIEW FPGA code runs directly on the FPGA without an overarching operating system, the code downloaded to the FPGA flash memory begins running within milliseconds of powering on your CompactRIO chassis. This code can begin executing a control loop or setting startup output values.

2.3 Mathworks–LabVIEW Interface

Designed to offer a desktop interface to mathematically manage a model, the languages known as .m files simplify the process of development of intellectual algorithms and IP but they often complicate the embedded hardware conversion. This kind of programs, used in software as MathWorks and Scilab, manipulate data as a numerical matrix, in this way the concept of ”data type” is not realized and there is a dynamic allocation of memory. This can be a strong limit in embedded architectures and OS cannot operate in similar conditions, due to timing constraints and their deterministic nature. Another aspect to focus that .m files are not compiled but interpreted: without a code compilation the fundamental benefit of errors identification before the execution is lost. Furthermore, this language does not contain timing and resources management causing the code to be written once again in a more suitable one (such as C) to program the embedded system. Taking a script developed using Matlab and replicating it in a multicore realtime hardware can request the depicted steps.

Figure 36: Main steps passing through Simulink environment to a real time one

With the matlab syntax, firstly it is necessary to test a script in the desktop development environment using the Parallel Computing toolbox to prepare code for a dual-core environment. Then the Embedded Matlab is adopted to generate C code and in the end the code has to be compiled and debugged in a seperate embedded toolchain. This path can be very dangerous for timing and precision of the mathematical implementations.

In this scenario, NI provides a Mathscript module that provides a textual programming math-oriented through a native compiler for .m files. In this way every .m file can be included in realtime hardware. The following figure shows the Mathscript Node which permit to execute scripts .m from a LabVIEW VI.

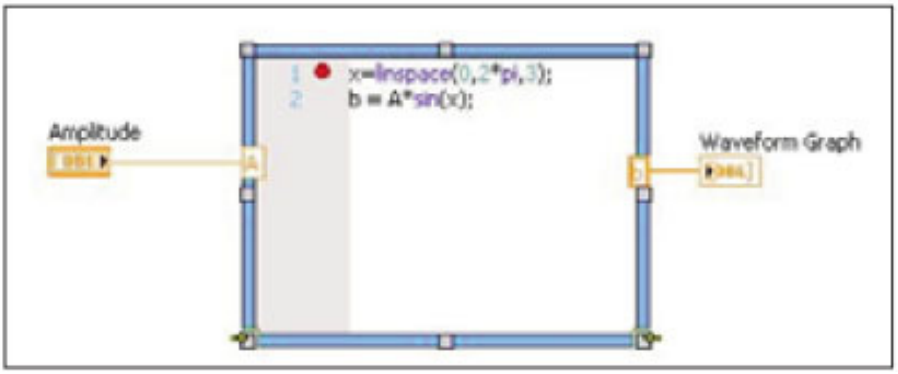

Figure 37: Matlab rootines read in a Labview application

A first advantage of the LabVIEW compiler is the capacity to express parallelism: there is no need for special markup or in the code to force a parallelism on the compiler, feature needed for a textual programming language.

Another important feature about the Labview–Matworks linking is the possibility to reproduce Simulink projects in the LabVIEW development, just in a few steps. The NI LabVIEW Simulation Interface Toolkit gives control system design and test engineers a link between the NI LabVIEW graphical development environment and The MathWorks, Inc. Simulink software. With the LabVIEW Simulation Interface Toolkit, it is easily possible to build custom LabVIEW user interfaces to view and control your simulation model during run time. This toolkit also provides a plug–in for use with The MathWorks, so designers are allowed to connect, using LabVIEW, a model developed in the Simulink environment, to the real world through a variety of real–time I/O platforms. With these capabilities, you can easily take your models from software verification to real–world prototyping and hardware–in–the–loop simulation. This particular characteristic was exploited and described to implement the controller shown in the previous sections.

3 The Use of FPGA in Industrial Applications

In this section we will present an experimental set–up that has been used to show the benefit of the FPGA features, to be integrated with the simulation environment previously presented, in order to better validate the control laws described in [3]. This set–up implements a fist prototype of a supervisory tool for the TRIGA reactor RC–1 [26], in the Rome Casaccia Centre. In the first part, it will be shown how Simulink and LabVIEW are used to obtain a tool capable of simulating mathematical model described by differential equations, with the possibility of 'downloading' the resulting numerical model on a mixed CPU–FPGA architecture. LabVIEW allows users to create an intuitive and user friendly panel, very useful to have a real–time synoptic representing the model state. In the second part, we will focus on the possibility to push control algorithms to very high performances, describing VHDL code generation and digital circuits synthesis through FPGA family development tools.

3.1 A Digital Supervisory Tool using LabVIEW Development Environment

LabVIEW allows users to create an intuitive front panel easy to customize. An example of such front panel is given in Figure 38. In this application the pressurizer model previously described has been reproduced with the three heaters at its bottom. A similar synoptic loads the Simulink model and parameters and it visualize them in a more user–friendly way, thus resulting in a better perception of the process real estate. Beside the pressurizer, some analog indicators have been implemented such as pressure, power and temperature monitoring. During the simulation all the implemented elements are dynamically animated, showing the same behavior recorded with simple graphs of the variables time evolution. Just using the simulation interface toolkit simulations have become more realistic and in this direction an hypothetical step to download a similar architecture on a realtime system would be immediate.

In the following section, the second test application will be presented to underline the advantages from the use of CompactRIO. This test shows the importance of such a digital architecture in supervision and monitoring systems: in the so–called old generation nuclear plant a digitalization process will be desirable in order to improve the HMI and to have a user–friendly visualization beside the always indispensable analog instrumentation. In this sense some important variables have been acquired and

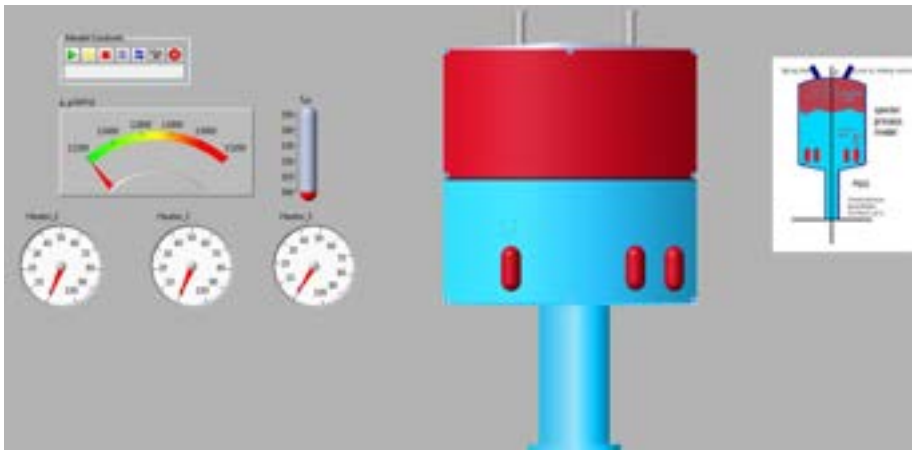

Figure 38: Synoptical view of the LabVIEW application loading the Simulink model

presented to the plant operator in a intuitive way, also with the possibility to record data in a dedicated memory and reproduce some particular plant conditions.

3.1.1 Test Bench: TRIGA RC–1 Reactor

TRIGA RC–1 is a pool thermal reactor having a core contained in an aluminum vessel and placed inside a cylindrical graphite reflector, bounded with lead shielding. The biological shield is provided by concrete having mean thickness of 2.2 m. Demineralized water, filling the vessel, ensures the functions of neutron moderator, cooling mean and first biological shield. Reactor control is ensured by four rods: two shims, one safety fuel–follower rods and one regulation rod. Produced thermal power is removed by natural water circulation through a suitable thermo–hydraulic loop including heat exchangers and cooling towers. Some irradiation facilities are listed below.

The core and the reflector assemblies are located at the bottom of an aluminum tank (190.5 cm diameter). The overall height of the tank is about 7 m, therefore the core is shielded by about 6m of water. The core, surrounded by the graphite reflector, consists of a lattice of fuel elements, graphite dummy elements, control and regulation rods. There are 127 channels divided in seven concentric rings (from 1 to 36 channels per ring). The channels are loaded with fuel rods, graphite dummies and regulation and control rods depending on the power level required. One channel houses the start–up Am–Be source, while two fixed channels (the central one and a peripheral) are available for irradiation or experiments. A pneumatic transfer system allows fast transfer from the peripheral irradiation channel and the radiochemistry end station. The diameter of the core is about 56.5 cm while the height is 72 cm. Neutron reflection is provided by graphite contained in an aluminum container, surrounded by 5 cm of lead acting as a thermal shield. The fuel elements consist of a stainless steel clad (AISI–304, 0.05 cm thick, 7.5

g/cm^3 density) characterized by an external diameter of 3.73 cm and a total height of 72 cm, end cap included. The fuel is a cylinder (38.1 cm high, 3.63 cm in diameter, 5.8 g/cm^3 of density) of a ternary alloy uranium–zirconium–hydrogen (H–to–Zr atom ratio is 1.7 to 1; the uranium, enriched to 20% in ^{235}U, makes up 8.5% of the mixture by weight: the total uranium content of a rod is 190.4 g, of which 37.7 g is fissile) with a metallic zirconium rod inside (38.1 cm high, 0.5 cm in diameter, 6.49 g/cm^3 of density). There are two graphite cylinders (8.7 cm high, 3.63 cm in diameter, 2.25 g/cm^3 of density) at the top and bottom of the fuel rod. Externally two end–fittings are present in order to allow the remote movements and the correct locking to the grid. The regulation rod has the same morphological aspect as the fuel rod: the only difference is that instead of the mixture of the ternary alloy Uranium–Zirconium–Hydrogen there is the absorber (graphite with powdered boron carbide). The control and safety rods are 'fuel followed': the geometry is similar to that of the regulation rod but with fuel element at its bottom. The graphite dummies are similar to a fuel element but the cladding is filled with graphite. Figure 39 shows an horizontal section.

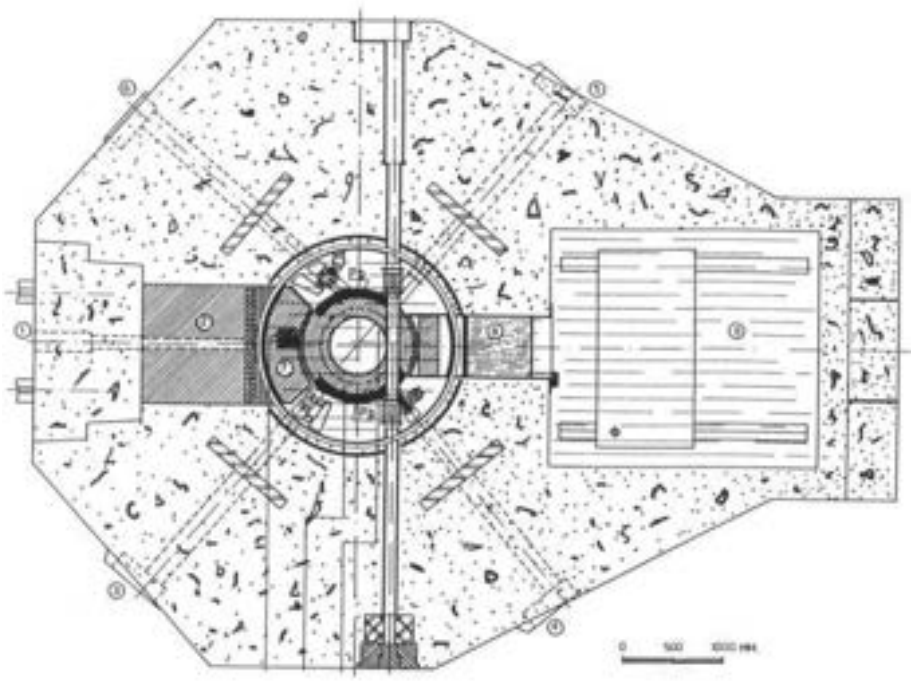

Figure 39: Horizontal section of TRIGA RC–1 nuclear research reactor at ENEA–Casaccia research Centre

The parameters used in order to perform the reactor monitoring can be classified into three large groups: power monitoring, process monitoring and radiological monitoring. The reactor power is monitored by means of one starting channel (0.0 to 1W), two wide range linear channels (0.5 -5.0o106 W) and one safety channel (10 kW to 1.1 MW). The process monitoring includes 6 temperatures (fuel elements,

primary and secondary loops, cooling towers), flow rates (primary and secondary loops, water cleaning system, reactor hall air), levels (reactor pool, shielding tank), conductivities (primary loop, shielding tank loop). The radiological control is carried out by monitoring water activity (primary and secondary loops), air activity (reactor hall and experimental channels) and environmental radiation levels (reactor hall, control room and experimental channels). Only main plant parameters are mentioned herein, but a lot of secondary parameters can be easily monitored (such as control rods positions, switches status, alarms and so on).

3.1.2 Analyzed Parameters

In this prototypical phase, in way of obtaining a complete knowledge about the plant and representing the operation of power increase, three kind of signals have been isolated

1. Variables representing generated power by the nuclear reactor.
2. Temperatures concerned with positions in the core and in the linked thermo–hydraulic loops.
3. Signals referred to rods position.

A logarithmic amplifier provides power value from the shutdown to the full power status, this is possible because the amplifier is linked to a ionization chamber covering a current range from 100 pA to 100μA. Variables coming from thermocouples are stored for a temperature monitoring. Conditioning of thermocouple signals is available for the input of our modules through transducers. The most important temperatures collected are process temperatures (well surface, purificator, towers entrance, exchangers entrance, etc.), and fuel temperatures recorded in two different positions of the core. An indication of rod positions (two shim rods, safety and regulation rod) is obtained through potentiometer repeater. On the front panel, numeric indicators presented on the control console are replicated, together with the luminous indicators representing: rod at full lower stroke and full upper stroke; rod uncoupled from the electromagnet. The system synoptic faithfully depicts a similar representation of the control panel on the console. Extraction and insertion (fixed–speed) of a selected rod is determined raising or lowering a lever, with a central equilibrium position. In the following picture the synoptic just described is shown.

At this stage, 45 different signals have been identified and sampled by the supervisory tool. The number of channels and the frequency of acquisition would not justify the choice of using a FPGA architecture: in fact, the real–time processor alone is presently able to guarantee a satisfactory result in terms of syncronism and parallelism of different process, but it could become obsolete increasing the number of signals and the rate of acquisition. In this sense, using an FPGA from scratch can satisfy

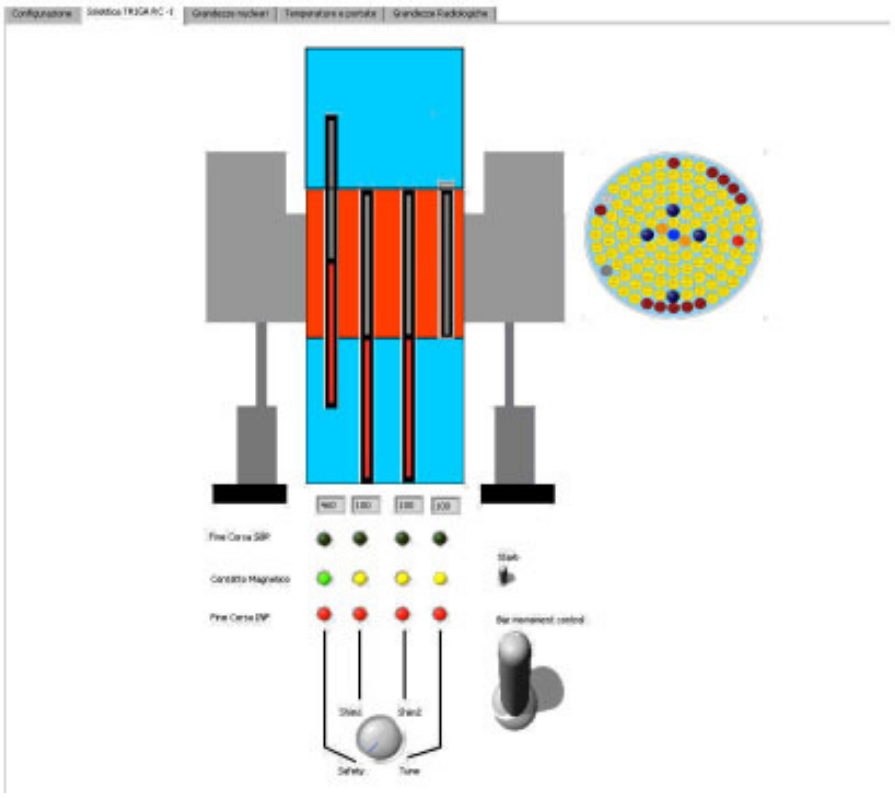

Figure 40: A Software Synoptic of the Real TRIGA RC–1 Control Console

the requirements of real–time acquisition and effective independence between processes also with more severe specs and facilitating a possible process of the system licensing.

3.1.3 Process on Investigation: Attainment of Maximum Power (1MW)

The plant process on investigation represents the transient state reaching the maximum power of operation. 1 MW top power is achieved by plant operators respecting some operating rules. Our digital system is able to monitor and save all of the user maneuvers, therefore in the following graphs main parameters development is shown. This process took 30 minutes to run out and in this lapse of time the prototype worked in parallel providing a user–friendly visualization, permitting to memorize in .txt files and offering its front panel on web server through the dedicated LAN.

The Figure 41 depicts the control rods rise, during the phenomenon formerly described. It is possible to recognized the correct sequence of rod rising as in a safe operating maneuver, in sequence: safety, regulation, shim1 and shim2 rod.

The power transient is shown from start–up to 1 MW state during the process. Two fuel elements, far–between in the core, recorded this kind of trend during this transient state, falling completely into fuel range of granted temperatures.

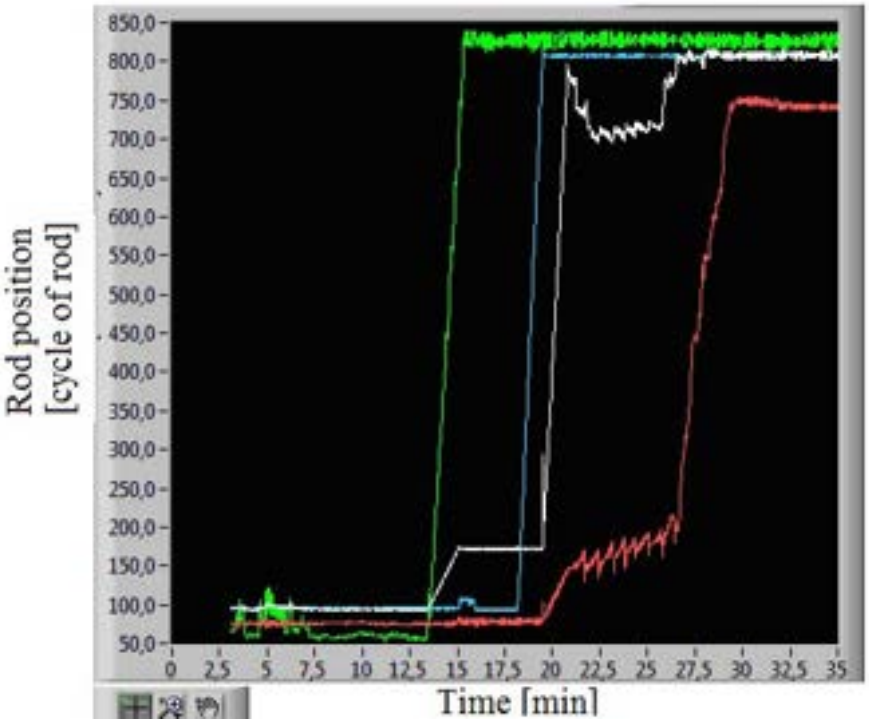

Figure 41: Transient State of Rod Position during Power Increase

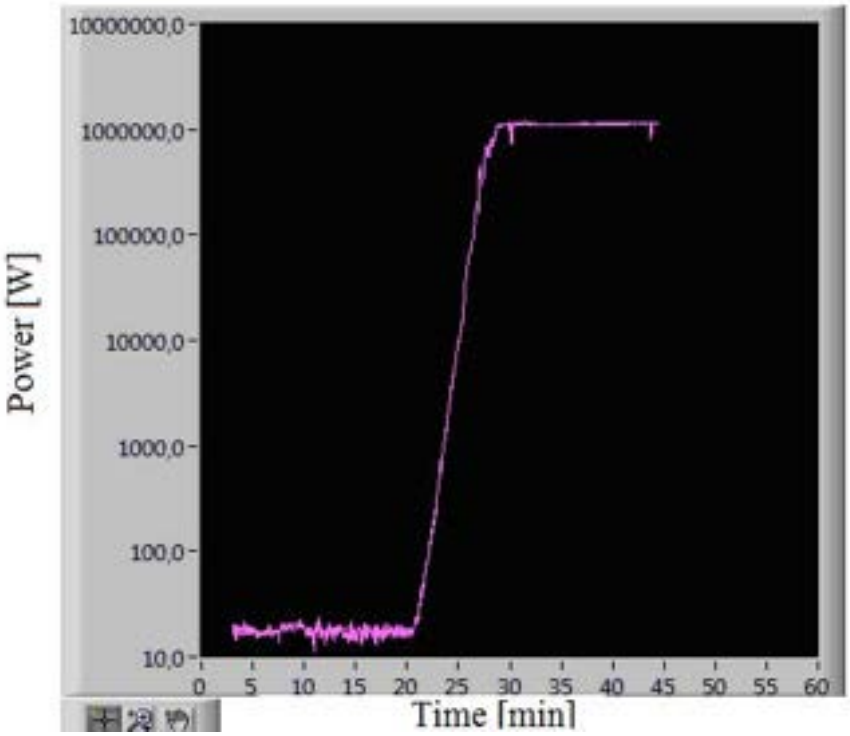

Figure 42: Power Evolution Reaching 1 MW vs time [min] – Power Values are Expressed in W

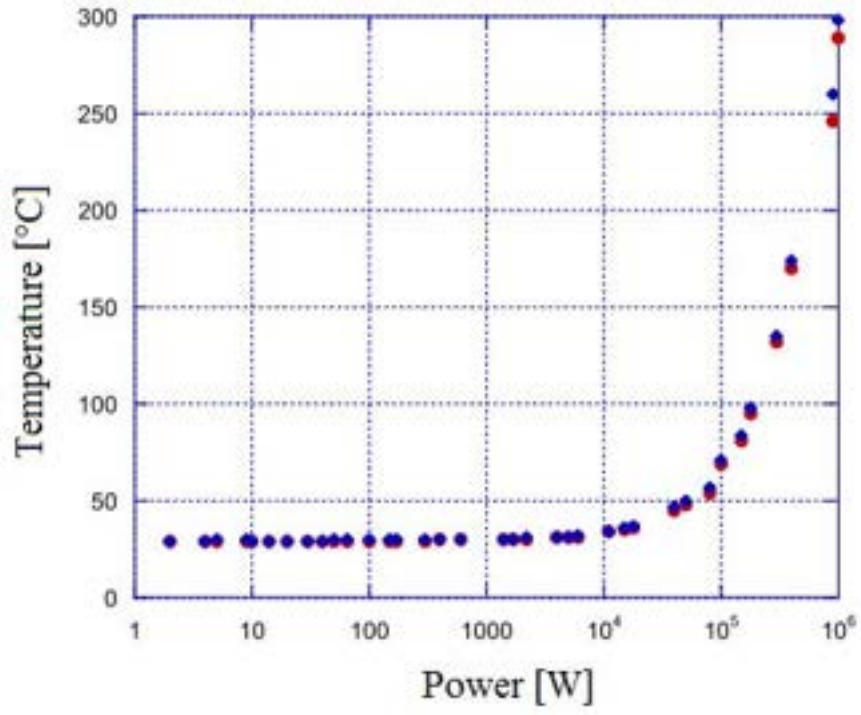

Figure 43: Fuel Temperature during Power Increase

A further useful tool, presented in the proposed supervisory system, is a dynamic visualization of the reactor synoptic: indeed an operator is able to check real–time rod positions in a very intuitive way compared to a digital indication of rod cycles. This user–friendly methods of displaying is conceived to facilitate the analogue to digital changeover, as required for plant console modernizations.

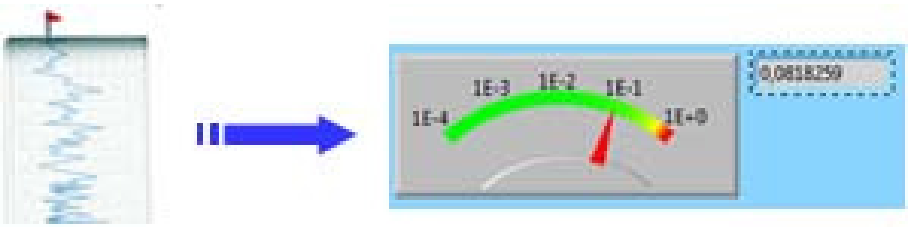

Figure 44: Analog to Digital Changeover using LabVIEW Tools

3.2 Generic Controller Implementation on FPGA–Based Platforms

In the previous sections we described how control algorithms can be simulated and then implemented on CPUs systems or hybrid CPU–FPGA architectures, achieving good performances in terms of time, programming complexity and graphical interface improvements. In this section we focus on the possibility to push control algorithms to very high performances, describing two different approaches: VHDL code generation and digital circuits synthesis through FPGA family development tools, as maintaining the compactRIO architecture with a lower level approach making use of the FPGA module. As a test application we defined a PID control loop algorithm in the different described design flows. It is good to underline the fact that we focused on the "PID core" implementation, presuming that problems such as signals conversion, conditioning and compatibility have already designed and solved.

3.2.1 Flow Design of Digital Architectures Using VHDL Code Generation

VHDL is a language for describing digital electronic systems. It arose out of the United States Government's Very High Speed Integrated Circuits (VHSIC) program, initiated in 1980. It became clear that there was a need for a standard language for describing the structure and function of integrated circuits (ICs). Hence the VHSIC Hardware Description Language (VHDL) was developed, and subsequently adopted as a standard by the Institute of Electrical and Electronic Engineers (IEEE) in the US. VHDL is designed to fill a number of needs in the design process. Firstly, it allows description of the structure of a design, that is how it is decomposed into sub–designs, and how those sub–designs are interconnected. Secondly, it allows the specification of the function of designs using familiar programming language forms. Thirdly, as a result, it allows a design to be simulated before being manufactured, so that designers can quickly compare alternatives and test for correctness without the delay and expense of hardware prototyping.

1. Describing structure

   A digital electronic system can be described as a module with inputs and/or outputs. The electrical values on the outputs arefunction of the values on the inputs. Figure 45 shows an example of this view of a digital system. The module F has two inputs, A and B, and an output Y. Using VHDL terminology, we call the module F a design entity, and the inputs and outputs are called ports. One way of describing the function of a module is to describe how it is composed of sub–modules. Each of the sub–modules is an instance of some entity, and the ports of the instances are connected using signals. Figure 45 also shows how the entity F might be composed of instances of entities G, H and I. This kind of description is called a structural description. Note that each of the entities

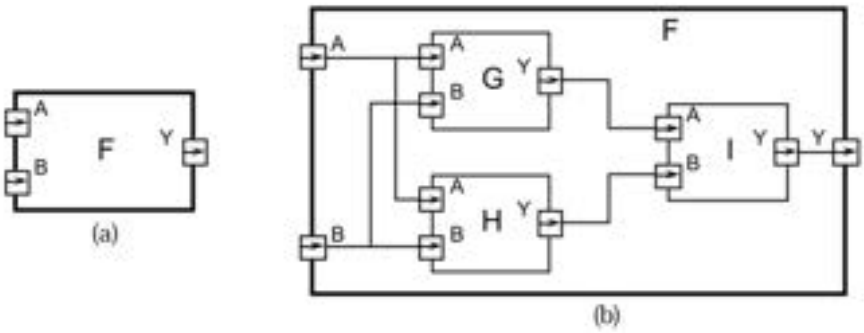

Figure 45: Example of a structural description

G, H and I might also have a structural description.

2. Describing Behaviour

   In many cases, it is not appropriate to describe a module structurally. One such case is a module which is at the bottom of the hierarchy of some other structural description. For example, if you are designing a system using IC packages bought from an IC shop, you do not need to describe the internal structure of an IC. In such cases, a description of the function performed by the module is required, without reference to its actual internal structure. Such a description is called a functional or behavioural description. To illustrate this, suppose that the function of the entity F in Figure 45 is the exclusive–or function. Then a behavioural description of F could be the Boolean function: $Y = \overline{A}.B + A.\overline{B}$

   More complex behaviours cannot be described purely as a function of inputs. In systems with feedback, the outputs are also a function of time. VHDL solves this problem by allowing description of behaviour in the form of an executable program.

Once the structure and behavior of a module have been specified, it is possible to simulate the module by executing its behavioral description. This is done by simulating the passage of time in discrete steps. At some simulation time, a module input may be stimulated by changing the value on an input port. The module reacts by running the code of its behavioral description and scheduling new values to be placed on the signals connected to its output ports at some later simulated time. This is called scheduling a transaction on that signal. If the new value is different from the previous value on the signal, an event occurs, and other modules with input ports connected to the signal may be activated. The simulation starts with an initialization phase, and then proceeds by repeating a two–stage simulation cycle. In the initialization phase, all signals are given initial values, the simulation time is set to zero, and each

module's behavior program is executed. This usually results in transactions being scheduled on output signals for some later time. In the first stage of a simulation cycle, the simulated time is advanced to the earliest time at which a transaction has been scheduled. All transactions scheduled for that time are executed, and this may cause events to occur on some signals.

In the second stage, all modules which react to events occurring in the first stage have their behavior program executed. These programs will usually schedule further transactions on their output signals. When all of the behavior programs have finished executing, the simulation cycle repeats. If there are no more scheduled transactions, the whole simulation is completed. The purpose of the simulation is to gather information about the changes in system state over time. This can be done by running the simulation under the control of a simulation monitor. The monitor allows signals and other state information to be viewed or stored in a trace file for later analysis. It may also allow interactive stepping of the simulation process, much like an interactive program debugger.

3.2.2 Example of a PID Digital Implementation through VHDL Code

In this section the PID algorithm described in deliverable1 is reproposed to show how it is immediately possible to create a digital core from a low level description as VHDL code. Firstly a new project in VHDL has been created using the same nomenclature adopted in deliverable1. The code is reported below:

```
library IEEE;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;
use IEEE.STD_LOGIC_1164.ALL;

entity PID_FSM is
  Port ( ADC_DATA : in  STD_LOGIC_VECTOR (15 downto 0);  --16 bit unsigned PID input
         DAC_DATA : out  STD_LOGIC_VECTOR (15 downto 0); --16 bit unsigned PID output
         mr,clk : std_logic);
end PID_FSM;

architecture Behavioral of PID_FSM is
    type state_type is (      Start,
                              CalculateNewError,
                              CalculatePID,
```

```
                              SOverload,
                              ConvDac
                              );

     signal state : state_type := Start;

     CONSTANT SetVal : integer := 33259;
     CONSTANT Kp : integer := 10;
     CONSTANT Ki : integer := 20;
     CONSTANT Kd : integer := 1;
     CONSTANT Kg : integer := 256;
begin
     states: process
     variable p,i,d : integer := 0;
     variable Output_Old : integer := 0;
     variable Error_Old : integer := 0;
     variable err: integer := 1000;
     variable out_value : integer := 1;
     variable sAdc : integer := 0 ;


     begin
     if (mr='0') then
          state <= Start;
     end if;
     wait until clk='1';
     case state is
          when Start =>
               sAdc := conv_integer(ADC_DATA);  --Get the input for PID
               DAC_DATA<= conv_std_logic_vector(out_value ,16);
               state <= CalculateNewError;
               Error_Old := err;  --Capture old error
               Output_Old := Out_value;    --Capture old PID output
```

```
        when CalculateNewError =>  --
             state <= CalculatePID;
             err := (SetVal-sAdc); --Calculate Error
        when CalculatePID =>
             state <= SOverload;
             p := Kp*(err);              --Calculate PID
             i := Ki*(err+Error_Old);
             d := Kd *(err-Error_Old);
             out_value :=  output_Old+(p+i+d)/2048; --Calculate new output
        when SOverload =>
             state <=ConvDac;
             if out_value > 65535 then
                  out_value := 65535 ;
             end if;
              if out_value < 1 then
                  out_value := 1;
             end if;
        when ConvDac =>           --Send the output to port
             DAC_DATA<= conv_std_logic_vector(out_value ,16);
             state <= Start;
             when others =>
             state <= Start;
             end case;
    END PROCESS states;

end Behavioral;
```

Once VHDL code is developed, the ALTERA tool “Quartus II” is used to synthesize the digital circuit: like other kind of FPGA design tools expanding a new project consists of some fundamentals phases. The first step is the “Analysis and Synthesis” step, where the VHDL code is check and compiled (on the basis of the chosen device); if this step is completed without any error, it is possible to display how the development environment has placed components to realize the circuit which is able to satisfy the main requirements. Using different tools can lead to obtain not equals results, because of the use of

different optimization algorithms. In this application the obtained circuit is depicted in the Figure 46, at register (RTL) level.

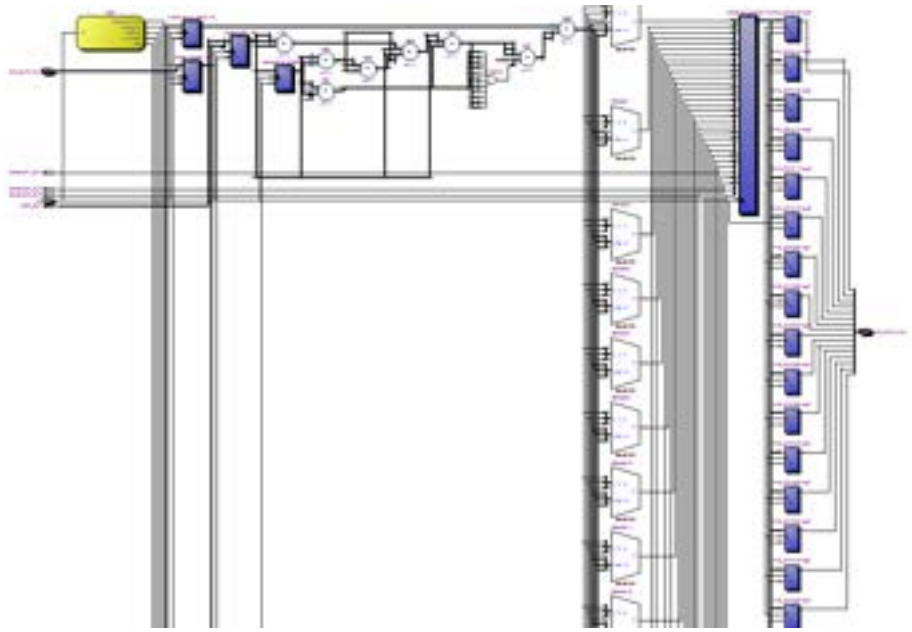

Figure 46: Register Transfer Level of main PID digital core implementation in FPGA

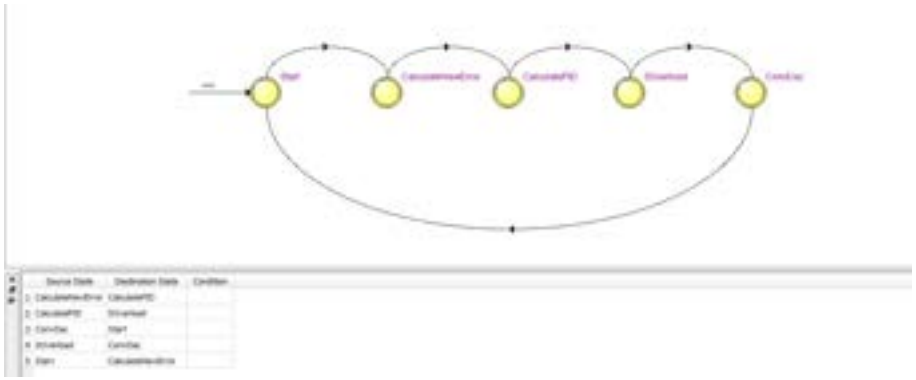

Figure 47: FSM visualization of PID digital core implementation in FPGA

At this stage, a compilation report is also available to check the FPGA total logic port usage and the maximum delay between input and output values in terms of clock edges. It is good rule to remember that in this phase this delays are evaluated just on the basis of the PID entity behavior and in the last phase the timing constraints of ports and real delays are considered.

3.2.3 Flow Design of Digital Architectures Using LabVIEW FPGA–Module

Alternatively to VHDL code description, a PID algorithm can be implemented maintaining the same National Instruments architecture adopted LabVIEW descriprion language [30]. As previously mentioned,

not only can you use compactRIO FPGA for high–speed acquisition and generation, but also to implement several control algorithms on the FPGA. Single–point I/O with multichannel can be exploited, tunable PID or other control approaches realize deterministic control with loop rates up to hundreds of kilohertz.

DMA channels can be used to stream high–speed data between the FPGA and real–time hardware. To create a DMA buffer for streaming data, just select on FPGA target a new FIFO. Giving the FIFO structure a descriptive name and choosing target–to–host as the type, the real–time processor is able to read from the FIFO those values stored by the FPGA. This means that data should flow through this DMA FIFO from the FPGA target to the real–time host. Data type can also be set and FPGA FIFO depths.

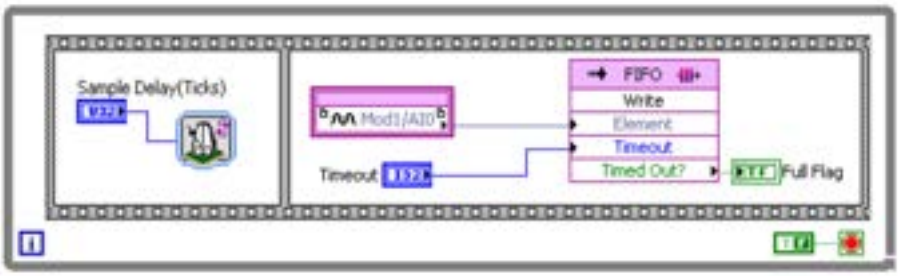

Figure 48: Simple MA transfer on One channel

It is fairly simple to put a DMA FIFO on a diagram. However, complexities arise when the default settings on the DMA transfer are not sufficient. If missing data points create a bug in your system, you must monitor the full flag on the FPGA and latch it when a fault occurs. Simply sampling this register from the host is not sufficient to catch quick transitions on that variable. Figure 49 shows various latching techniques on the timeout (full flag).

If you are receiving full flags, you need to either increase buffer size on the host, read larger chunks on the host, or read faster on the host. Keep in mind that many control and monitoring applications need only the most up–to–date data. Therefore losing data may not be an issue for a system as long as it returns the most recent data when called.

Another consideration for DMA transfer is using one DMA FIFO for multiple channels. Hybrid mode CompactRIO systems only have one DMA channel available. To pack multiple channels into one DMA FIFO, use an interleaving technique, and unpack using decimation on the host.

To read DMA channels from a realtime program, a reference to the FPGA VI or bitfile and FPGA target is needed to be specify. Then the modality to read a value from or write a value to a control or indicator in the FPGA VI on the FPGA target is set. This can be a trigger condition, sampling rate, or

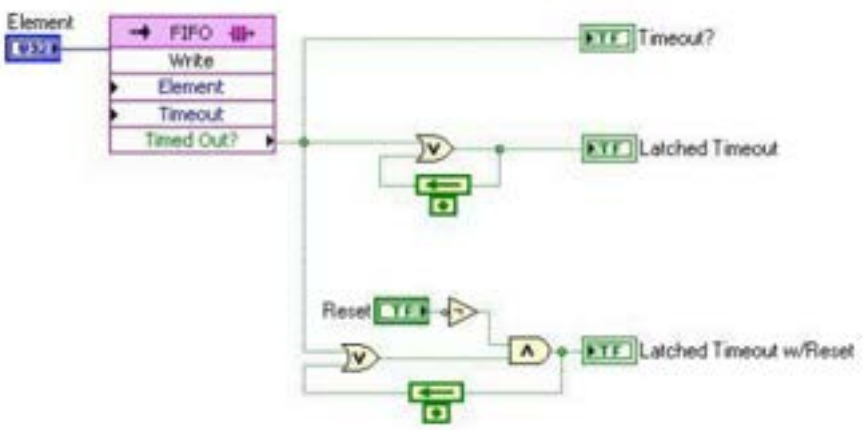

Figure 49: Example of No Latch, Simple Latch, and Latch with Reset

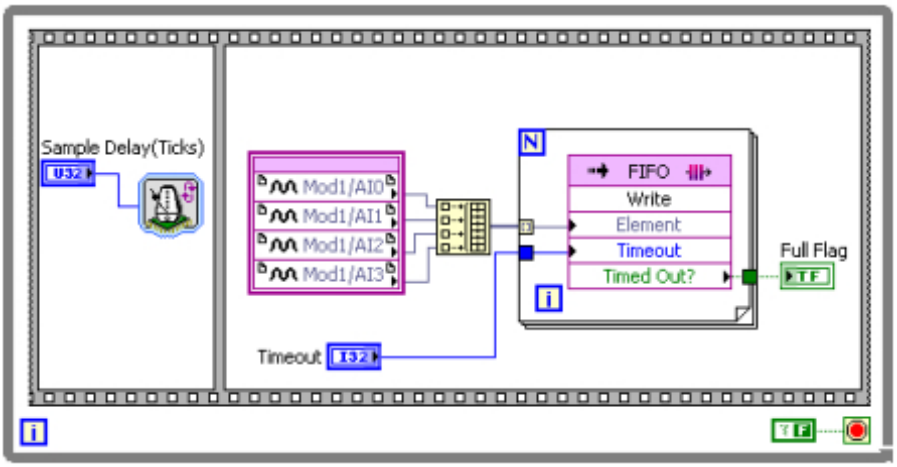

Figure 50: interleaved multichannel data stream

any other data or parameter set by a control or indicator in the FPGA VI. Invoking an FPGA interface method or action from a host VI on an FPGA VI is essential to implement the following operations: download, abort, reset, and run the FPGA VI on the FPGA target; wait for and acknowledge FPGA VI interrupts; read DMA FIFOs; and write to DMA FIFOs. The methods a user can choose from depend on the target hardware and the FPGA VI. It is necessary to wire the FPGA VI Reference input to view the available methods in the shortcut menu. On the realtime applicative we have to open a reference to the FPGA VI, set parameters, use the Invoke Node in a task loop to read waveform data, and close the FPGA reference to implement the simplest DMA read. However, as previously discussed, complexities can arise from buffer sizes, timeouts, and synchronization. Below is a simple real–time application that reads a waveform from the FPGA, performs an average calculation on the waveform and passes the data to a separate control loop that is running a PID loop to control a PWM output based on the waveform

average. This type of application might be used to control a signal generator or laser that is tuned using a PWM input signal.

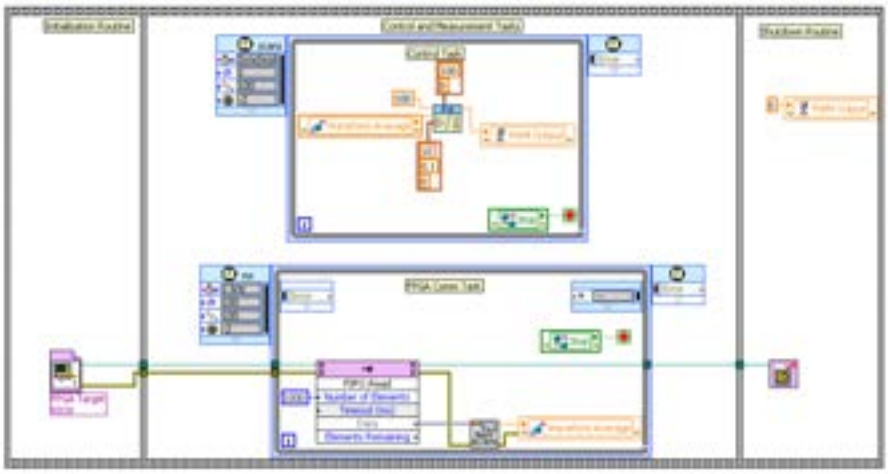

Figure 51: Simple DMA Read Using the FPGA Interface on the Real–Time Host

4 Conclusions

In this deliverable, a Simulink simulation environment has been used to check the performance of the designed digital controllers, for various values of the sampling time. Moreover, LabVIEW has been analyzed as potential solution to interface Simulink with real–time environments. This allows testing the controllers with some hardware devices, such as sensors and actuator, in the control loop. Although not certified for nuclear applications, this allows checking the methodological steps for the real–time prototyping of the controllers for non–nuclear industrial applications. For nuclear applications, more costly nuclear–certified softwares have to be considered. Such Simulink/LabVIEW simulation environment can be integrated in the next future with an experimental set–up, here proposed to show the benefit of the FPGA features.

References

[1] S. Di Gennaro and B. Castillo–Toledo, Comparative Study of Controllers for the Supervision, Control and Protection Systems in Pressurized Water Reactors of Evolutive Generation, Deliverable 2, PAR2010 Project, 2011.

[2] S. Di Gennaro and B. Castillo–Toledo, Performance Study of the Control Systems in the presence of Faults and/or Reference Accidents in Pressurized Water Reactors of Evolutive Generation, Deliverable 2, PAR2010 Project, 2011.

[3] S. Di Gennaro, B. Castillo–Toledo, and F. Memmi, Study, Design and Realization of Supervisory, Control and Protection Systems for Performance and Safety Improvements of Novel Nuclear Plants, Deliverable 1, PAR2011 Project, 2012.

[4] Areva, *U.S. EPR Nuclear Plant – The Path of Greatest Certainty*, Areva 2007.

[5] Areva, *Design Control Document "U.S. EPR Final Safety Analysis Report"*, available at US–NRC, 2008.

[6] B. Castillo–Toledo, M. Cappelli, S. Di Gennaro, and M. Sepielli, Pressurizer Pressure Control in PWRs of New Generation, *Proceedings of the* 8^{th} *Internatonal Topical Meeting on Nuclear Plant Instrumentation, Control, and Human–Machine Interface Technologies*, San Diego, CA, USA, 2012.

[7] A. Crabtree, M. and Siman–Tov, *Thermophysical Properties of Saturated Light and Heavy Water for Advanced Neutron Source Applications*, Oak Ridge National Laboratory, 1993.

[8] Cranfor, III et al, Device and method for measuring temperature of a liquid contained in a pressurizer vessel, United States Patent N. 5426677, June 20, 1995.

[9] F. Champomier, *PCSR – Sub–chapter 5.4 – Components and Systems Sizing*, UKEPR–0002–054 Issue 02, AREVA NP & EDF, June 19, 2009.

[10] J. Freixa, and A. Manera, Verification of a TRACE EPRTM Model on the Basis of a Scaling Calculation of an SBLOCA ROSA Test, *Nuclear Engineering and Design*, Vol. 241, pp. 888–896, 2011.

[11] Hou, D. Lin, M. Xu, Z.–H. Yang, Z. Y.-h. Chen, Expansion of Relap5 Control and Protection Simulation Functions with Simulink, *Nuclear Power Engineering*, Hidongli Gongcheng, Cina, Vol. 28, Part 6, pp. 112–115, 2007.

[12] N. Larsen, *Simulation Model of a PWR Power Plant*, Riso National Laboratory, DK–4000, Roskilde, Denmark, 1987.

[13] J. Lienhard, and J. Lienhard, *A Heat Transfer Textbook*, 3^{rd} Edition, Phlohiston, Press Cambridge University, 2003.

[14] Meng Lin, Dong Hou, Zhihong Xu, etc., System Simulation of Nuclear Power Plant by Coupling Relap5 and Matlab/Simulink., 14th International Conference on Nuclear Engineering, Miami, Florida, USA, pp. –, 2006.

[15] Mitsubishi Heavy Industries Ltd, *Design Control Document for the US–APWR*, Chapter 7 – Instrumentation and Controls, MUAP–DC007, Revision 3, March 2011.

[16] N. Muellner, M. Lanfredini, Bases for Setting up a Thermal–Hydraulic Model of a Generic PWR Pressurizer, including Controls, Agreement N.IN.E S.r.l. – University of L'Aquila, 2011.

[17] G. Petrangli, *Nuclear Safety*, 2005.

[18] Z. Szabó, P. Gárspár, and J. Bokor, Reference Tracking of Wiener Systems Using Dynamic Inversion, *Proceeding International Symposium on Intelligent Control*, Limassol, Cyprus, on CD, 2005.

[19] N. E. Todreas, and M. S. Kazimi, *Nuclear Systems I – Thermal Hydraulic Fundamentals*, Hemisphere Publishing Cooperation, 1990.

[20] N. E. Todreas, and M. S. Kazimi, "Nuclear Systems II – Elements of Thermal Hydraulic Design", Hemisphere Publishing Cooperation, 2001.

[21] L. S. Tong, and J. Weisman, *Thermal Analysis of Pressurized Water Reactors*, 3^{rd} Edition, ANS, 1996.

[22] J. L. Tylee, Bias Identification in PWR Pressurizer Instrumentation using Generalized Likehood–Ratio Techquique, *Proceedings of the 1981 American Control Conference*, Charlottesville, Virginia USA, Tech. Report N. CONF–810605–1, 1981.

[23] UK–EPR, *Fundamental Safety Overview, Vol. 2: Design and Safety, Chapter E: the Reactor Coolant System and Related Systems.*

[24] Research Reactor Group, Research reactor Modernization and Refurbishment, *IAEA* Progress Report, 2009.

[25] E. Bachmach, O.Siora, V.Tokarev, S. Reshetytsky, V. Kharchenko, and V. Bezsalyi, FPGA–Based Technology and Systems for I&C of Existing and Advanced Reactors, *IAEA–CN–164–7S04*, 2010.

[26] F. Memmi, O. Aronica, R. Bove, M. Cappelli, L. Falconi, M. Palomba, E. Santoro, and M. Sepielli, A High Operability Supervisory Digital System for TRIGA–Type Research Reactors, *Proceedings for the European Research Reactor Conference – RRFM*, 2010.

[27] CompactRIO Developers Guide, *Recommended LabVIEW Architectures and Development Practice for Machine Control Applications*, National Instruments, 2011.

[28] F. Memmi, R. Bove, M. Cappelli, L. Falconi, M. Palomba, E. Santoro, and M. Sepielli, A Preliminary User–Friendly, Digital Console for the Control Room Parameters Supervision in Old–Generation Nuclear Plants, *Proceedings for the International Congress on Advances in NPPs – ICAPP*, 2012.

[29] F. Memmi, R. Bove, M. Cappelli, L. Falconi, M. Palomba, and M. Sepielli, The Role of FPGA–based Architectures in the Control Room Modernization Process: Preliminary Results of a Case–Study, *ICONE20*, 2012.

[30] Press LCC, *LabVIEW Advanced Programming Techniques*, 2001.