SLAC-TN-10-025
August, 2010

Non-Linear Time-Domain Simulations of the RF Station-Beam Dynamics Interaction for the LHC

T. Mastorides, C. Rivetta

August 2010

Abstract

Non-linear time-domain simulations have been developed for the Positron-Electron Project (PEP-II) and the Large Hadron Collider (LHC). These simulations capture the dynamic behavior of the RF station-beam interaction and are structured to reproduce the technical characteristics of the system (noise contributions, non-linear elements, and more). As such, they provide useful results and insight for the development and design of future LLRF feedback systems. They are also a valuable tool for the study of diverse longitudinal beam dynamics effects such as coupled-bunch impedance driven instabilities and single bunch longitudinal emittance growth. Results from these studies and related measurements from PEP-II and LHC have been presented in multiple places [1], [2], [3], [4], [5], [6], [7]. This report presents an example of the time-domain simulation implementation for the LHC.

Work supported in part by US Department of Energy contract DE−AC02−76SF00515.

SLAC National Accelerator Laboratory, Menlo Park, CA 94025

Contents

1 System Description

The LHC RF system consists of 8 RF stations per beam. The RF system accelerates the beam during the ramp, compensates the small energy losses during coasting, and also provides longitudinal focusing. The beam and the RF station are two dynamic systems with a strong interaction, which complicates stability considerations for the composite system. A simplified block diagram of the LHC RF system is shown in Figure 1.

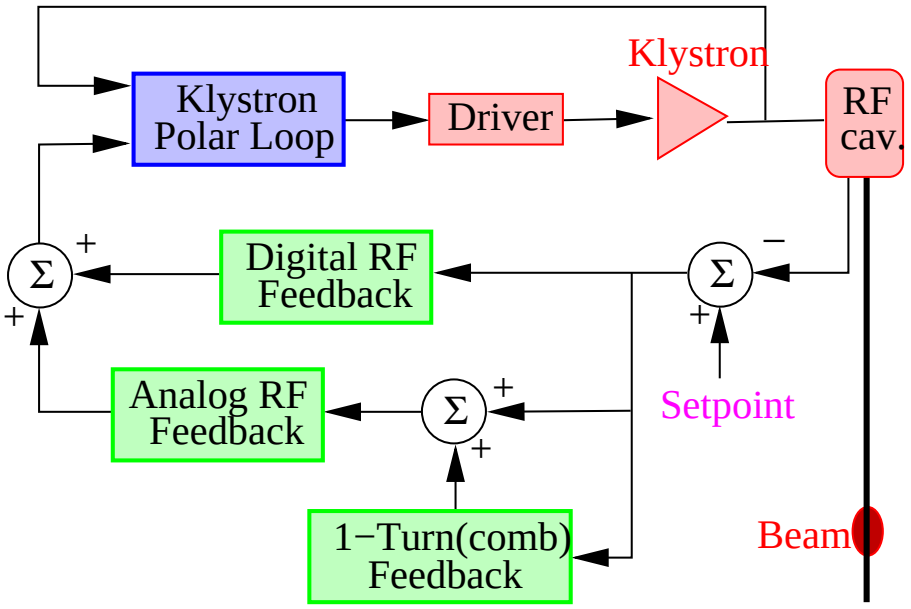

Figure 1: Simplified LHC RF block diagram.

Each RF station includes an accelerating super-conducting cavity, a 330 kW klystron, and the LLRF system consisting of the klystron polar loop and the impedance control feedback system. Figure 2 shows the LHC RF Cavern including the super-conducting cavities, klystrons, and LLRF [1]. The super-conducting cavity has an R/Q of 45, a resonance frequency of 400.8 MHz, and a mechanical cavity tuner with a range from -100 to 0 kHz. The cavity voltage and loaded quality factor Q_L are set to 1 MV and 20,000 respectively during injection and to 2 MV and 60,000 during physics, for nominal intensity beams. An LHC super-conducting cavity is shown in Figure 3. The klystrons used at LHC are inherently non-linear. The klystron polar loop used at the LHC acts around the klystron to reject power supply perturbations and compensate the gain and phase shift of the non-linear klystron at low frequencies for different operation points. The feedback system controls the accelerating fundamental impedance of the RF station to achieve longitudinal stability. It

[1] Figures 2, 3, and 4 are LHC pictures by Dan Van Winkle

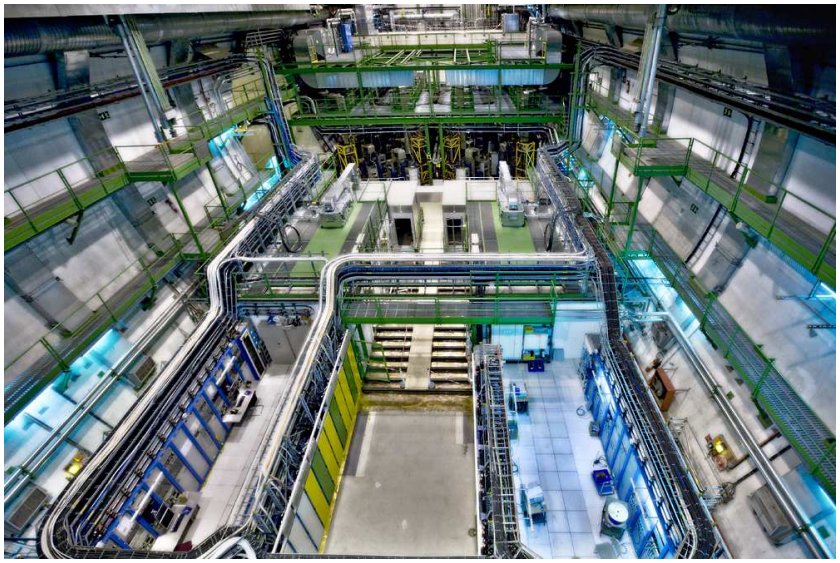

Figure 2: RF Cavern.

incorporates digital and analog paths, as well as the 1-Turn feedback (comb), which acts to reduce the impedance at the synchrotron sidebands. A detailed block diagram and description of the LHC LLRF components can be found in [8]. The LLRF crates are shown in Figure 4.

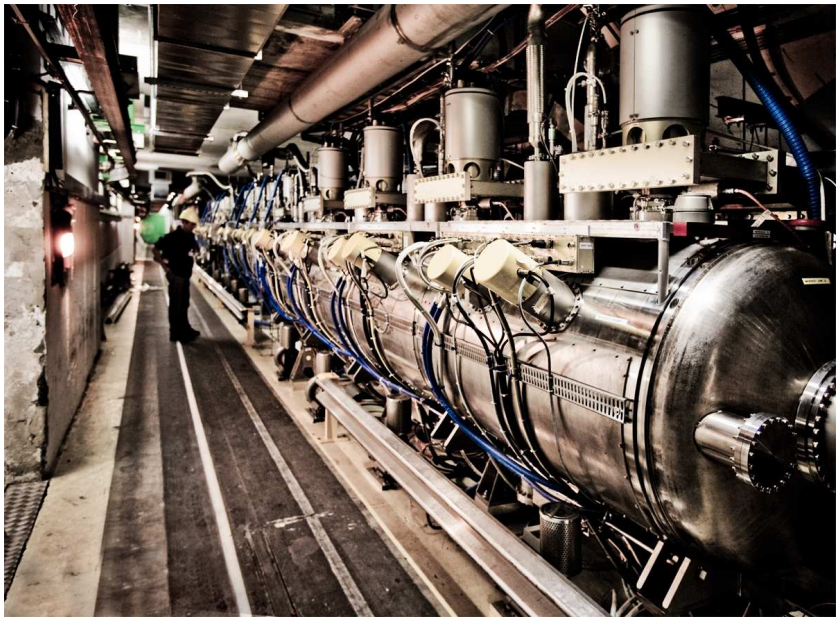

Figure 3: LHC super-conducting cavity.

2 Simulation and Model Description

The components modeled in our simulation model include

- the accelerating super-conducting cavity with an R/Q of 45 and a resonance frequency of 400.8 MHz,
- the 300 kW klystron,
- the klystron polar loop,
- the impedance controlling feedback (both digital and analog paths), including quantizing effects in the digital 1-turn feedback system
- cable and processing delays.
- and the beam.

To accurately describe the system, reduced models of the individual components are included in the simulation. The waveguide, cable, and processing delays are included, and the gains and phases of the RF feedback components are adjusted in a similar manner as for the real machine. The 1-Turn feedback (comb), which acts to reduce the impedance at the synchrotron sidebands, has not yet reached the hardware commissioning phase for all RF stations, but will be validated when data is available during commissioning.

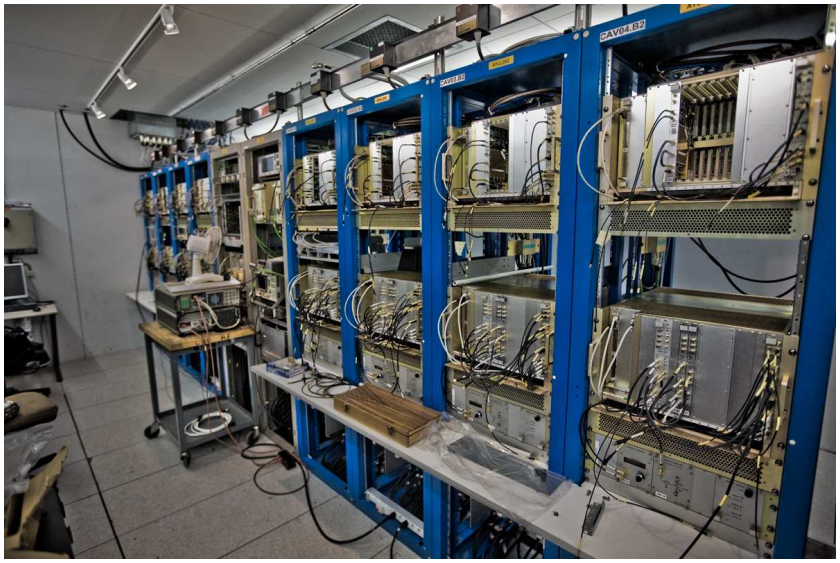

Figure 4: LHC LLRF crates.

The simulation is developed as a block system in Simulink. The top level Simulink block system for LHC is shown in Figure 5. The slow loop dynamics (software control regulators) are sufficiently slow compared to the time scale of the simulation that they are calculated as initial conditions (constants) in Matlab.

An earlier simulation effort by J. Holma [9] had been used to study and set-up the loops on a full-scale test bunch that included the LLRF, klystron, and cavity. This initial model did not include the beam dynamics, but certain components were incorporated in this work. The Matlab algorithms for the LHC implementation of the time-domain simulation will be presented in Section 3.

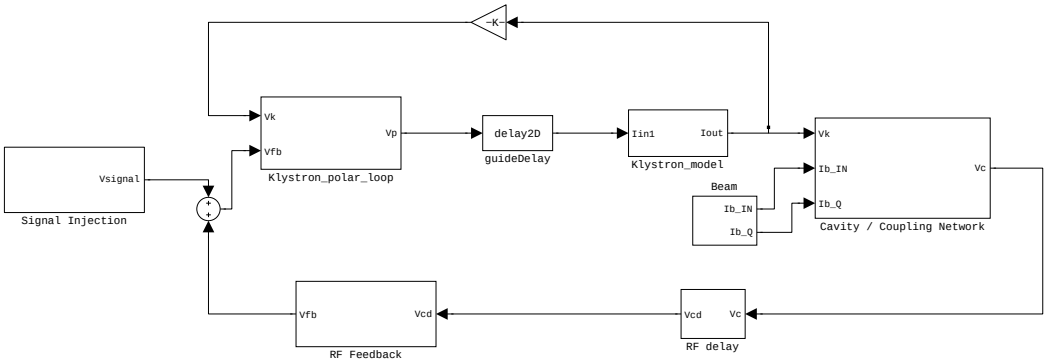

Figure 5: Time-domain Simulation: Simulink Block Diagram for the LHC.

3 Matlab Code

The time-domain simulation is initialized using file "INI_LHC.m". This file calculates all the operational parameters for the models of the RF components and also their initial conditions. This file calls on "set_FB_prm.m", "inte_cic.m", and "load_klystron_C.m".

"set_FB_prm.m" is introduced for operational flexibility. It defines the variable RF and LLRF parameters. "inte_cic.m" includes an interpolator for the noise injection techniques used for the identification of RF parameters [10]. "load_klystron_C.m" is used to incorporate the operational variations due to the klystron saturation curve and transfer function. It is based on data from an installed LHC klystron.

Multiple additional algorithms are used to process the data from the simulation and estimate beam dynamics effects. Due to the very diverse applications and their size, they are not included here.

3.1 INI_LHC.m

```
% INI_LHC.m
% LHC time-domain simulation: initialization file
%
% Copyright 2008 T. Mastorides, C. Rivetta

% Accelerator parameters
```

```
h = 35640;
wo = 2*pi*400.8e6;
frev = wo/(2*pi*h);
chirp_on = 0; % Noise injection: set to 1 for chirp, otherwise noise

% Set adjustable feedback parameters
feedback = set_FB_prm;
SET_OL = feedback.OL; % set to 1 for Open Loop

% Measurement type selection. Accelerator parameters are loaded
% respectively
switch feedback.meas
    case 1
        fit_prm = [];
    case 2
        load prm_nulled
    case 6
        load prm_notch
    case 3
        load prm_klystron
    case 5
        load prm_OL
    case {8,9}
        load DataJan10/prmB
        fit_prm = prm.fit_prm;
end
prm.fit_prm = fit_prm;

Vc = feedback.Vc; % Cavity voltage
wd = feedback.wd; % Cavity detuning
Ql = feedback.Ql; % Loaded Q
prm.system.measurement = feedback.meas;

% Comb parameters
prm.OTFdbk.gain = feedback.OTFdbk.gain*feedback.OTFdbk.switch;
prm.OTFdbk.phase = pi/180*feedback.OTFdbk.phase;
prm.OTFdbk.by_pass = feedback.OTFdbk.by_pass;
prm.OTFdbk.switch = feedback.OTFdbk.switch;

IB = feedback.IB; % Current

% Analog/Digital Loop
prm.rffbI.AnalogAttn = feedback.fdk.att;
prm.rffbI.DigitalAttn = feedback.fdk.att;
prm.rffbI.Command_digitalFdbkOn = feedback.digital.switch;
```

```
prm.rffbQ.Command_digitalFdbkOn = feedback.digital.switch;

% Set constant feedback parameters
prm.setpoint.phases_vcav = 0;
prm.system.wrf = 2*pi*400.8e6;
prm.system.Trev = 1/frev;
prm.rffbI.Command_digitGain = 6;
prm.rffbI.Command_digitIrrFilter = 14;
prm.OTFdbk.LPF.Coeff_default = [0.0767 0.1425 0.1127 0.0724 0.0321 ...
    0.0010 -0.0158 -0.0468 zeros(1,8)];
prm.OTFdbk.Eq.Coeff_default = [0.5 zeros(1,15)];
prm.OTFdbk.bandwidth = 1; % corresponds to alpha = 15/16
prm.system.h = h;
prm.system.Ts = 1/40.08e6;

% Simulink Data
Ts1 = prm.system.Ts;        % 40 MHz sampling
Ts2 = Ts1*16;               % 2.5 MHz sampling
TsC = Ts1*4;                % 10 MHz sampling for the comb filter

% Controller Attenuator calibration
AttLoc = 5; % 5 in simulation correponds to -6 in machine (-7 == -16)
nGa = 10^(AttLoc/20)*33e-3/sqrt(4e7);

% Chirp data
chirpGain = 1e-5;
fmin = 2e5; % fmin for chirp signal
fmax = 3e5; % fmax for chirp signal
cyclesChirp = 5e4; % simulink cycles

% Noise Injection data
noiseGain = feedback.noiseGain;
pr = feedback.pr;

lenV = pr;
ts_drive = Ts1*2^pr;  %ts_drive: is the sample rate of the playback

% Noise injection files
load('i_noise_BL.mat');
load('q_noise_BL.mat');
i_noise = inte_cic(i_dat,lenV,ts_drive);
i_noise_time = [(0:Ts1:(length(i_noise)-1)*Ts1)' i_noise];
q_noise = inte_cic(q_dat,lenV,ts_drive);
q_noise_time = [(0:Ts1:(length(q_noise)-1)*Ts1)' q_noise];
```

```
if chirp_on == 1
    cycles = cyclesChirp;
    noiseGain = 0;
else
    con = floor(log2(length(i_noise)/3564));
    cycles = 2^con*(h/10);
    chirpGain = 0;
end

% Temporary constants for reference and beam
Ib_I = 0;
Ib_Q = IB;
Ib_IC = [Ib_I Ib_Q]';

%-----------------------------------------------------------------
% SuperConducting Cavity
%-----------------------------------------------------------------

% Model Parameters

delta_wd = 0;
wr = wo+wd-delta_wd;
R_Q = 45;
Lo = R_Q/wr;
Co = Lo/45^2;
Rl = 45*Ql;
Ro = R_Q*Ql*1e3;
Rs = Rl;

% Assuming Zo = 50ohms, for the measurement of klystron power
Zo = 50;
N = sqrt(Rs/Zo);
N = sqrt(2)*N;  % Include a conversion Vkl(rms) to Vkl(peak)

% Cavity Model

A = .5*[-1/Ro/Co wo-wr^2/wo; -wo+wr^2/wo -1/Ro/Co];
B = .5*[1/Co 0; 0 1/Co];
Cd = eye(2);
Dd = zeros(2,2);

[Ad,Bd] = c2d(A,B,Ts1);

%-----------------------------------------------------------------
% Klystron + Drivers
```

```
%-----------------------------------------------------------------

% Klystron Transfer Function - Parameters

wrb = 2*pi*404.8e6;
Qr = 1100;
Gr = 10;
G_Q = Gr/Qr;
Lr = G_Q/wrb;
Cr = Lr/G_Q^2;
Rr = Gr;

% Klystron Transfer Function - I-Q Model

Akl = .5*[-1/Rr/Cr wo-wrb^2/wo; -wo+wrb^2/wo -1/Rr/Cr];
Bkl = .5*[1/Cr 0; 0 1/Cr];
Ckl_d = eye(2);
Dkl_d = eye(2);

[Akl_d,Bkl_d] = c2d(Akl,Bkl,Ts1);

% Adjustment of Static Characteristic

G_o = Ckl_d * (-Akl)^(-1) * Bkl + Dkl_d;
gain_factor = sqrt(G_o(1)^2 + G_o(2)^2);
phase_factor = atan2(G_o(2),G_o(1));

[a, pb] = load_klystron_C(gain_factor, phase_factor, 1);

Sat_up_limit = 67; % Maximum Klystron input rms voltage (Saturates
% output to prevent klystron saturation
Sat_low_limit = 0; % Minimum Klystron input rms voltage

Gdr = 280;

% RF Guide

GuideDelay = 200e-9;

Ncells = round(GuideDelay/Ts1);
prm.guideDelay.N = Ncells;

% Klystron Noise

klystron_error_A_eps = 0;
```

```
klystron_error_A_freq = 0;
klystron_error_A_phase_init = 0;
klystron_error_phase_eps = 0;
klystron_error_phase_freq = 0;
klystron_error_phase_phase_init = 0;

%_____________________________________________________________________
% RF feedback Loop (PI + Comb Filter)
%_____________________________________________________________________

% Closed Loop _________________________
% PI loop parameters
% Analog RF feedback (Proportional)

%Goo (Analog gain, high freq. gain)
Ga =3.5*(10^(-prm.rffbI.AnalogAttn/20));

% Digital Rf feedback (Integral)

Gi = (0.45/Ga)*(2^prm.rffbI.Command_digitGain)*...
    (10^(-prm.rffbI.DigitalAttn/20));      % Integral Gain
if feedback.digital.switch == 0;
    Gd = 1e-8;
else
    Gd = Ga*Gi;
end
phi = 0;    % Integral Phase in degrees

% Analog RF feedback
% Proportional

num = Ga*[1 0]; den = [1 1e4]; % tauA
prop = tf(num,den);

propIQ = append(prop,prop);

ana = propIQ;

anaD = c2d(ana, Ts1);

[Aana, Bana, Cana, Dana] = ssdata(anaD);

% Digital RF feedback (Integral)

Ki = Gd*[cosd(phi) -sind(phi); sind(phi) cosd(phi)]; % Gain Matrix
```

```
k = prm.rffbI.Command_digitIrrFilter; % 10,12 or 14
b = 1 - 2^(-k);

Aint = [b 0; 0 b]; Bint = [(1-b) 0; 0 (1-b)];
Cint = Aint;   Dint = Bint;

Bdig = Bint*Ki;
Ddig = Dint*Ki;

digD = ss(Aint, Bdig, Cint, Ddig, Ts1);

% Attenuation and Delay

Vc_max = 2e6; Pin_RF = 10e-3; Vin_RF = sqrt(Pin_RF*50);
FbAtt = Vin_RF/Vc_max;  % Cavity voltage Attenuator.

RFdelay = 400e-9;

Ncells = round(RFdelay/Ts1);
prm.RFdelay.N = Ncells;

%-----------------------------------------------------------------
% Comb Filter
%-----------------------------------------------------------------

% FIR coefficients
ntaps = 15;
ftaps = [0 .865 1.4 1.65 1.85 2.2 2.57 2.85 3.23 3.6 3.92 4.3 4.7 5]/5;
ataps = sqrt([1 .55 0 .008 0 .005 0 .0032 0 .002 0 .0013 0 0]);

FIRcoefs = firpm(ntaps,ftaps,ataps);
FIRcoefs = FIRcoefs/sum(FIRcoefs);

% Delays
time_delay = feedback.OTFdbk.delay;
coarse_delay = floor(time_delay/TsC);
fine_delay = round((time_delay-coarse_delay*TsC)/Ts1);
if fine_delay == 0
    fine_delay = 4;
    coarse_delay = coarse_delay - 1;
else
end
prm.coarseDelay.N = coarse_delay;
prm.fineDelay.N = fine_delay;
```

```
% Comb filter block
prm.OTFdbk.N = 891;   % 3564 for real impementation. Reduced by
% a factor of 4 by placing the filter inside the rate transition
prm.OTFdbk.alpha = 15/16;

%------------------------------------------------------------------
% Modulator
%------------------------------------------------------------------

load ATT_mod

Vc_att_max = 0;  Vc_att_min = -2.5;

MASTER_Ampl = 1;  % Master Oscillator AMPL

OSC_gain = MASTER_Ampl;

%------------------------------------------------------------------
% Klystron Polar Loop
%------------------------------------------------------------------

% Polar Loop Parameters

polar_loop_gain_setting = 1;
polar_loop_phase_setting = 0;
polar_loop_scaling_factor_gain = -.005;
polar_loop_scaling_factor_phase = .003;

% Attenuation Ic FWD signal

Pkl_max = 300e3; Pin_PL = 10e-3;
Coupl_FWD = sqrt(Pin_PL/Pkl_max);

% Cordic Klystron FBK

Delay_Cordic = 400e-9;

Ncells = round(Delay_Cordic/Ts1);
prm.cordicDelayK.N = Ncells;

% Cordic Feedback signal

prm.cordicDelayF = prm.cordicDelayK;
```

```
% Klystron Static gain estimation

Min_divisor_value = 1e-15;   % Parameter for simulink block

% CIC Model

prm.CIC = CIC_filter(0); % For Amplitude/phase loops the initial
% condition is set to zero.

% Rate transition from 40MHz to 2.5Mhz

% Integral in Polar Loop

% Interpolators - Amplitude / Phase Loops

Delay_Interpolator = 700e-9;

Ncells = round(Delay_Interpolator/Ts1);
prm.AInterpolatorDelay.N = Ncells;

prm.phiInterpolatorDelay = prm.AInterpolatorDelay;

%_____________________________________________________________________
% Initial Conditions
%_____________________________________________________________________
%
% ____________________________________________________
% Klystron + Drivers
% ____________________________________________________

Vcav_IC = Vc;
It_IC = inv(Bd)*(eye(2)-Ad)*Vcav_IC;
Ig_IC = It_IC - Ib_IC;
Vkl_IC = (Rs*Ig_IC + Vcav_IC)/N/2;

Akl_IC = abs(Vkl_IC(1) + j*Vkl_IC(2));
phiKl_IC = atan2(Vkl_IC(2), Vkl_IC(1));

% % Polar Loop only_____________________

x_IC = inv(Ckl_d + Dkl_d*inv(Bkl_d)*(eye(size(Akl_d))-Akl_d))*Vkl_IC;
Vk_IC = inv(Dkl_d)*(Vkl_IC - Ckl_d*x_IC);

Ak_IC = abs(Vk_IC(1) + j*Vk_IC(2));
phiK_IC = atan2(Vk_IC(2), Vk_IC(1));
```

```
Klystron_limit = polyval([a(2) 0 a(1) 0], Sat_up_limit)*gain_factor;

if abs(Ak_IC) > Klystron_limit
    u = Sat_up_limit;
else
    Aux = roots([a(2) 0 a(1) -Ak_IC]);
    u = Aux(abs(Aux) <= Sat_up_limit);
end

phK = (pb(1)+pb(2)*u+pb(3)*u^2+pb(4)*u^3);

Adr_IC = u/ Gdr; phidr_IC = phiK_IC - phK;

Vdr_IN_IC = Adr_IC * cos(phidr_IC);
Vdr_Q_IC = Adr_IC * sin(phidr_IC);

prm.guideDelay.IC = ones(prm.guideDelay.N,1)*[Vdr_IN_IC Vdr_Q_IC];
prm.guideDelay.IC = prm.guideDelay.IC(:);

%_____________________________________________________
% Klystron Polar Loop + Modulator
%_____________________________________________________

Direct_phase = feedback.MODphase*pi/180;

Vkf_IC = Vkl_IC*Coupl_FWD;
prm.cordicDelayK.IC = ones(prm.cordicDelayK.N,1)*Vkf_IC';
prm.cordicDelayK.IC = prm.cordicDelayK.IC(:);

Avf_IC = Akl_IC*Coupl_FWD/polar_loop_gain_setting;
phivf_IC = phiKl_IC - polar_loop_phase_setting - Direct_phase;

Cordic_IC_F = Avf_IC * [cos(phivf_IC) sin(phivf_IC)];
prm.cordicDelayF.IC = ones(prm.cordicDelayF.N,1)*Cordic_IC_F;
prm.cordicDelayF.IC = prm.cordicDelayF.IC(:);

Att = Adr_IC/Avf_IC/OSC_gain;

Att_max = polyval(at,Vc_att_max);
Att_min = polyval(at,Vc_att_min);

if abs(Att) < Att_max
    A = Vc_att_max;
elseif abs(Att) > Att_min
```

```
    A = Vc_att_min;
else
    at0 = zeros(size(at)); at0(end) = Att;
    Aux = roots(at-at0);
    re = find(imag(Aux)==0); Aux = Aux(re);
    A = Aux(Aux <= Vc_att_max & Aux >= Vc_att_min);
end

ph_dist = polyval(ph,A);
phi = phidr_IC - phivf_IC - ph_dist;

prm.AInterpolatorDelay.IC = ones(prm.AInterpolatorDelay.N,1)*A;
prm.AInterpolatorDelay.IC = prm.AInterpolatorDelay.IC(:);

prm.phiInterpolatorDelay.IC = ones(prm.phiInterpolatorDelay.N,1)*phi;
prm.phiInterpolatorDelay.IC = prm.phiInterpolatorDelay.IC(:);

AIntegrator_IC = A;
phiIntegrator_IC = phi;

%_____________________________________________________
% RF feedback Loop (PI + Comb Filter)
%_____________________________________________________

Vref_IN_OL = Avf_IC * cos(phivf_IC);
Vref_Q_OL =  Avf_IC * sin(phivf_IC);

error = (Ki)\[Vref_IN_OL Vref_Q_OL]';
Vc_setpoint = Vcav_IC*FbAtt + error;

comb_in = prm.OTFdbk.gain*error;
comb_out = [cos(prm.OTFdbk.phase) -sin(prm.OTFdbk.phase); ...
    sin(prm.OTFdbk.phase) cos(prm.OTFdbk.phase)]*comb_in;

xint_IC = [Vref_IN_OL Vref_Q_OL]';
xana_IC = inv(eye(size(Aana))-Aana)*Bana*(error+comb_out);

prm.OTFdbk.IC = ones(prm.OTFdbk.N,1)*comb_in';
prm.OTFdbk.IC = prm.OTFdbk.IC(:);

prm.RFdelay.IC = ones(prm.RFdelay.N,1)*Vcav_IC';
prm.RFdelay.IC = prm.RFdelay.IC(:);

prm.fineDelay.IC = ones(prm.fineDelay.N,1)*comb_out';
prm.fineDelay.IC = prm.fineDelay.IC(:);
```

```
prm.coarseDelay.IC = ones(prm.coarseDelay.N,1)*comb_out';
prm.coarseDelay.IC = prm.coarseDelay.IC(:);
```

3.2 set_FB_prm.m

```
% set_FB_prm.m
% LHC time-domain simulation: File for Feedback parameter definition
%
% Copyright 2008 T. Mastorides, C. Rivetta

function feedback = set_FB_prm

feedback.meas = 5;
% Measurement type selection
switch feedback.meas
    case 1, feedback.pr = 8;
    case 2, feedback.pr = 1;
    case 3, feedback.pr = 6;
    case 5, feedback.pr = 3;
    case 6, feedback.pr = 1;
    case 8, feedback.pr = 7;
    case 9, feedback.pr = 8;
end
if feedback.meas == 3
    feedback.digital.switch = 0;
else
    feedback.digital.switch = 1;
end
switch feedback.meas
    case 3
        feedback.noiseGain = 1e-3;
    case 9
        feedback.noiseGain = 1e-5;
    otherwise
        feedback.noiseGain = 1e-7;
end
switch feedback.meas
    case {5,9}
    feedback.OL = 0;
    otherwise
    feedback.OL = 1;  % 1 for OL on
end
switch feedback.meas
    case {8,9}
    feedback.OTFdbk.switch = 1; % 1 for comb on
    otherwise
    feedback.OTFdbk.switch = 0; % 1 for comb on
end
```

```
% A Optimal settings for Io = 0, wd = 0, Vc = 1 MV, Ql = 2e4
gain = -17.44; phase = 2.4; combDelay = 8.78e-5;
current = 0; cavity = 1e6; detuning = 0e3; Q = 2e4;

feedback.OTFdbk.by_pass = abs(1-feedback.OTFdbk.switch);%0 for comb ON
feedback.Ql = Q;
feedback.fdk.att = gain;
feedback.MODphase = phase; % Modulator phase in degrees
feedback.OTFdbk.phase = 0.05; % in degrees
feedback.OTFdbk.delay = combDelay;
feedback.OTFdbk.gain = .65;
Io = current;
feedback.IB = 2*Io;
feedback.Vc = [cavity 0]';
feedback.wd = -2*pi*detuning;
```

3.3 inte_cic.m

```
% inte_cic.m
% LHC time-domain simulation:
% INTE_CIC interpolator implemented for injecting noise at LHC RF stations
% SISO model
%
% x input signal (length = 2^12 = 4096)
% rate: rate of interpolation (Ts = Tr/2^rate)
% Ts : Output sampling period
% y: output signal (length = 2^12 * 2^rate)
%
% Copyright 2008 T. Mastorides, C. Rivetta

function y = inte_cic(x,rate, Ts)

if rate == 0

    y = double(x');

else

    N = length(x);

    da = zeros(1,N*2^rate);
    da(1:2^rate:end) = double(x);

    numI = ones(1,2^(rate+1))/(2^(rate));
    denI = zeros(1,2^(rate+1)+1);
    denI(1) = 1;
    INTE = tf(numI,denI,Ts);

    INTE2 = INTE*INTE*(2^(rate-2));

    tfil = (0:1:length(da)-1)*Ts;

    y = lsim(INTE2, da,tfil);

end
```

3.4 load_klystron_C.m

```
% load_klystron_C.m
% LHC time-domain simulation:
% Function load_klystron_C calculates the coefficient for a fitting
% curve with the means of linear regression and plots original
% points and the curve. The function call includes parameters
% gain_factor, phase_factor, plot_figure and figure_number
% This function includes the correction due to the transfer
% function of the klystron (freq. response)
%
% Copyright 2006 J. Holma
%           2008 T. Mastorides, C. Rivetta

function [a, pb] = load_klystron_C(gain_factor, ...
    phase_factor, figure_number)

% plot settings
plot_limit_lower = -100;
plot_limit_upper = 118;
step_size = 0.1;

% number of added values to the measured data to make phase
% shift constant in small input voltages
add_values = 100;

% open file data7.mat (klystron data)
data = open('data7.mat');
data7 = data.data7;

% reads data
P_in = data7(1:end,1);
P_out = data7(1:end,2);
phase = data7(1:end,3);

% Parameters from curves
P_out_max_W = 300e3;
P_out_max_dB = -14.856;
P_dB_300kW = 10*log10(300e3/P_out_max_W)+P_out_max_dB;

%%
P_in_dBm_300kW = -1.00;     % from plot
G = 37.0;
P_G = 300e3;
```

```
% calculates P_out and P_in in Volts
P_out_W = P_out_max_W*10.^((P_out-P_out_max_dB)/10);
U_out_V = sqrt(50.*P_out_W);
P_in_W =P_G*10^-(G/10)*10.^((P_in-P_in_dBm_300kW)/10);
U_in_V = sqrt(50.*P_in_W);

%% edit klystron7 data
% Set the correction due to the klystron transfer function

phase = phase - phase_factor;              % shifts phase response!
U_out_V = U_out_V / gain_factor;           % changes gain response!


% mirrors the input/data to negative input/output values
U_in_V_double = (zeros(2*length(U_in_V),1));
U_in_V_double((length(U_in_V)+1):end)=U_in_V(1:end);

U_out_V_double = (zeros(2*length(U_out_V),1));
U_out_V_double((length(U_out_V)+1):end)=U_out_V(1:end);

phase_double = (zeros(2*length(phase),1));
phase_double((length(phase)+1):end) =phase(1:end);

for i=1:(length(phase))
    U_in_V_double(i) = -1*U_in_V((length(U_in_V)+1-i));
    U_out_V_double(i) = -1*U_out_V((length(U_out_V)+1-i));
    phase_double(i) = -1*phase((length(phase)+1-i));
end;

%%
% calculates coefficients of Uout = a1*Uin+a3*Uin^3
X = [U_in_V_double U_in_V_double.^3];
a = X\U_out_V_double;

Uin_fit = (plot_limit_lower:step_size:plot_limit_upper)';
Uout_fit = [Uin_fit Uin_fit.^3]*a;

% calculates coefficients of Uout = b1*Uin+b3*Uin^3+b5*Uin^5
X2 = [U_in_V U_in_V.^3 U_in_V.^5];
b = X2\U_out_V;

Uout_fit2 = [Uin_fit Uin_fit.^3 Uin_fit.^5]*b;


%% plots fitted curves and measured values
```

```
if (figure_number ~= 0)

    figure(figure_number);
    subplot(2,1,1)
    plot(Uin_fit,gain_factor*Uout_fit,'-',U_in_V_double,...
        gain_factor*U_out_V_double,'x');
    title('Uout = a1*Uin+a2*Uin^3')
    xlabel('Uin [V_{rms}]')
    ylabel('Uout [V_{rms}]')
    grid on;
    zoom on;
end

%%
% adds points with the value of the first measured value to phase,
% makes phase constant in small voltages
phase_added = (zeros(length(phase)+add_values,1));
phase_added(1:(length(phase_added)-length(phase))) = phase(1);
phase_added(length(phase_added)-length(phase)+1:end)=phase;

% corrects the lenght of U_in_V => U_in_V_added
P_in_added = (zeros(length(P_in)+add_values,1));
for i=1:add_values
    P_in_added(add_values+1-i)=P_in(1)-i*(P_in(2)-P_in(1));
end

P_in_added(add_values+1:end)=P_in;

 % uses parameters P_G, G and P_in_dBm_300kW from read_klystron7
 P_in_W_added =P_G*10^-(G/10)*10.^((P_in_added-P_in_dBm_300kW)/10);
 U_in_V_added = sqrt(50.*P_in_W_added);

 Uin_fit2 = (plot_limit_lower:step_size:plot_limit_upper)';

 % calculates coefficients of phase = pa1+pa2*Uin+pa3*Uin^2

 Y = [ones(size(U_in_V_added)) U_in_V_added U_in_V_added.^2];
 pa = Y\phase_added;
 phase_fit1 = [ones(size(Uin_fit2)) Uin_fit2 Uin_fit2.^2]*pa;

 % calculates dots of the phase curve without first term
 % (begins from zero)
 pa_zero = pa;
 pa_zero(1) = 0;
```

```
 phase_fit2 = [ones(size(Uin_fit2)) Uin_fit2 Uin_fit2.^2]*pa_zero;

 % calculates coefficients of phase=pb1+pb2*Uin+pb3*Uin^2+pb4*Uin^3

 Y2 = [ones(size(U_in_V_added)) U_in_V_added ...
     U_in_V_added.^2 U_in_V_added.^3];
 pb = Y2\phase_added;
 phase_fit3 = [ones(size(Uin_fit2)) Uin_fit2 ...
     Uin_fit2.^2 Uin_fit2.^3]*pb;

 % calculates dots of the phase curve without first term
 % (begins from zero)
 pb_zero = pb;
 pb_zero(1) = 0;
 phase_fit4 = [ones(size(Uin_fit2)) Uin_fit2 ...
     Uin_fit2.^2 Uin_fit2.^3]*pb_zero;

 % calculates coefficients of phase = pc1+pc2*Uin+pc3*Uin^3

 Y3 = [ones(size(U_in_V_added)) U_in_V_added U_in_V_added.^3];
 pc = Y3\phase_added;
 phase_fit5 = [ones(size(Uin_fit2)) Uin_fit2 Uin_fit2.^3]*pc;

 % calculates dots of the phase curve without first term
 % (begins from zero)
 pc_zero = pc;
 pc_zero(1) = 0;
 phase_fit6 = [ones(size(Uin_fit2)) Uin_fit2 Uin_fit2.^3]*pc_zero;


%%
if (figure_number ~= 0)
    % plots phase curves

    subplot(2,1,2);
    plot(Uin_fit2,phase_fit3 + phase_factor,'-',U_in_V_added,...
        phase_added + phase_factor,'x');
    hold on;
    plot(Uin_fit2, phase_fit4,'RED');
    title('\theta = b0+b1*Uin+b2*Uin^2+b3*Uin^3')
    xlabel('Uin [V_{rms}]')
    ylabel('phase shift \theta [rad.]')
    grid on;
    zoom on;
end
```

References

[1] C. Rivetta, T. Mastorides, J. D. Fox, D. Teytelman, and D. Van Winkle, "Modeling and simulation of longitudinal dynamics for Low Energy Ring - High Energy Ring at the Positron-Electron Project," *Phys. Rev. ST Accel. Beams*, vol. 10, p. 022801, 2007.

[2] T. Mastorides, C. Rivetta, J. D. Fox, D. Van Winkle, and D. Teytelman, "Analysis of Longitudinal Beam Dynamics Behavior and RF System Operative Limits at High Beam Currents in Storage Rings," *Phys. Rev. ST Accel. Beams*, vol. 11, p. 062802, 2008.

[3] T. Mastorides, C. Rivetta, J. D. Fox, D. Van Winkle, and D. Teytelman, "RF system models for the LHC with Application to Longitudinal Dynamics," Submitted for Publication to Phys. Rev. ST Accel. Beams.

[4] T. Mastorides, C. Rivetta, J. D. Fox, D. Van Winkle, and D. Teytelman, "RF Noise Effects on Large Hadron Collider Beam Diffusion," Submitted for Publication to Phys. Rev. ST Accel. Beams.

[5] T. Mastorides *et al.*, "Modeling and Simulation of the Longitudinal Beam Dynamics - RF Station Interaction in the LHC Rings," Contributed to 11th European Particle Accelerator Conference (EPAC 08), Magazzini del Cotone, Genoa, Italy, 23-27 Jun 2008.

[6] T. Mastorides, C. Rivetta, J. D. Fox, and D. Van Winkle, "Application of Non-Linear Time-Domain RF Simulations to Longitudinal Emittance Studies for the LHC," Contributed to Particle Accelerator Conference (PAC 09), Vancouver, BC, Canada, 4-8 May 2009.

[7] T. Mastorides, C. Rivetta, J. D. Fox, D. Van Winkle, and D. Teytelman, "LHC Beam Diffusion Dependence on RF Noise: Models and Measurements," Presented at International Particle Accelerator Conference (IPAC 10), Kyoto, Japan, 23-28 May 2010.

[8] P. Baudrenghien *et al.*, "The LHC Low Level RF," Proceedings of European Particle Accelerator Conference (EPAC 06), Edinburgh, Scotland, 26-30 Jun 2006.

[9] J. Holma, "The Model and Simulations of the LHC 400 MHz Cavity Controller," CERN-AB-2007-012, CERN, Feb. 2007.

[10] D. Teytelman, "A non-invasive technique for configuring low level RF feedback loops in PEP-II," Presented at Particle Accelerator Conference (PAC 05), Knoxville, Tennessee, 16-20 May 2005.