Institute for Research in Fundamental Sciences

Institute for studies in Theoretical Physics and Mathematics (IPM)

School of Particles and Accelerators

Ph.D. Thesis

Accelerator Physics

Longitudinal Beam Dynamics Studies at CTF3 And Pulse Compressor Controlling

By: Seyd Hamed Shaker

Supervisor: Roberto Corsini

2009, March

Acknowledgement

I would like to thank Dr. Roberto Corsini who supervised this project and being always ready to help.

I would like to thank IPM, especially Dr. Arfaei who supported me to do this project.

I am grateful to my wife, *Somayeh*, for every little or big thing which I'll never forget.

I'd like to thank E. Adli, H. H. Braun, A. Dabrowski, A. Latina, T. Lefevre, P. K. Skowronski, I. Syratchev, F. Tecker, P. Urschutz and all my friends during my studies at CERN which excluding the help of any of them, this thesis wouldn't have been possible to finish.

Abstract

The aim of the CLIC Test Facility CTF3, built at CERN by an international collaboration, is to address the main feasibility issues of the CLIC electron-positron linear collider technology by 2010. One key-issue studied at CTF3 is the generation of the very high current drive beam, used in CLIC as the RF power source. It is particularly important to simulate and control the drive beam longitudinal dynamics in the drive beam generation complex, since it directly affects the efficiency and stability of the RF power production process. In this thesis how to use pulse compressors to achieve a high RF power in the drive beam accelerator is discussed. We also describe the ongoing effort in modelling the longitudinal evolution of the CTF3 drive beam and compare the simulations with experimental results. Our study is based on the single bunch simulation.

Keywords: Pulse compressor, longitudinal beam dynamic, CTF3, CLIC

TABLE OF CONTENTS

List of Tables

List of Figures

1. AN OVERVIEW TO CLIC

To know more about CLIC (Compact LInear Collider) we should start from the LEP (Large Electron Positron collider). The LEP [1] was a circular accelerator where electrons and positrons were accelerated in opposite direction and then brought into collision. LEP had a limited energy of 100 GeV in the centre-of-mass. This limitation was coming from the energy loss induced by synchrotron radiation. This energy loss is proportional to fourth power of particle energy and RF power should compensate it and for the energy more than 100 GeV it becomes very difficult and expensive.

LEP was located in a 27km long tunnel, about 100 meter underground. It was stopped in 2000 and it had been replaced by LHC, built in the same tunnel.

LHC (Large Hadron Collider) accelerates two proton beams, in opposite direction. It uses the superconductive technology. The nominal centre-of-mass energy of LHC is 14 TeV. However, the results of LHC interactions are very complicated because of the structure of the proton. Therefore the LHC is considered to be mainly a discovery machine. For example, there are strong expectations that it will discover the Higgs boson. In order to

perform more precise studies we need another machine. CLIC has the potential to be that machine.

CLIC (see Fig. 1-1) is a linear accelerator and accelerate electrons and positrons in opposite directions. The synchrotron radiation is negligible in linear acceleration, then it doesn't impose any limitation.

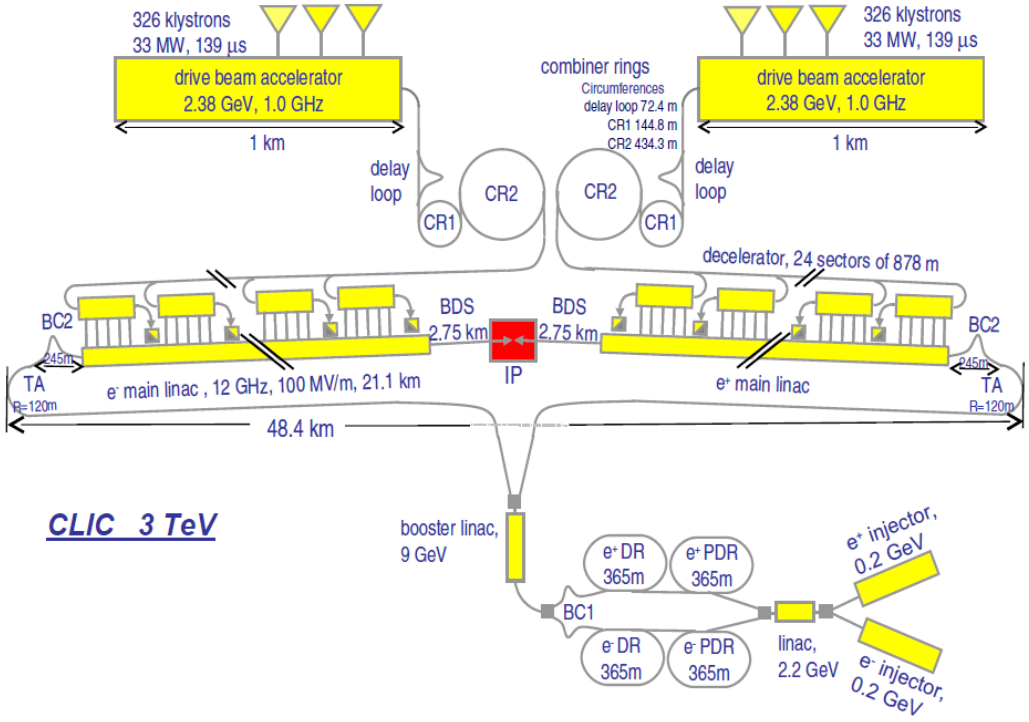

Figure 1-1: CLIC layout [2].See also [3] for the older version.

The aim of the CLIC Study is to investigate the feasibility of a linear e+ e- collider with a centre-of-mass energy reach of $E_{CMS} = 3\,\text{TeV}$ and high luminosity ($5.9 \times 10^{34}\ \text{cm}^{-2}\text{s}^{-1}$) [2]. In order to minimize the total length, CLIC employs normal-conducting accelerating structures operating at a very high gradient (100 MeV/m). In this case the final length of CLIC should be around 50 km. High peak RF power is required in CLIC to feed the accelerating structures and is obtained using a two-beam acceleration concept [4], in which a high current electron beam (drive beam) runs

parallel to the main beam and is decelerated to produce the RF power. Since the drive beam is separately generated in a central area, a single tunnel with a limited diameter (≈ 4.5 m) and without active high-power components can be used to house the two parallel beams.

The generation of the high-intensity drive beam pulses with the right time structure is indeed one of the main challenges of CLIC. In the adopted scheme, a long pulse is accelerated using a low frequency normal-conducting linac. Funnelling techniques in delay lines and rings are then used to give the beam the desired structure while increasing its intensity. In this process the electron bunches are interleaved by the use of transverse RF deflectors. The bunch spacing is thus reduced and the beam current is increased.

It is generally accepted that CLIC technology is the only possible path to multi-TeV colliders. However, several critical issues still need to be addressed.

2. AN OVERVIEW TO CTF3

The experimental program of the present CLIC Test Facility, CTF3 [5], tackles most of the main issues of the study, related to the generation and use of the drive beam and the testing of accelerating structures and RF components.

CTF3 was constructed at CERN by an international collaboration and it will be finished around 2010 and from the result of the CTF3 will be decided whether to build the CLIC or not.

The CTF3 complex is designed to work at a lower beam current and a lower energy than the CLIC RF power source (3.7 A instead of 5.5 A and 120 MeV compared to 2.4 GeV).

CTF3 (see Fig. 2-1) is composed of a 70 m long drive beam linac followed by two rings, a 42 m long delay loop and an 84 m combiner ring. The linac delivers a 1.5 μs long electron beam. The beam, bunched at the frequency of 1.5 GHz , is injected in the rings where the beam average current and the bunch frequency are multiplied by a factor 8. After the combination process, the drive beam has a current of 30 A and can be transported in an experimental area to produce RF power at 12 GHz (corresponding to the

final bunch repetition frequency) in a high power test stand. In the same area, a separate linac will provide a main beam to a representative CLIC two-beam module and a test decelerator will be dedicated to drive beam stability studies.

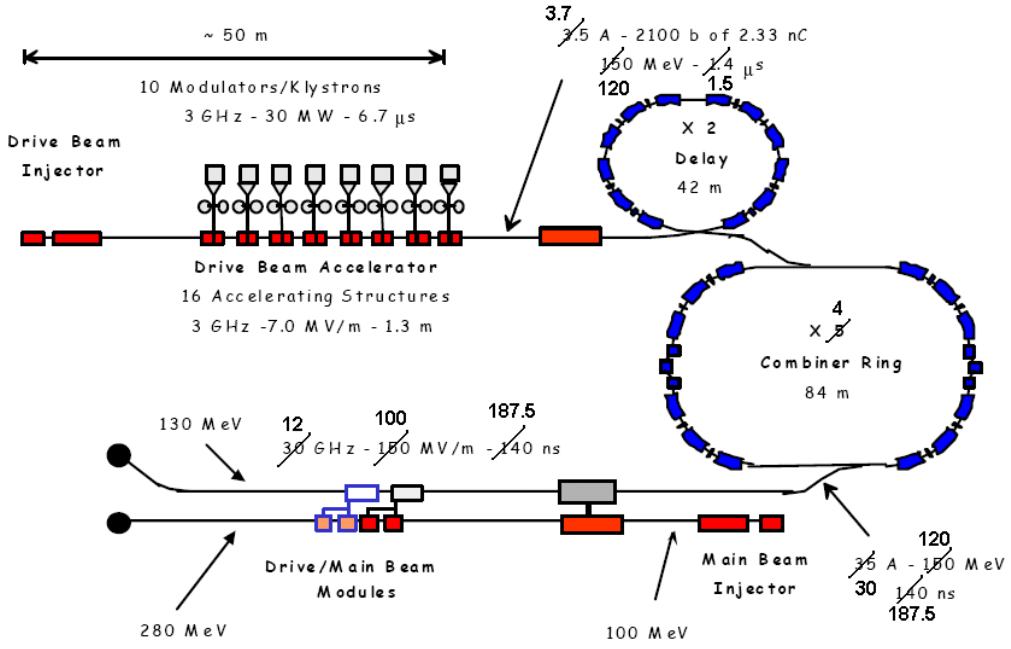

Figure 2-1: Conceptual layout of CTF3 with the new parameters [5]

3. PULSE COMPRESSOR SIMULATION AND PHASE MODULATOR CONTROLLING

3-1. Introduction

The current layout of the injector and the drive beam accelerator including the klystrons (MKx) and the pulse compressors (LIPS and BOC) is shown in Fig. 3-1. These klystrons provide the necessary power in the form of electromagnetic waves for the cavities in the injector and drive beam accelerator sections.

Shape of the pulse out of the klystrons is roughly rectangular. The output power of klystrons at the CTF3 is around 30 MW and the pulse length is around 6000 ns. It is difficult to build a klystron with higher output power. The pulse compressor is an alternative method to achieve higher power.

Pulse compressors store the RF energy of the klystrons and release it to the acceleration sections in a very short time before the end of the RF pulse of klystron. They make the pulse shorter but more powerful and can double the power output.

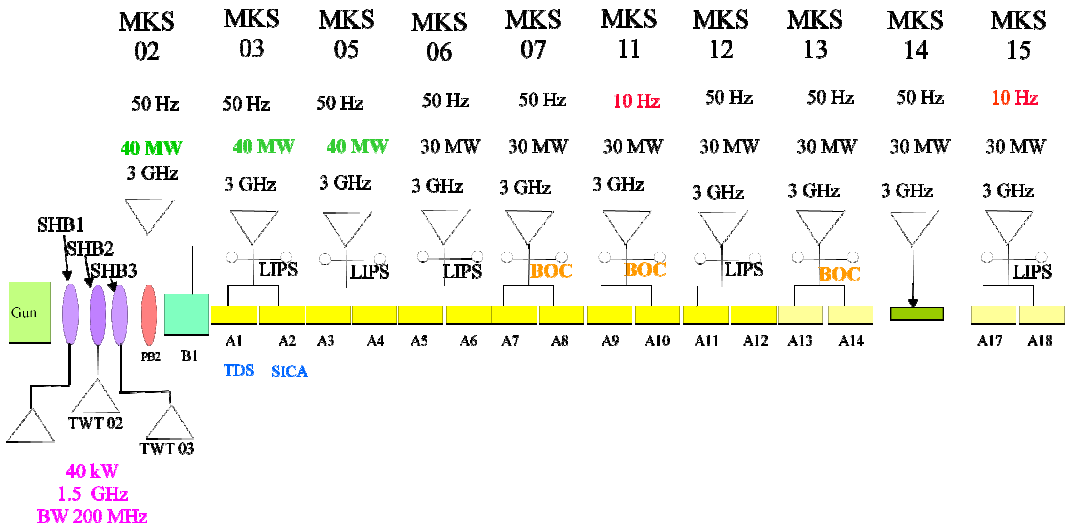

Figure 3-1: Klystrons layout of CTF3 2007. [6]

There are two kinds of pulse compressors at the CTF3: LIPS and BOC.

The LIPS contains two cylindrical cavities while the BOC (Barrel Open Cavity) contains one spherical cavity.

It is not our goal to go inside the structure of different pulse compressors but a little introduction about the LIPS will be given. For more details see [7 and 8].

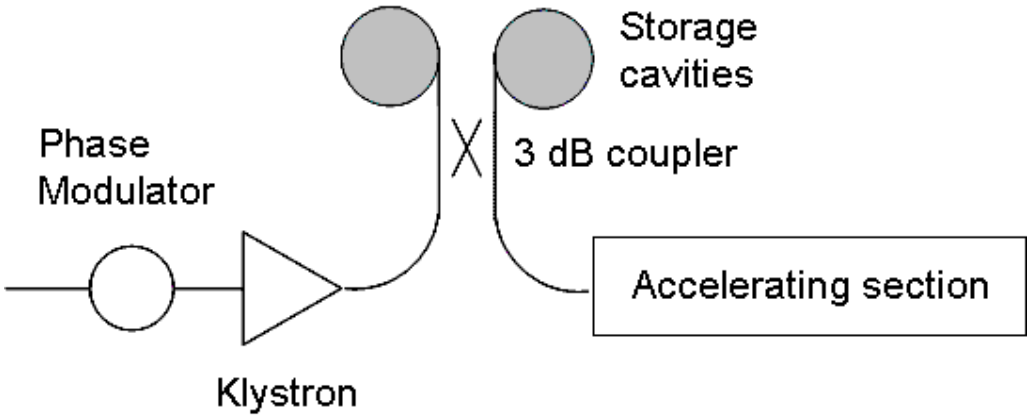

Figure 3-2: Layout of LIPS and phase modulator. [7]

The LIPS contains two storage cavities, see Figs. 3-2 and 3-3. These are cylindrical cavities resonating at the same frequency. Mechanical tuners in these cavities can synchronize them by a precision of a few KHz.

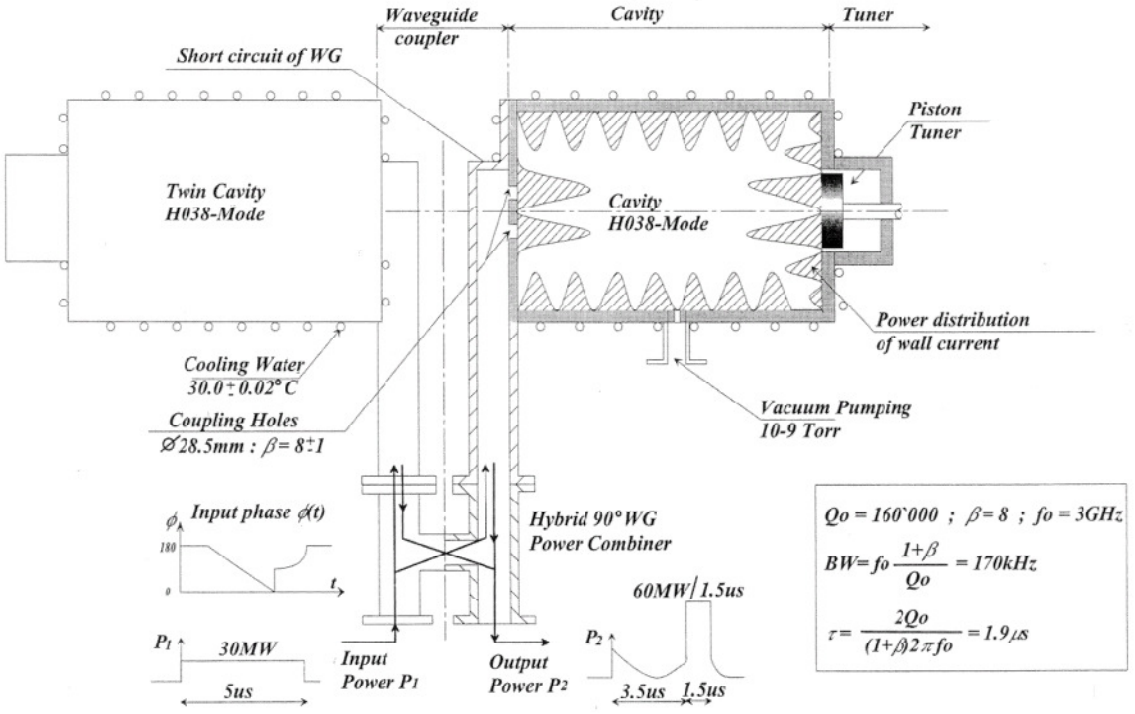

Figure 3-3: LIPS structure [8]

A 3db hybrid coupler prevents reflecting of the wave back to the klystrons.

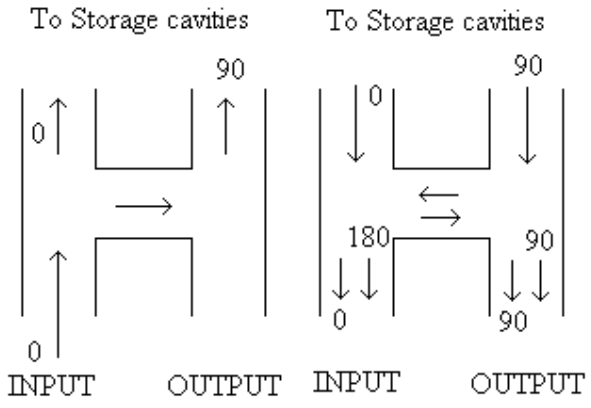

Figure 3-4: The mechanism of 3db hybrid coupler

As shown in Fig. 3-4 the input wave is divided into two equal waves inside the 3db hybrid coupler. The right wave travels a longer trajectory, equal to one quarter of the wavelength, and makes a 90 degree phase difference with the left wave. The reflected waves from the cavities are divided into two equal waves similarly and in the input of the 3db hybrid the two waves have the same magnitude and opposite direction which makes them to cancel each other, while in the output they make the same magnitude and the same direction. Then we will not have any waves reflected to the klystrons.

3-2. Phase modulator

Fig. 3-5 shows the output wave of pulse compressor when the output pulse of klystrons are rectangular with a constant phase. But we need a flat top at the end of output pulse.

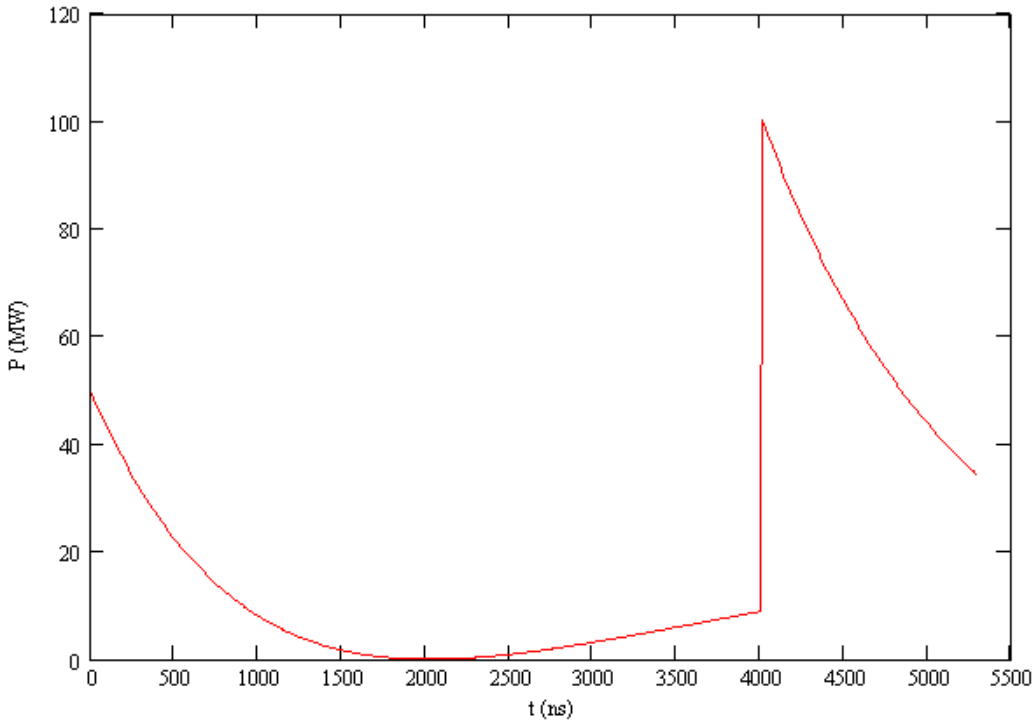

Figure 3-5: Pulse compressor output without the phase modulator.

Fiebig and Schieblich [7] proposed a phase modulator to be installed before the klystrons (see Fig. 3-2). They can manipulate the phase of output pulse of the klystron. By controlling the phase of wave by time we can have a flat top at the end output pulse. Fig. 3-6 shows the shape of input phase for the phase modulator.

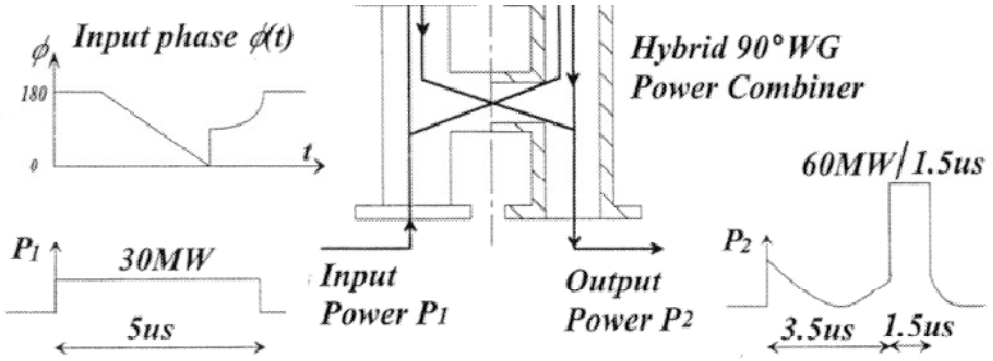

Figure 3-6: Shape of input phase for the phase modulator and the input and the output of the pulse compressor (Look Fig.3-3).

By choosing the correct phase shape for the phase modulator - as shown in Fig. 3-6 - we can have a flat top pulse output for the short time in the end of output pulse of the pulse compressor.

The shape and how to control it to have an output with the required flat top will be addressed in section 3-4.

3-3. Simulation of the pulse compressors

Fiebig and Schieblich [7] proposed the equivalent circuit (see Fig. 3-7) and an equation (see Eqn. 3-1) for the relation among the phase modulator, klystron and the pulse compressor.

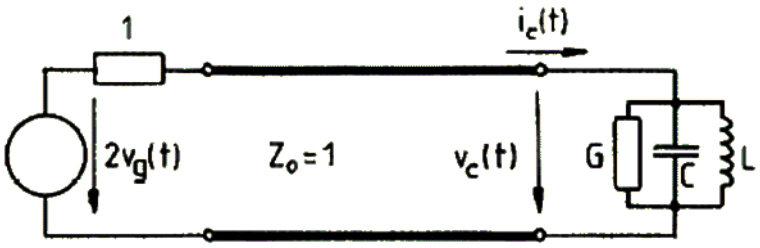

Figure 3-7: The equivalent circuit of phase modulator klystron and pulse compressor [7]

$$\alpha V_g = V_c + \tau \dot{V}_c, \qquad (3\text{-}1)$$

where V_g is the voltage of input wave and V_c is the voltage that is produced by pulse compressor cavities which both are complex variables, the output wave is V_r that is equal to $V_c - V_g$, $\alpha = 2\beta/(1+\beta)$, $\tau = 2Q_0/[(1+\beta)\omega_0]$, $\beta \approx 7$ is the coupling coefficient of the pulse compressor cavities, $Q_0 \approx 1.8\times10^5$ is the quality factor of pulse compressor cavities, and $\omega_0 = 2\pi \times 2.99855\,\text{GHz}$ is the angular frequency of electromagnetic waves which results to $\alpha \approx 1.7$ and $\tau \approx 2\,\mu\text{s}$.

We are mainly interested in the power than the complex voltage. The relation between complex voltage and power is $P_{g,c,r} = C \| V_{g,c,r} \|^2$. In this equation the constant C could be removed from both sides of the Eqn. 3-1 then we can replace $V_{g,c,r}$ by $\sqrt{P_{g,c,r}}e^{i\phi_{g,c,r}}$, where $\phi_{g,c,r}$ is the phase of electromagnetic waves. ϕ_g is the phase of input wave of pulse compressor that is determined by the phase modulator and in the absence of phase modulator is constant. Figs. 3-6 and 3-9 show the shape of this phase.

ϕ_r is the phase of output wave of pulse compressor which is constant when the phase modulator is absent but with the phase modulator it will change by time. To minimize these changes we should change $\Delta\omega$ that will be described below also in the sections 3-4 and 3-5.

Eqn. 3-1 is valid if the two cavities of LIPS or two orthogonal output waves of BOC are matched and reflected power back to the klystrons is zero and also if the resonant frequency of the pulse compressor cavities is equal to the wave frequency. In case the resonant frequency is not equal to the wave frequency - that is manually done to remove the first order (linear part) of the embowed shape phase output (ϕ_r) - the Eqn. 3-1 should be replaced by this equation:

$$\alpha V_g = V_c(1 + i\tau\Delta\omega) + \tau\dot{V}_c, \qquad (3\text{-}2)$$

where $\Delta\omega = \omega_0 - \omega_c$ and ω_c is the angular resonant frequency of the pulse compressor cavities.

This equation can be implemented for the simulation program (appendix I) in this numerical form:

$$\alpha V_{g,n} = V_{c,n}(1 + i\tau\Delta\omega) + \tau\left(\frac{V_{c,n+1} - V_{c,n-1}}{2\Delta t} + O(\Delta t^2)\right)$$

or (3-3)

$$V_{c,n+1} = V_{c,n-1} + \frac{2\Delta t}{\tau}[\alpha V_{g,n} - V_{c,n}(1 + i\tau\Delta\omega)] + O(\Delta t^3)$$

There are two reasons for the simulation of the pulse compressors:

1- To check the theory (Eqns. 3-1 and 3-2)

2- To measure the parameters of the pulse compressor cavities.

Using the results of the simulations $Q_0/(1+\beta)$, $\Delta\omega$ and β (with lower precision) can be calculated. In Eqn. 3-1 and 3-2 there are α and τ which $\alpha = 2\beta/(1+\beta)$ showing that α is not very sensitive to β, *e. g.* if β changes from 6 to 9, α changes from 1.7 to 1.8. $\tau = 2Q_0/[(1+\beta)\omega_0]$ and ω_0 is known then we can find $Q_0/(1+\beta)$. τ is a time scale which can be found by comparing the measurement and simulation results. In section 3-4 I will show how to measure $\Delta\omega$.

The simulation results show a good agreement with the measurement (see Fig. 3-8).

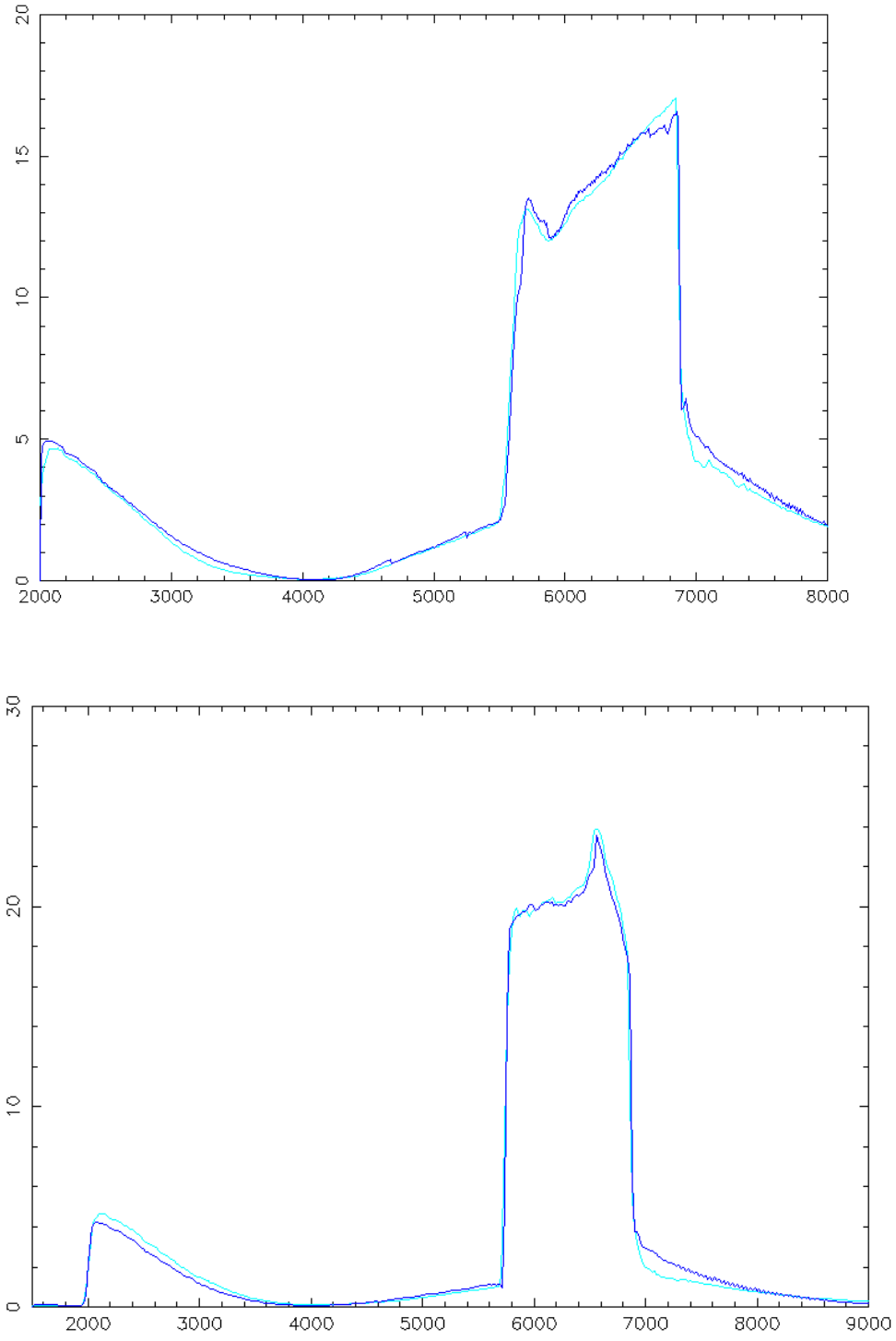

Figure 3-8: Result of simulation in comparison with the measurement result. (dark blue: measurement, light blue: simulation)

Fig. 3-8 shows a very good agreement but still there is some difference. There are some reasons for this:

1- Non-linearity of phase modulator: The output of phase modulator is not exactly what has been demanded, but there is a known non-linear relation between them that we could use it before sending data to the phase modulator

2- Non-equality of klystron phase and phase modulator

2.1- Slow response effect: if output phase of phase modulator changes fast then klystron cannot follow it rapidly then we will see some difference

2.2- Noises

3- Difference of real parameters from the used parameters in the simulations

4- Partial mismatch of the two cavities of LIPS.

5- The approximation that is used in the main equation: it's not considerable but in the Eqn. 3-1 and 3-2 there are some approximations.

3-4. Phase modulation

The computer program used for the simulations is also able to control the phase modulator (appendix I).

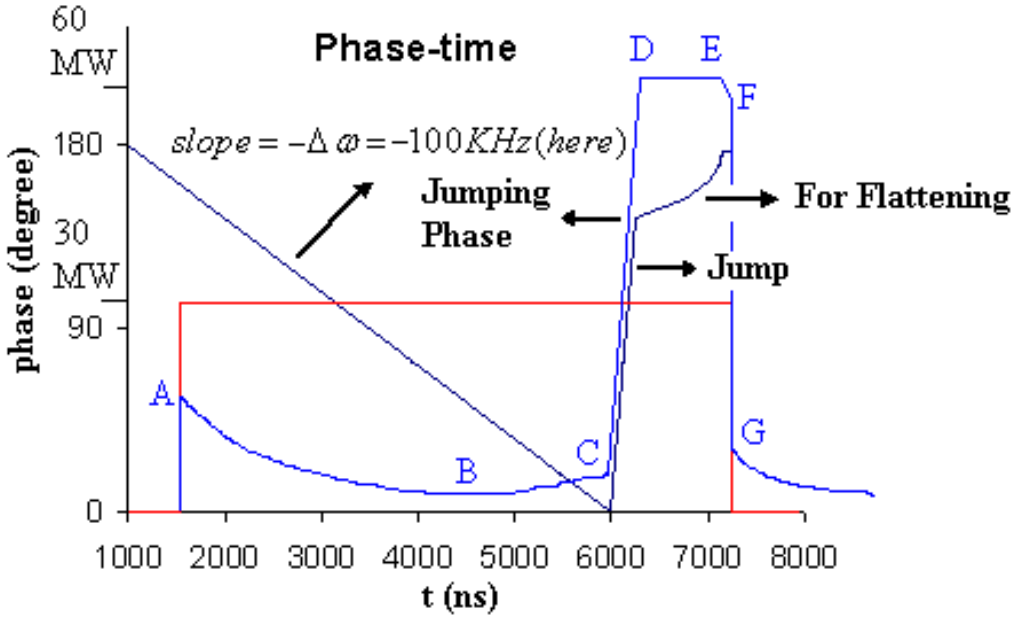

Figure 3-9: Phase input for the phase modulator and the input and output pulse of pulse compressor.

Fig. 3-9 shows the general phase shape of the phase modulator. The input of pulse compressor is supposed to be rectangular (red line). The output pulse (light blue line) is started from the same time of input pulse (point A). We will have the minimum magnitude in the point B. The first part of phase is a line with the slope of $-\Delta\omega$ which in this case we have the minimum power at point B that gives us a method for finding the $-\Delta\omega$. We change the slope and look for the power value at point B. When the minimum amount is reached then the slope of line is equal to $-\Delta\omega$. This line crosses the zero at point C. $\Delta\omega$ is zero by default which we should change it to achieve a minimum phase change for the output pulse of the pulse compressor (ϕ_r) (see section 3-5). $\Delta\omega$ can be changed using the tuners of the cylindrical cavities in the LIPS. For the BOC it is more difficult but we can change $\Delta\omega$ by changing the temperature.

At the point C the phase jumps from 0 to the jumping phase (point D) and the output power changes to final power (point D). The magnitude of

phase jump is important because it determines the final output power and improves the optimum flattening.

From point D the iterating part of the program will start. This iteration finds the next phase to keep constant the output pulse power.

In the point E, the phase reaches 180°. It means we cannot keep the output pulse power constant further more since it will start to decrease exponentially from point E to point F.

At the point F the input pulse is finished and the output pulse will jump to the point G and will start to decrease exponentially.

Fig. 3-9 shows us that we can multiply the input power by a factor of 2 from 30 MW to 60 MW.

Fig. 3-11 shows two real examples of the output pulse of a pulse compressor that has been tried to reach the required flat top at the end of pulse using this method. This figure also indicates that there are still some ripples which should be removed. I used two methods to remove the ripples:

First method consists of:

- Sending constant phase to the phase modulator.
- Reading the klystron output phase.
- Re-sending the negative of the read out of klystron to the phase modulator.

Second method consists of:

- Sending the real phase to the phase modulator.
- Reading the klystron output phase: In this case the phase changes appreciably and the non-linearity of phase reader devices becomes important that needs a new code (appendix II) which reads the klystron output phase part by part with changing the phase base of phase reader and combines the results to achieve a linear result.

- Re-sending the value of the real phase minus difference between the input and output phase to the phase modulator.

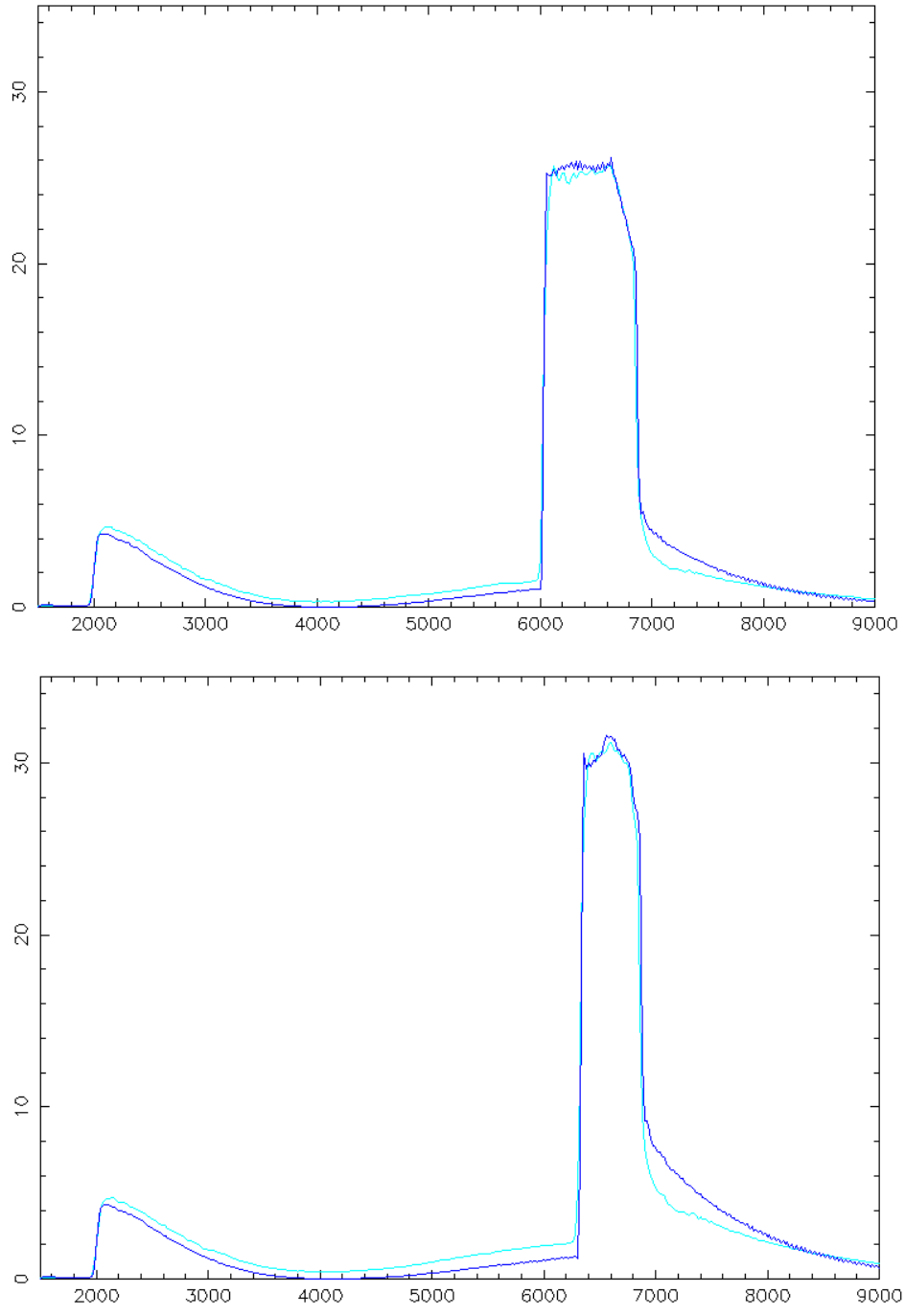

Figure 3-10: The output pulse of the pulse compressor that the phase modulator is used for the flattening.

These programs make a nice flat output pulse. To achieve a very good output pulse we used a program that is used in the control room of CTF3. This program is a feed-back program that looks at the output pulse of the pulse compressor and changes the phase of the phase modulator till it is almost flat.

3-5. Removing the first order (linear part) of the embowed shape phase output

Fig. 3-10 shows the phase of input and output pulse of pulse compressor. The Input phase is set by the phase modulator to achieve a flat top output at the end of the output pulse but the phase of output pulse is changing by $\Delta\varphi \approx 60^0$. There is one way to decrease the change by removing the first-order or linear part of phase. By doing so the phase change becomes acceptable (see Fig. 3-10). To remove the linear bias it is enough to change $\Delta\omega$ equal to the slope of the linear part of phase output and then set the slope of the first part of the phase of phase modulator equal to $-\Delta\omega$ as described before (see Fig. 3-9). For more details see [6].

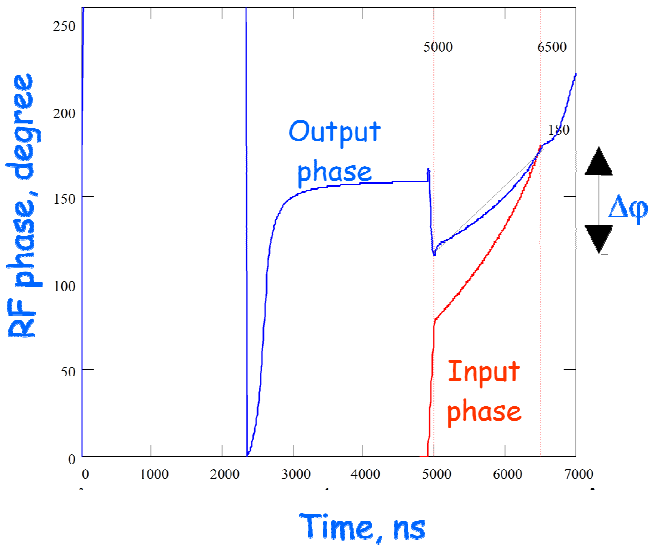

Figure 3-11: The phase shape of the input and output pulse of pulse compressor [9]

3-6. Technical notes about the computer programs

These computer programs (Appendix I & II) are written in C++ and are able to communicate with the devices of the CTF3 to read or write the data. "The CERN-PS Equipment Access Library" [10] provides standard routines for these communications.

Table 3-1 shows the communicating devices in the CTF3 that is used to read data.

Table 3-1: Devices in the CTF3 to read data

CK.SVPKIxxA-C	Input pulse power of the pulse compressor
CK.SVPKIxxP-C	Input pulse phase of the pulse compressor
CK.SVPSIxxA-C	Output pulse power of the pulse compressor
CK.SVPSIxxP-C	Output pulse phase of the pulse compressor
CK.SVWFGMKSxx-C	Phase of the phase modulator

"xx" is changed from "03" to "15" for the klystrons and pulse compressors for the girders of 3-15 in the injector and drive beam accelerator.

To read the data from these devices I used EqpCreateByName function using the properties in table 3-2 by EqpWrite and EqpRead functions:

Table 3-2: Properties for reading data

EQP_INTERV	To set the interval time for reading data
EQP_DELAY	To read the starting time of data
EQP_STRT	To set the starting time to read data
EQP_HIGH	To read the number of samples
EQP_AQN	To read data

For EQP_AQN property EQP_TYPE_FLOAT type should be set by EqpSetProperty function.

I used also CK.WFGMKSxx devices for setting the phase of the phase modulator by EQP_CCV property and EQP_TYPE_FLOAT type by EqpWrite and EqpSetProperty functions.

At the moment some of these devices are replaced by the XenericSampler class. For the future work these devices should be replaced by this class and the new method to access the devices can be used too [11 and 12].

4. LONGITUDINAL BEAM DYNAMICS STUDIES

The performance of the accelerator depends directly on the control of the electron bunch length. In the linac the bunches must remain short (about $1\,\mathrm{mm}$ r.m.s.) to keep the energy spread as low as possible, but need to be stretched $2-2.5\,\mathrm{mm}$ r.m.s. before the rings to minimize coherent synchrotron radiation (CSR) emission, which can cause energy loss and increase the bunch energy spread and transverse emittance. The bunches can finally be recompressed to about $1\,\mathrm{mm}$ r.m.s. in the transfer line following the combiner ring [13], specially designed to have a tunable momentum compaction R_{56}. The bunch length in the linac and in the rings can be controlled using two magnetic chicanes, the first downstream of the injector and the second upstream of the first ring (see Fig. 4-1). Normally, bunch shortening or lengthening is obtained by changing the phase of the RF in the first or the last accelerating structure, respectively. Bunch length measurements can be performed using Optical Transition Radiation (OTR) screens coupled to a Streak camera [14], but the present system limits the time resolution to $2\,\mathrm{ps}$ or more. Shorter bunches are measured with RF deflectors, also used in conjunction with OTR screens. In particular, we will present in the following, results of the measurements obtained using

the 1.5 GHz RF deflector [15] normally used to inject the particles in the Delay Loop. Recently, a new detector has been commissioned, in order to measure the bunch length in a non-destructive manner. It is based on microwave spectrometry, which we commonly refer to as the "RF-pickup" [16]. In this project, both the RF deflector and the RF-pickup measurement techniques will be presented and their measurement results compared to simulation.

4-1. Introduction

The result of the PARMELA [17] code for the CTF3 injector [18] was used as the input for both the PLACET [19] simulation code and a 1-D analytical model using the MathCAD [20]. The PARMELA model provides us with the transverse and longitudinal particle distributions, see Fig. 4-2, which can be used as initial conditions for the other codes, at three points (see Fig. 4-1): a- End of the girder 2 b- End of the injector c- End of the chicane. Our work is focused on the longitudinal beam dynamics, but PLACET, with the complete initial conditions from PARMELA, provides for the full 3D dynamics and can be used for the transverse studies as well [21].

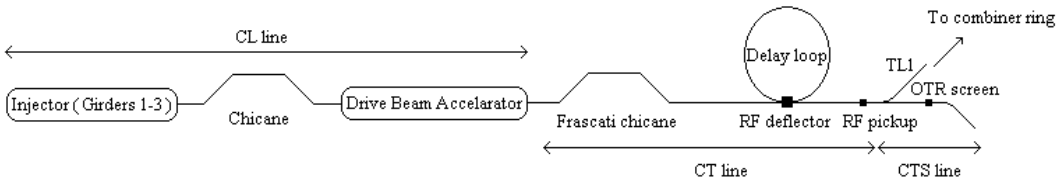

Figure 4-1: Layout of CTF3 from the injector to the combiner ring.

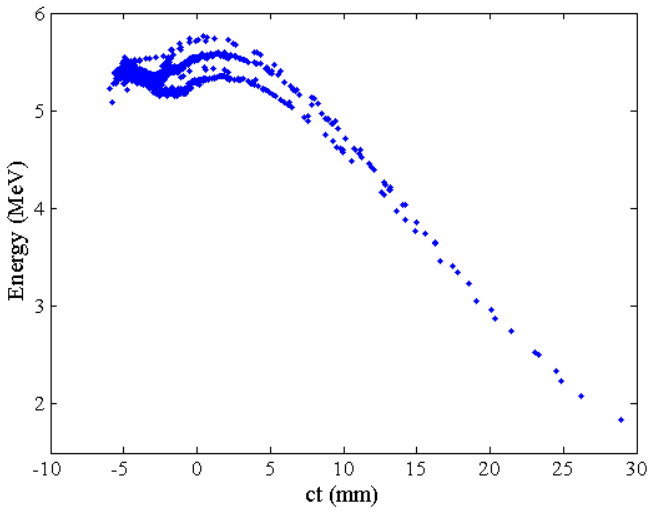

Figure 4-2: The single-bunch longitudinal distribution at the entrance of girder 3, simulated using PARMELA and used as input for MathCAD.

Our project is mainly based on the MathCAD model. This model started from the girder 3, at the entrance of the first accelerating structure of the linac. The overlap between the PARMELA and the MathCAD model gave us the opportunity to assess the importance of different phenomena by comparing the result of the two models, as described later.

Using the MathCAD and PLACET models, the CL (CTF3 Linac, from girder 3 in MathCAD and from girder 5 in PLACET) and the CT and the CTS lines were simulated (See Fig. 4-1). In Table 4-1, a comparison of the different simulation codes is given.

The bunch length, for different conditions, was measured in the CT line, by using the RF deflector of the delay loop and an OTR screen and was compared to the result of the simulation. The bunch length was also measured by the RF-pickup cavity at the end of the CT line.

Table 4-1: Simulation Codes Comparison

PARMELA	MathCAD	PLACET
Injector, Chicane	Girder 3, Chicane, DBA*, CT & CTS lines	DBA, INFN-Frascati Chicane
Space charge	No space charge	No space charge
No wake field	Wake field	Wake field
Longitudinal & Transverse	Longitudinal	Longitudinal & Transverse
No CSR	No CSR	CSR

4-2. Injector and chicane

In the Injector, the electron beam is produced by a DC electron gun, and then bunched either at 1.5 or $3\,\mathrm{GHz}$ by the bunching system and is accelerated to about $18\,\mathrm{MeV}$.

The chicane (in the girder 4) is used for two reasons:

-To cut the low energy tail of the bunch: Given that each particle in the bunch, travels on a different trajectory related to its energy, then by using an adjustable slit in the chicane it is possible to stop the particles of a chosen energy, see Fig. 4-3.

-To change the bunch length: As a consequence of each particle within the bunch, having a momentum p_i, different from the average momentum $\bar{p}$, the path length difference for each particle, Δz_i, can be described to first order by

$$\Delta z_i = R_{56}(p_i - \bar{p})/\bar{p}, \qquad (4\text{-}1)$$

* Drive Beam Accelerator

where R_{56} is the chicane momentum compaction factor. For the chicane in the injector, R_{56} was equal to -15.2 mm. Hence, particles with a higher energy travel on a shorter trajectory than lower energy. And depending on this energy-time correlation within the bunch, the chicane can make the bunch shorter or longer. It is possible to change such correlation by changing the value of the R_{56} parameter in Eqn. 4-1.

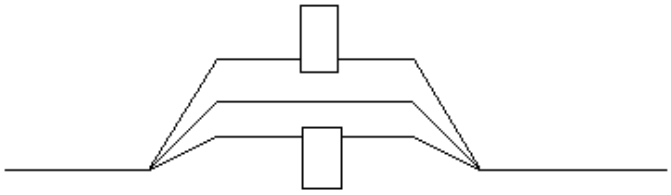

Figure 4-3: Mechanism for cutting the bunch tail by an adjustable slit.

As mentioned before, we have the result of the PARMELA model in three points, namely the end of the girder 2, the end of the injector and the end of the chicane. Since the MathCAD model starts from the end girder 2, we can hence compare the result of the PARMELA model and the MathCAD model after the injector and after the chicane. This enables us to check the relative importance of effects such as space charge and short-range wake fields and to check the difference between the chicane modeling in these codes.

In addition, the MathCAD model doesn't include the space charge effect and hence we can assess its magnitude.

For the comparison we start from the girder 3. The input is from PARMELA. To model the acceleration in girder 3 we neglect for the moment the short-range wake fields and use the simple equation for a momentum gain of Δp_i for a given particle i with time t_i with respect to the input RF arrival time of $\varphi_{in}/2\pi f_{in}$, as described by

$$\Delta p_i = \Delta P \cos(2\pi f_{in} t_i + \varphi_{in}) \qquad (4\text{-}2)$$

Where $f_{in} = 3$ GHz is the RF frequency and ΔP the maximum momentum gain of an electron travelling on crest ($\Delta P = 13.27$ MeV /c for girder 3).

Using this model, a comparison between the MathCAD and PARMELA model, after the injector, is made see Fig. 4-4. This comparison shows that the difference in average energy and bunch length is small. The space charge force pushes the energy of the head and tail of the bunch respectively up and down by less than 1%. The space charge effect is therefore insignificant in other parts of the machine, where the energy is higher.

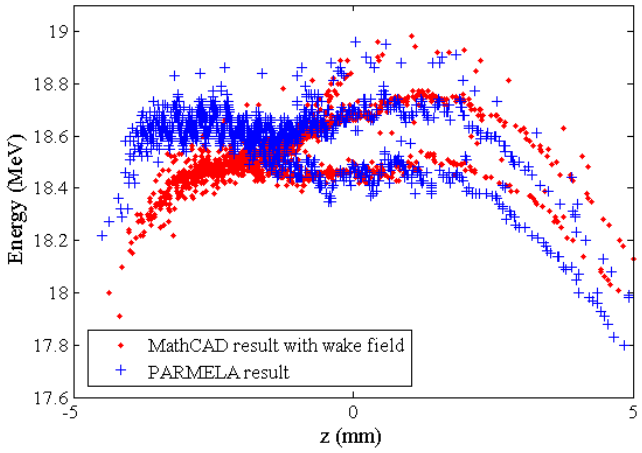

Figure 4-4: The single-bunch longitudinal distribution simulated using MathCAD and PARMELA after the injector, between z=-5 mm and z=5 mm, particles for which |z| > 5 mm will be cut in the chicane of girder 4.

Fig. 4-5 shows the difference between the chicane modelling in the PARMELA and the MathCAD model is negligible, and hence we can use the MathCAD model as a basis for the simulation for the rest of the machine.

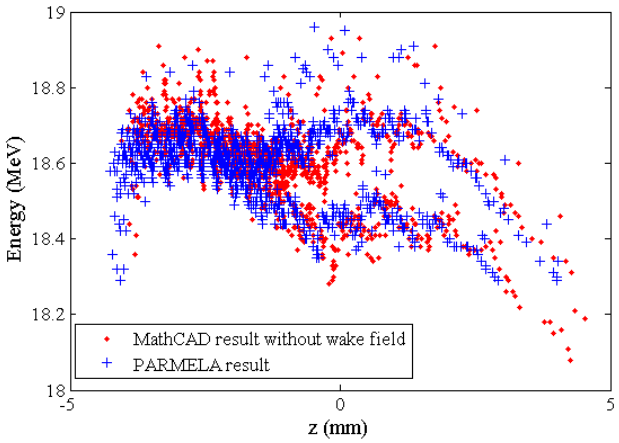

Figure 4-5: The single-bunch longitudinal distribution simulated using MathCAD without the wake field effect and PARMELA after the chicane by the starting from the end of the injector. Input is from PARMELA.

4-3. The wake field modelling in the MathCAD and PLACET

The Karl Bane prescription [22, 23] is used to model the wake field effect in the cavities:

$$W_i\left(\frac{eV}{m}\right) = \frac{q}{n}\left[Z_0 \frac{c}{\pi a^2}\right]\sum_{j=0}^{i} e^{-\sqrt{\frac{S_{ij}}{S_{00}}}}, \qquad (4\text{-}3)$$

Where q is the charge of one bunch, containing n macro particles, Z_0 is the impedance, a, g and l are the geometrical parameters of the cavities as shown in Fig. 4-6. For our simulations, $q = 3.27 \times 10^{-9}$ C, $n = 1420$, $Z_0 = 377$, $a = 0.017$ m, $g = 0.0225$ m, $l = 0.033$ m and $S_{ij} = c\left|t_j - t_i\right|$, where t_i is the arrival time of the macro particle experiencing the force and t_j is the arrival time of the other macro particles that contribute to the force on macro particle i, where $t_j < t_i$ and $S_{00} = \frac{0.41 \times a^{1.8} \times g^{1.6}}{l^{2.4}}$.

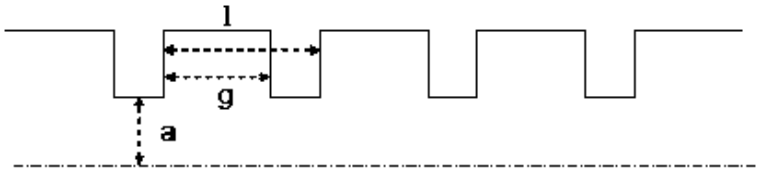

Figure 4-6: Basic shape of 2D periodic cavity structure [24]

Fig. 4-7 shows that the effect of the wake field in girder 3 is not negligible, and should always be included for a more realistic simulation. Because of that we used the MathCAD model result with the wake field effect included as input in the rest simulation, instead of the PARMELA model result at the end of chicane that doesn't contain the wake field effect, even though the final difference to the energy distribution from the wake field after further acceleration in the drive beam accelerator is fractionally negligible, see Fig. 4-8.

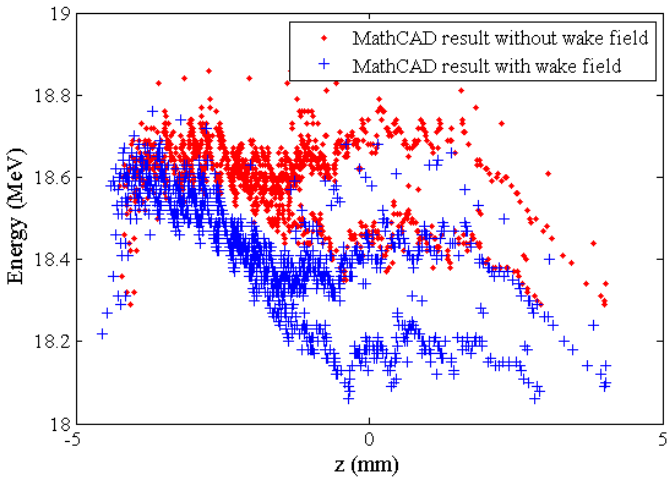

Figure 4-7: The single-bunch longitudinal distribution simulated using MathCAD after the chicane with the wake field effect -which provides the input for the MathCAD model- and without wake field.

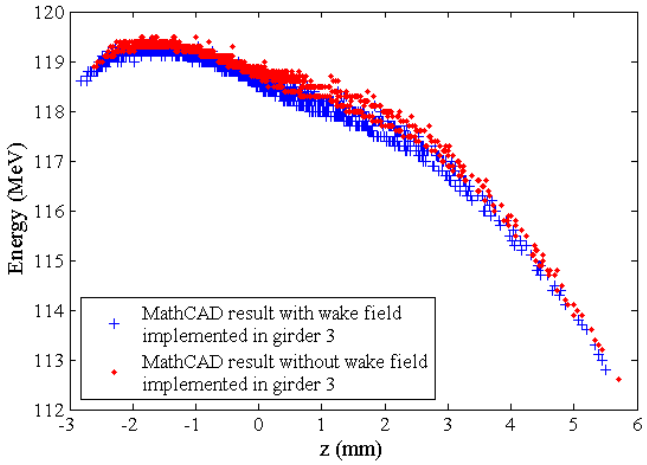

Figure 4-8: The single-bunch longitudinal distribution from the MathCAD model after the DBA with the wake field effect implemented in girder 3 and without the wake field effect. The input is from the MathCAD model.

4-4. Drive Beam Accelerator

In the Drive Beam Accelerator (DBA) the electron beam is accelerated to about $120\,\mathrm{MeV}$ by the structures located in girders 5, 6, 7, 11, 12, 13 and 15. The energy gain of an electron transversing the DBA is $102\,\mathrm{MeV}$ (acceleration on the RF wave crest). Each of these girders has two accelerating cavities and each of these cavities contains 34 cells [5]. The MathCAD and PLACET models were used for this part and they both include the effect of short-range wake-fields.

To model the acceleration in the DBA, using the MathCAD model, Eqn. 4-2 is used with $\Delta P = 14.57\ \mathrm{MeV/c}$ and the phase of the RF is chosen to be on crest, i.e. $\varphi_{in} = 0$, for the accelerating structures on girders 5-13 and $f_{in} = 3\ \mathrm{GHz}$. For the accelerating structures on girder 15, the RF phase φ_{in} can be adjusted in order to manipulate the bunch length after the INFN-Frascati chicane. Fig. 4-9 shows the longitudinal single bunch distributions for on-crest acceleration at the end of the DBA for the MathCAD and the PLACET models. The average energy in the two distributions differs by

about $0.085\ \text{MeV}$, which is negligible and shows a good agreement between these two models. For the rest of the simulation of the machine we used the MathCAD model.

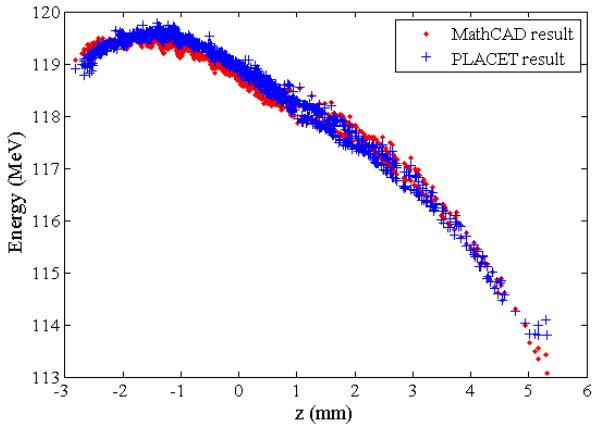

Figure 4-9: The single-bunch longitudinal distribution simulated using the MathCAD and the PLACET models after the DBA. The input is from the PARMELA model. The wake field effect is included for both.

The off-crest acceleration is shown in Fig. 4-10, where the resulting difference in energy between the on crest and the off crest acceleration of $\varphi_{in} = -20^{\circ}$ is about $2\,\text{MeV}$ which corresponds to about a 6% difference in energy gain for the beam in the final two accelerating structures of girder 15. The bunch length is conserved to be $\sigma = 1.62\ \text{mm}$ at the end of DBA for both cases.

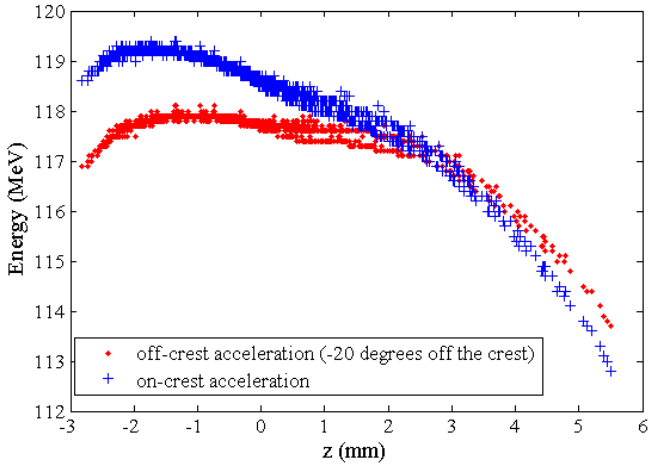

Figure 4-10: The single-bunch longitudinal distribution from the MathCAD model before the INFN-Frascati chicane for on-crest acceleration, $\varphi_{in}=0^{\circ}$ and for off-crest acceleration by $\varphi_{in}=-20^{\circ}$ in girder 15. The input is from the MathCAD model.

4-5. INFN-Frascati Chicane

The INFN-Frascati Chicane is used to change the bunch length. Contrary to the chicane located in the injector, this four-bend chicane contains quadrupoles which can be used to tune its momentum compaction [15]. For the bunch length measurements presented in this thesis, the quadrupoles inside the chicane were turned off and thus we refer to this setting as "natural''. In such a configuration, the value of the momentum compaction is $R_{56} = 0.46$ m (calculated by the MadX [25] and the PLACET models). The optimum bunch length after the chicane should be about 2 mm r.m.s. so as to be not too short in order to minimize coherent synchrotron radiation in the delay loop and the combiner ring and not too long in order to have an acceptable injection into the delay loop and the combiner ring. Figs. 4-10 and 4-11 show the result of MathCAD model before and after the INFN-Frascati chicane.

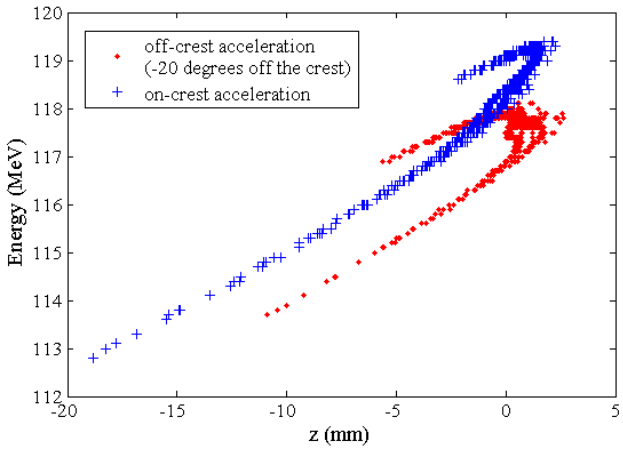

Figure 4-11: The single-bunch longitudinal distribution from the MathCAD model after the INFN-Frascati chicane for on-crest acceleration, $\varphi_{in}=0^{\circ}$ where $\sigma=2.43$ mm and for off-crest acceleration by $\varphi_{in}=-20^{\circ}$ where $\sigma=1.47$ mm in girder 15. The input is from the MathCAD model.

4-6. Calculating the R_{56} and T_{566} by the MADX and PLACET models

The description of the path length difference, as described in Eqn. 4-1, can be expanded to include a second order contribution as follows:

$$\Delta z_i = R_{56}\frac{p_i-\bar{p}}{\bar{p}} + T_{566}\left(\frac{p_i-\bar{p}}{\bar{p}}\right)^2 \qquad (4\text{-}4)$$

The MADX model of CTF3 was used to evaluate R_{56} by a direct R-matrix calculation, while using the PLACET model, calculated the value of R_{56} and T_{566} by tracking. In this tracking method we set, as an input, a few particles with transverse coordinates on a reference trajectory and the same longitudinal position, but with different energy values. The output contains the longitudinal position, Δz_i of each particle, and the corresponding momentum, p_i. By fitting the correlation between the Δz_i and $(p_i-\bar{p})/\bar{p}$ to a second order polynomial, one can obtain R_{56} and T_{566}. Using the data, see Fig. 4-12, R_{56} and T_{566} was calculated to be $0.46\,\mathrm{m}$ and $-1.14\,\mathrm{m}$ respectively. The PLACET model result after the INFN-Frascati chicane is not understood at the moment and will be studied in future work, nevertheless,

we used the PLACET model for calculating the R_{56} and T_{566}. We feel confident in the PLACET model's ability to calculate T_{566} because in the calculation of R_{56} by MADX and PLACET models, the two models were in agreement.

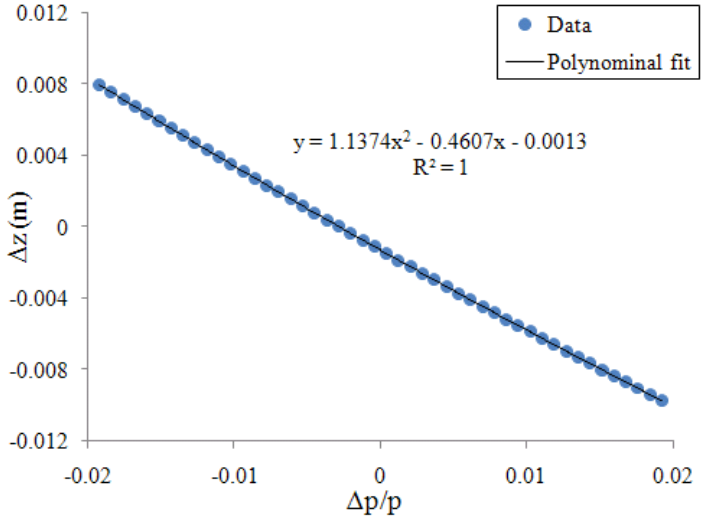

Figure 4-12: The diagram of Δz_i versus $(p_i - \bar{p})/\bar{p}$, showing the extraction of the values of the transfer matrix elements $R_{56} = 0.46$ m and $T_{566} = -1.14$ m by fitting the data from the output of the PLACET simulation to a second order polynomial.

4-7. Coherent Synchrotron Radiation effects

The coherent synchrotron radiation (CSR) effects were investigated for the chicane located after the CTF3 linac. The CSR effects have recently been added to the simulation code PLACET [26], following the approach taken by Elegant [27]. The PLACET implementation simulates free-space CSR. The vacuum pipe diameter is in the order of the critical aperture $(\pi\sigma_z\sqrt{R})^{\frac{2}{3}}(36\,\mathrm{mm})$ meaning that the effect of shielding might be non-negligible. The CSR effects, found by PLACET simulations, should therefore be interpreted as an upper limit for the Chicane CSR. Analytical estimates [27] based on formulas for steady-state CSR in bends, for a

Gaussian bunch, indicate that the mean energy loss ($<\delta>=-(0.35 r_e QL)/e\gamma(R^2\sigma_z^4)^{\frac{1}{3}}$) due to CSR effects will be less than 1 per mille of the mean energy. Fig. 4-13 shows the results of PLACET simulations for a test distribution of arbitrary bunch shape, and a bunch length of around 1.6 mm, with and without CSR effects (CSR simulated in bends only) included. The simulations confirm sub per mille relative energy loss. We therefore do not expect significant effects of CSR in the INFN-Frascati chicane for the measurements described in this thesis.

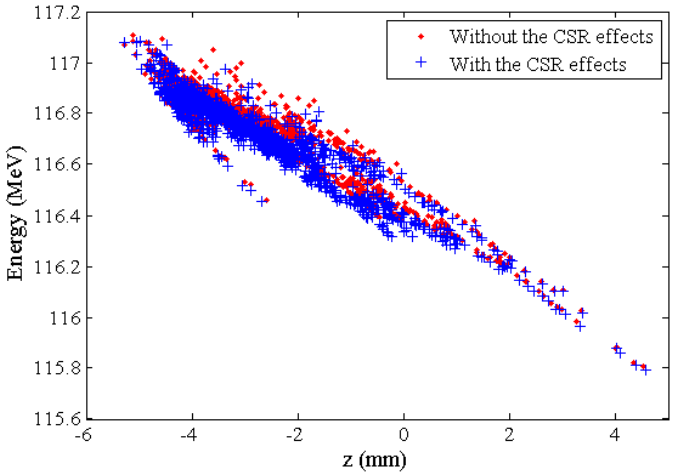

Figure 4-13: The longitudinal particle distribution after the chicane with and without the CSR effects (σ=1.6 mm).

4-8. Technical notes about the PLACET model

Appendixes III and VI show the codes which were used to simulate the DBA and INFN-Frascati chicane in the PLACET.

The "run.tcl" file that is shown in appendix III mainly does these works:

1- Get the input particle distribution data from the "coord_entrance_AS0530_sb.prn" file that is produced by the PARMELA.

2- Convert it to the PLACET data file. Table 4-2 compares the formats of PLACET and PARMELA particle distribution data.
3- Set the lattice from the "linac.tcl" file that is shown in appendix IV.
4- Set the initial beam for this lattice.
5- Run the simulation.
6- Save the particle distribution after the DBA in the "before_chicane.dump" file.
7- Save the particle distribution after the INFN-Frascati chicane in the "output.dump" file.

Table 4-2: Comparison of the variables in PARMELA and PLACET particle distribution data file, in the order they appear in each file.

PARMELA particle distribution file format	PLACET particle distribution file format
x (cm)	E (GeV)
x′ (m rad)	x (μm)
y (cm)	y (μm)
y′ (m rad)	z (μm)
z (degree)	x′ (μ rad)
E (MeV)	y′ (μ rad)

The "linac.tcl" that is shown in the appendix IV is the lattice of DBA and INFN-Frascati chicane.

In section 4-4 it was mentioned that each accelerating girder in DBA (5, 6, 7, 11, 12, 13 and 15) has two accelerating cavities, 34 cells each. The first and final cells have the same geometrical properties which are different from the inner 32 cells; inner cells have the same geometrical properties.

Fig. 4-14 shows the electric field and the energy of an electron which travels on the crest of the electromagnetic field (appendix V). The energy

gain in one cavity is nominalized to 7.97 MeV. In case of different energy gain a linear renormalization would be sufficient.

Table 4-3 shows the energy at the end of the girders which contain the accelerating cavities. It also shows the energy gain and the re-normalized factor equal to the ratio of real energy gain to nominal energy gain.

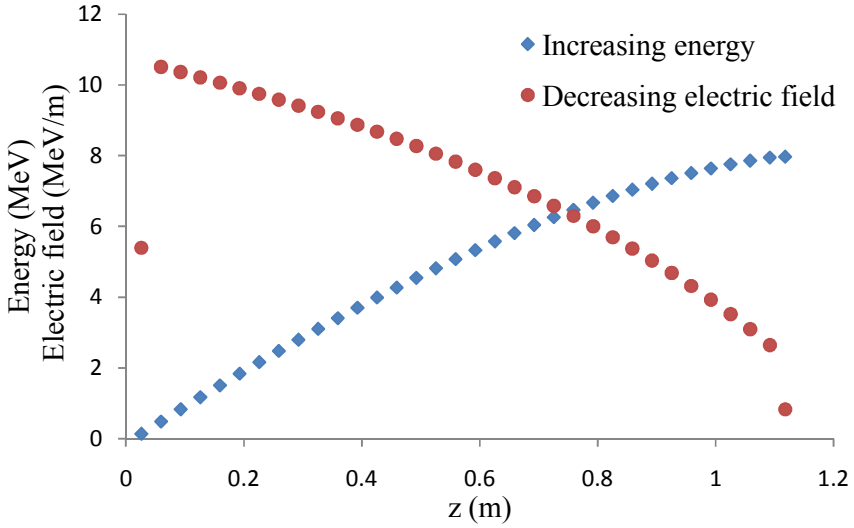

Figure 4-14: Energy of one electron that travels on the crest and electric field in one cavity in the place of each cell.

Table 4-3: Energy gain in each girder in the DBA

Structure	Energy (MeV)	ΔE (MeV)	ΔE/15.94 (MeV)
After Injector	20	--	--
Girder 5	39	19	1.19
Girder 6	56	17	1.07
Girder 7	71	15	0.94
Girder 11	85	14	0.88
Girder 12	98	13	0.82
Girder 13	111	13	0.82
Girder 15	122	11	0.69

Two things about the INFN-Frascati chicane should be mentioned here:

1- The quadrupoles "QDD0220", "QFE0230", "QDF0245" and "QFE0250" which are located the INFN-Frascati chicane, while turned off, can result to the natural mode in which $R_{56} = 0.46$ m and $T_{566} = -1.14$ m .

2- When "sbend_synrad" (for incoherent synchrotron radiation) and "sbend_csr" (for coherent synchrotron radiation) variables are not zero, coherent and incoherent synchrotron radiation will be calculated. These variables are used as the parameters of the bending magnets. "nbins_val" and "nsectors_val" set the step size for the bunch and the bending magnet respectively.

To know more about the details of PLACET commands see [19].

4-9. Bunch length measurement using the RF deflector and OTR screen

The RF deflector is a cavity in the TE mode that gives a transverse kick to the electron bunch. The RF deflector is nominally used to inject the electron beam into the delay loop, but if it is used near the zero crossing of the RF, it is useful for the bunch length measurement, making a correlation between transverse and longitudinal positions in the bunch. (See Fig. 4-15). Therefore the transverse shape of the beam on the OTR screen contains the information of longitudinal shape [28, 29 and 30].

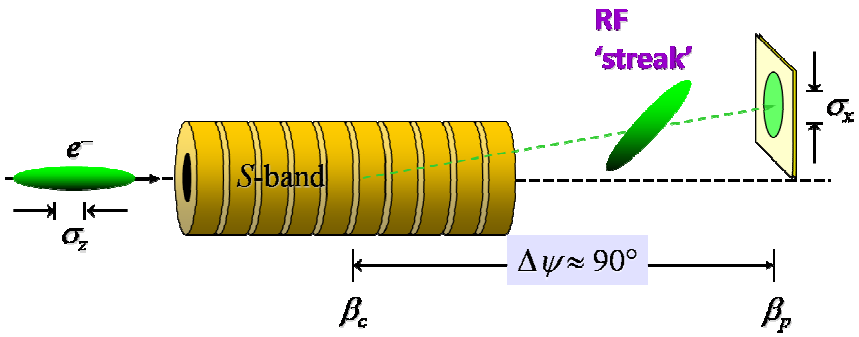

Figure 4-15: Using RF deflector for bunch length measurement [31].

Nominally, at CTF3, the beam enters the RF deflector with a phase near the crest of the input RF deflector wave in order to inject the beam into the delay loop. However, for the bunch length measurement, the beam enters the RF cavity near the zero crossing of the input RF deflector wave. In this case the beam experiences a small transverse kick depending on the input wave power (The RF deflector klystron can be operated between 18 and 27 MW). This kick makes a correlation between the longitudinal and transverse position for each particle within the bunch that translates the longitudinal information to the transverse shape as measured on the OTR screen.

For a good measurement, the magnitude of beta should be near zero at the place of RF deflector and should be large enough downstream at the position of the OTR screen. The betatron phase advance between these two points should be around 90°. This phase advance can be achieved by changing the quadrupole strengths of the magnets after the DBA. In addition, the dispersion after the DBA should be zero. MADX was used calculate these optical parameters. In our case R_{56} is "natural'', and equal to $0.46\,\mathrm{m}$. The magnitude of T_{566} in this case is $-1.14\,\mathrm{m}$.

During the bunch length measurement, the RF deflector was powered and phased such that the center of the bunch arrived close to the RF zero

crossing. In this configuration, the head and the tail of each bunch were then kicked in opposite directions. Because the 3 GHz bunching of the beam during this measurement campaign, and the RF deflector operating at 1.5 GHz , the phase of the RF deflector was adjusted slightly off zero crossing, to image the two beam spots separately on the one screen.

Once the two beam spots were separated using the appropriate phase of the RF deflector, a horizontal corrector magnet was used in order to move one beam spot to the center of the screen, in order to maximize the light collection from the optical line. The second beam spot was therefore out of the acceptance of the screen. The increase in the observed transverse beam size, when the RF deflector was switched on was used to determine the bunch length, see Fig. 4-16.

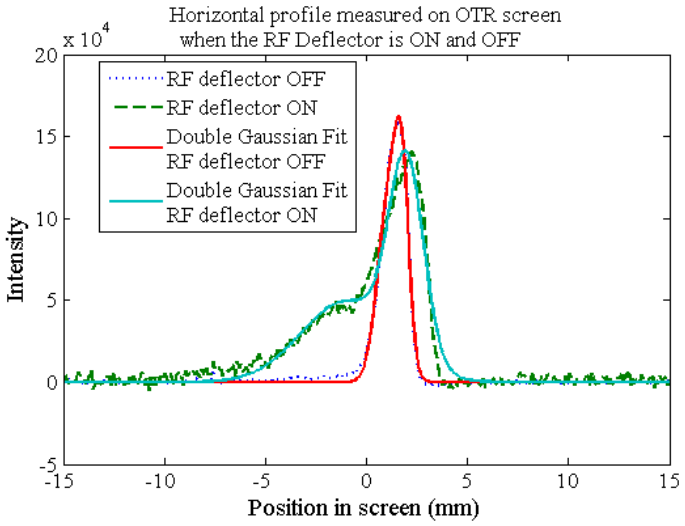

Figure 4-16: The horizontal profile on screen when the RF deflector is ON and when it is OFF; $\sigma >> \sigma_0$.

The bunch length was calculated using the RF deflector and the size of the image on the OTR screen, using two different methods, namely, “single image method” and “scan method”, which will be described below.

4-10. RF deflector: Single image method

In the "single image" method, the horizontal size of the beam on the screen is measured when RF deflector is on (σ) and off (σ_0), as shown in Fig. 4-16. With the following equation we can calculate the bunch length, σ_z, as shown in Fig. 4-15.

$$\sigma_z = \frac{1}{2\pi}\frac{c}{f}\frac{E}{|eV_0 \sin\Delta\psi\cos\varphi|}\sqrt{\frac{\sigma_x^2 - \sigma_{x0}^2}{\beta_d \beta_s}} \qquad (4\text{-}5)$$

where V_0 is the peak voltage of RF deflector input wave, $\Delta\psi$ is the phase advance, φ is the wave phase in the middle of the bunch, E is the beam energy and β_d and β_s are the magnitude of beta function in the middle of the RF deflector and at the position of the OTR screen respectively.

We hence choose to define the calibration factor as follows:

$$\sigma_z \equiv \frac{\sqrt{\sigma_x^2 - \sigma_{x0}^2}}{\text{calibration factor}} \qquad (4\text{-}6)$$

Instead of determining the magnitude of each parameter in Eqn. 4-5, the calibration factor is calculated by a measurement of the deflected beam position on the OTR screen as a function of the RF deflector phase. The calibration factor is the measured linear relation of these quantities, for a given RF power, and an example of such a calibration curve is shown in Fig. 4-17. The phase of the RF deflector is related to the bunch length by the c/f factor, where c is the speed of light and f is the frequency of the RF deflector. For this measurement, $f = 1.5\ \text{GHz}$.

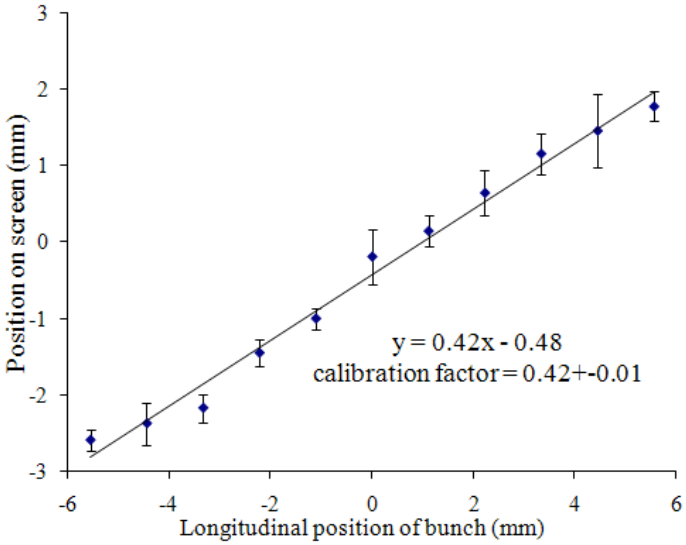

Figure 4-17: The calibration factor is calculated based on the linear relation of the screen position unit and the longitudinal position unit, for a given power of the RF deflector.

The bunch length measurement was performed on two occasions, using this method, and compared to three different MathCAD simulations, see Fig. 4-18. The simulations have different input distributions and cut different parts of the low energy tail in the first chicane. The r.m.s. bunch length is sensitive to the energy cut in the first chicane, as shown in Fig. 4-18, where the results of the bunch length is shown for a cut of the low energy bunch tail of 10.4% - in the machine - and 9%.

We suspect that the difference between the three simulations, as shown in Fig. 4-18, is greater here than it would be if the three cases were measured in the machine. This is because, in the simulation, the r.m.s. of the distribution is calculated and plotted, which is more sensitive to cuts in the tail of the distribution, whereas in the measurement, we fit to the full distribution, and use the resulting standard deviation, in order to extract the bunch length, and are hence, less sensitive to the tail. Future work will be devoted to treating the simulation and the measurement in a more consistent manner.

In the measurement performed on the 12th September 2008, a 22° phase difference was noticed between the measured crest and the crest that is to be expected, based on simulation. This difference can be explained if the bunch doesn't travel on the crest in previous girders. A 3° phase shift off the crest, in each previous girder, can result in such a difference.

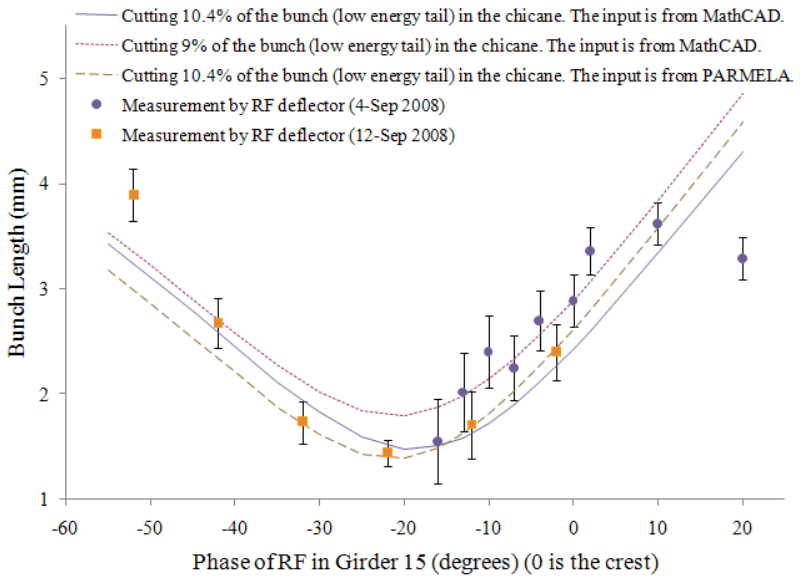

Figure 4-18: Comparison of bunch length measurement result with the single image method and the simulation result from MathCAD model.

4-11. Double Gaussian fit

Fig. 4-16 shows that the double Gaussian fit is a very good candidate for fitting the shape of the image profile. However, it must be noted, that the shape of the bunch profile at the RF deflector depends on the phase of the RF in girder 15, see Fig. 4-11. Our result in Fig. 4-18 is based on the single Gaussian fit. The result doesn't show a big difference when compared to the double Gaussian fit. For double Gaussian Eqn. 4-5 is not valid, however, in the limit of $\sigma_x >> \sigma_{x0}$ then the equation becomes $\sigma_z = \sigma_x$ / calibration_factor and is valid for both the Gaussian and double

Gaussian fit. Future work will be dedicated to studying in more detail the single bunch shape using the Streak camera [14] which has the advantage over using the RF deflector which measures the average bunch length measurement over the full bunch train.

4-12. RF-deflector: Scan method

In this method the average intensity of a thin band on the middle of the OTR screen versus RF deflector phase is measured, as shown in Fig. 4-19. The standard deviation of the Gaussian fit of the result of the intensity vs. phase scan when multiplied by the c/f factor is the bunch length, see Fig. 4-20. In this method we suppose that $\sigma_x >> \sigma_{x0}$ and this can be reached by changing some parameters in the machine. A series of measurements, using this technique was performed in May 2007, see Fig. 4-21, and the result were averaged to have a measured bunch length of 2.45 ± 0.28 mm.

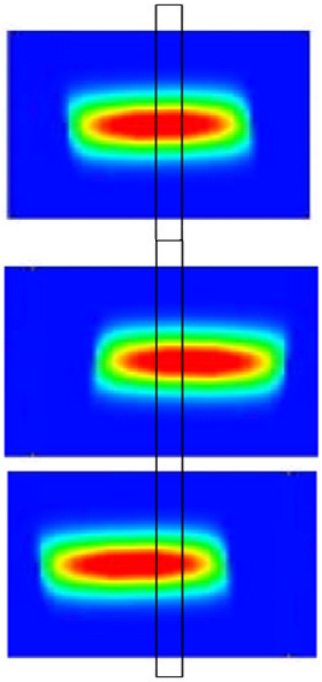

Figure 4-19: Measurement of the average intensity of a thin band on the middle of OTR screen.

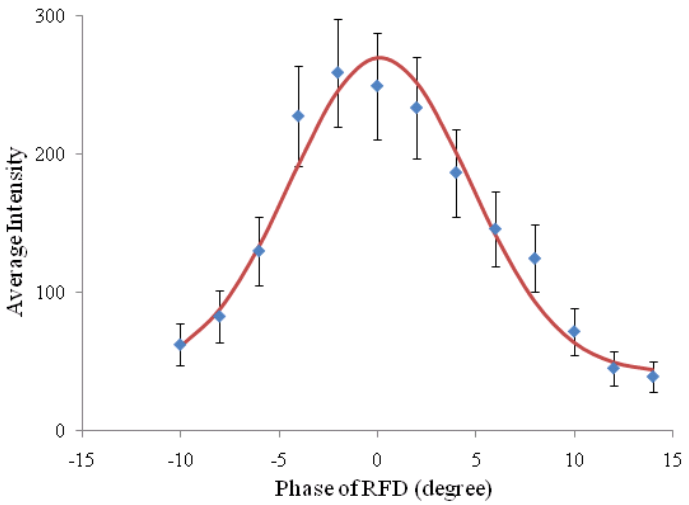

Figure 4-20: An example of one measurement and the corresponding Gaussian fit superimposed. The bunch length = standard deviation * c / f = 2.84 ± 0.35 mm.

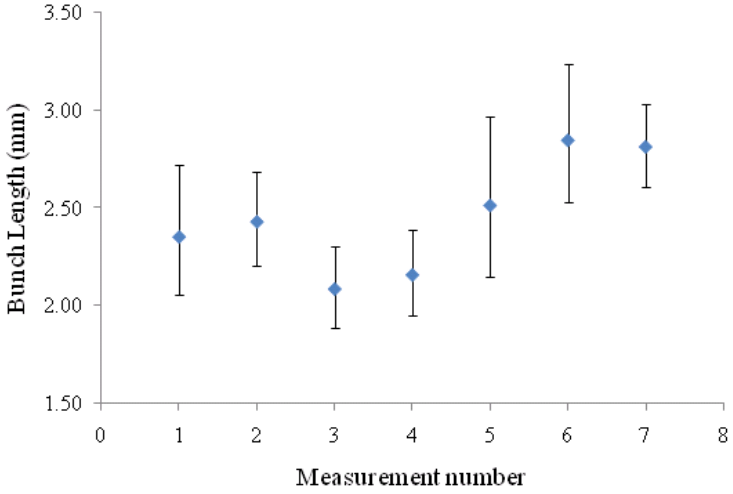

Figure 4-21: Bunch length measurement on May 2007, using the "Scan" method. The average bunch length = 2.45 ± 0.28 mm. The phase of the RF in girder 15 was kept constant for all 7 measurements.

The "Scan" method has an advantage over the "Single Image" method because the same position of the screen is used, hence the systematic effects associated with the non-linear effects from the screen, are reduced. Another advantage is that the image can be bigger than the screen, which results in an improved resolution. One disadvantage is the need to assume

that the bunch shape remains constant during the scan and also the sensitivity to jittering is another disadvantage.
We used the “single image” method to compare to the simulations, because it was less sensitive to jittering and need less time to perform the measurement.

4-13. RF-pickup detector method

A non-destructive single shot bunch length monitor, the so-called “RF-pickup”, measures the frequency spectrum of the electromagnetic field emitted by the particles and collected by a rectangular K_a waveguide. The RF-pickup was installed $50\ \text{cm}$ upstream of the OTR screen, see Fig. 4-1, and hence cross calibrations can be performed between the RF-deflector and the RF-pickup monitor. This monitor has a sub-ps time resolution and, under the assumption of a certain bunch shape, calibration is done in a self-consistent manner. Moreover, this monitor has the advantage of being non-destructive and relatively inexpensive compared to other techniques.
The RF-pickup consists of a single WR-28 waveguide connected to the beam pipe. A $0.5\ \text{mm}$ thick CVD diamond window [32] is used to isolate the vacuum in the beam pipe from the atmospheric pressure in the waveguide. Signal frequencies above the cut-off of the WR-28 waveguide ($21\ \text{GHz}$) are transported in a continuous WR-28 waveguide for about $18\ \text{m}$ to the detection station in a technical gallery.

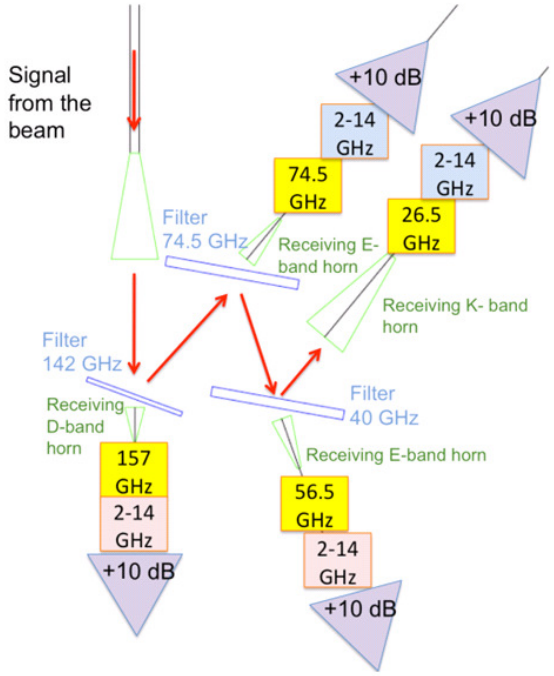

Figure 4-22: Schematic of the detection system for the RF pickup

At the detection station, the RF signal from the beam is emitted using a K_a band horn antenna, as shown in Fig. 4-22. The detection system is designed to measure the amplitude of the RF signals from the beam, simultaneously in four frequency bands, namely $26.5 - 40$ GHz, $45 - 69$ GHz, $75 - 90$ GHz and $142 - 170$ GHz.

As shown schematically in Fig. 4-22, two down-mixing stages in series are required in order to measure the high frequency RF signals, given the bandwidth constraints imposed by our digitising system. The first down mixing stage has a fixed local oscillator frequency for each band, namely 26.5 GHz, 56.5 GHz, 75 GHz and 157 GHz. The second down mixing stage is in common to each of two of the four detection bands, obtained by using two synthesizers with a variable frequency range of $2 - 14$ GHz. From this setup, measurements of the beam harmonics of 30 GHz, 33 GHz, 36 GHz and 39 GHz are made using the K-band detection, the beam harmonics of 60 GHz, 63 GHz, 66 GHz and 69 GHz are made using the first E-band

detection stage, and the beam harmonics of $78\ \mathrm{GHz}$, $81\ \mathrm{GHz}$, $84\ \mathrm{GHz}$ and $87\ \mathrm{GHz}$ are made with the second E-band detection stage. The D-band detection stage provides signals only for beam conditions with very short bunches, which was not the case for the measurements presented in this thesis. The signals are amplified by $+10\ \mathrm{db}$ after the second down mixing stage, and then digitized using a fast Acqiris digitizing scope with $2\ \mathrm{Gs/s}$ per channel. The data acquisition is controlled remotely by a LabView [33] program, which stores, displays and analyses the signals in real time.

4-14. Bunch length measurement using RF-pickup detector

For each machine condition, which corresponds to a particular setting of the phase of the last Klystron (MKS15) of the CTF3 linac, 15 successive measurements are stored and their Fourier transforms performed. The mean height of the peak, corresponding to the power of each $3\ \mathrm{GHz}$ beam harmonic of interest, is measured and used for the bunch length determination. The response was also normalised to the fluctuations in the beam current and position. An example of the response of the $30\ \mathrm{GHz}$, $60\ \mathrm{GHz}$ and $78\ \mathrm{GHz}$ signals, normalized to a phase of MKS15 -20° from the crest, as a function of the phase of the MKS15 is shown in Fig. 4-23.

The error bars of the response, see Fig. 4-23, are the standard deviation of the 15 successive measurements, and indicate the shot to shot jitter of the machine. The response of the system is inversely proportional to the bunch length, and hence the minimum bunch length is measured to be between -25° and -20°, using the RF-pick response which is consistent with the simulation and RF deflector measurement, see Fig. 4-18.

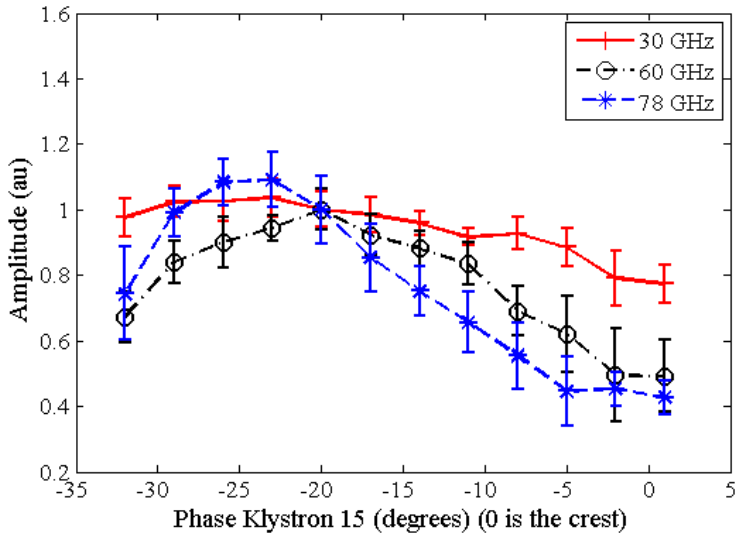

Figure 4-23: An example of the response of the RF pickup for beam harmonics of 30 GHz, 60 GHz and 78 GHz as a function of the phase of the last Klystron in the CTF3 linac.

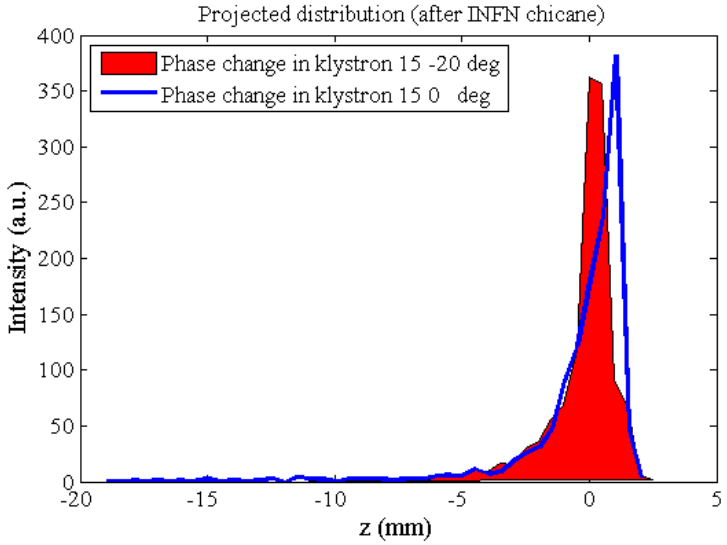

Figure 4-24: A simulation of the single bunch shape distribution, after the INFN-Frascati chicane in the CT line, as obtained by projecting the distribution in Fig. 4-11.

The amplitude of each beam harmonic measured, as a function of the phase of MKS15, is used in a fitting procedure to extract the bunch length. The longitudinal distribution of the electron bunch is assumed to be single Gaussian distribution. However the bunch shape depends on the phase of MKS15, see Fig. 4-11 and Fig. 4-24, and hence the assumption of a single

Gaussian distribution, for all settings of MKS15, is only an approximation. Hence, we do not expect the fitting procedure using this single bunch shape assumption to be accurate for all phases of MKS15. Nevertheless, we perform a χ^2 (chi-squared) minimization fit, to the Gaussian function, with the fit parameters being the r.m.s. bunch length at each machine setting, and the response factor of each frequency band.

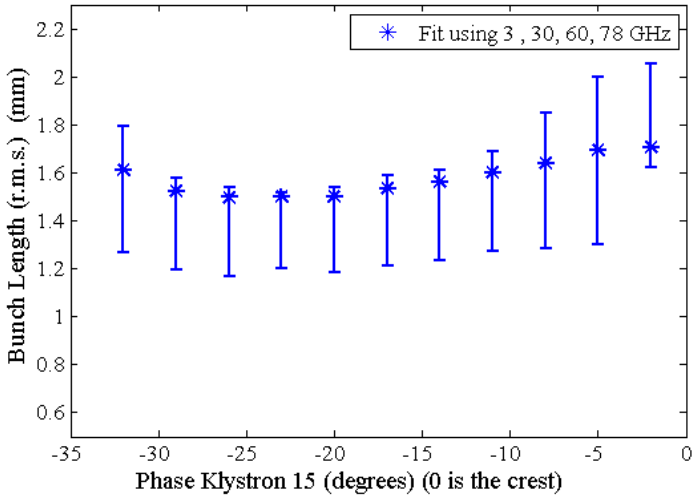

Figure 4-25: Bunch length measurement using the RF pickup and 3 GHz beam signal from the BPR as extracted using a χ^2 fit and a Gaussian distribution assumption on the bunch shape.

The result of the bunch length measurement is shown in Fig. 4-25. The fit indicates a minimum bunch length of about 1.5 mm (r.m.s.), at a phase of -20° from the crest. This is consistent with the measurement of the RF deflector and the prediction of the simulation. The range of the measurement was limited to about 35° of MKS15, because this is the range over which the beam current is conserved and the losses in the transport through the chicane is less than 1%. The error bars assigned to the measurement in Fig. 4-25 correspond to the amount of change attributed to the respective parameter in order for the χ^2/ν of the fit to increase by one unit. The error bars are asymmetric, owning the asymmetrical shape of the chi-squared distribution.

4-15. Comparison of RF deflector and RF-pickup methods results with simulation

The bunch length measurements, obtained using the RF deflector and the RF-pickup, are compared to the simulation as described in Sec. II. The simulation and the measurements from the RF deflector and the RF-pickup seem to be in agreement, see Fig. 4-26, although the RF pickup measurement was performed in a limited range of phase. The position of the minimum bunch length is in common between both measurement and the simulation.

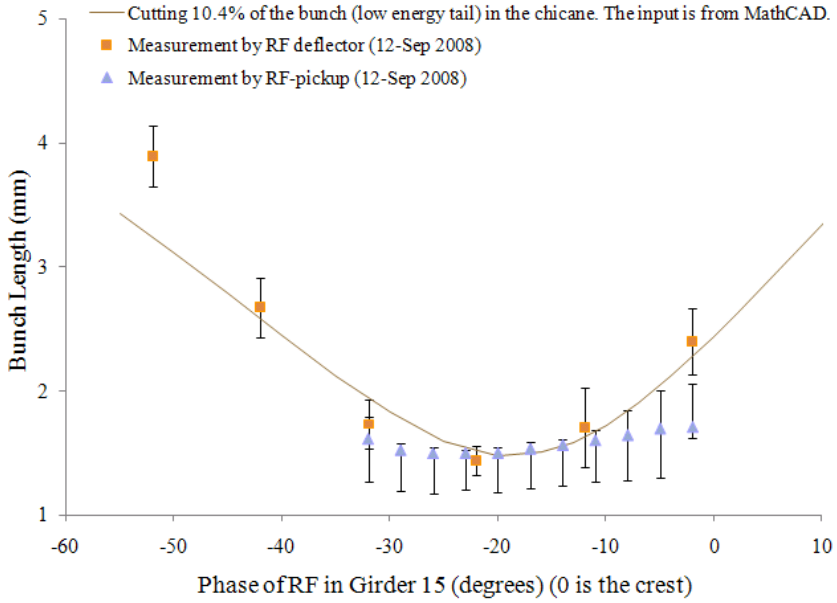

Figure 4-26: Bunch length measurement as obtained using both the RF deflector and the RF-pickup and compared to simulation.

4-16. Conclusion

The longitudinal beam dynamics has been simulated at CTF3 from the injector to the CTS line, and the results compared to measurements. An agreement between the simulations and the bunch length measurement using the RF deflector and the RF-pickup is shown in Fig. 4-18 and Fig. 4-

26. The errors in the RF deflector measurements are mainly due to the jittering of beam on the screen, where the RF deflector and the electron gun are the main source of this jittering. The non-linearity of OTR screen also contributed to the errors. The measurement with the RF-pickup shows an agreement with the RF deflector and with the simulation, within the limited range of the measurement and under the assumption of a single Gaussian bunch shape. Future experimental work is planned in order to study the single bunch shape distribution, which is needed in order to do a more reliable self-calibration of the RF pickup. A study of the bunch shape is also important in order to treat the bunch length calculation in the simulation and measurements in a more self-consistent manner. Future simulation work will be dedicated to checking the PLACET result after the INFN-Frascati chicane, and expanding the simulation code to include the delay loop, combiner ring and TL2.

5. REFERENCES

[1] http://sl-div.web.cern.ch/sl-div/history/lep_doc.html

[2] H. H. Braun *et al*, "2008 CLIC Parameters", CERN-OPEN-2008-021 Note (CLIC-Note-764)

[3] H. H. Braun *et al*, "Updated CLIC parameters 2005", CERN-OPEN-2006-022 Note (CLIC-Note-627)

[4] H. H. Braun *et al* "The CLIC RF power source – A novel scheme of two-beam acceleration for electron-positron linear colliders", CERN report 99-06 (1999)

[5] G. Geschonke, A. Ghigo (ed.) et al.: "CTF3 Design Report", CERN/PS 2002-008(RF), LNF-02/008(IR).

[6] https://edms.cern.ch/nav/PS-000041

[7] A. Fiebig and Ch. Schieblich, "A SLED type pulse compressor with rectangular pulse shape", Proc. of European Particle Conference, Nice, 1990, p.937

[8] R. Bossart, P. Brown, J. Mourier, I.V. Syratchev, L. Tanner (CERN) "High-power microwave pulse compression of klystrons by phase-modulation of high-Q storage cavities", CLIC-NOTE-592, Jun 2004.

[9] I.Syratchev,"RF Pulse Compression system for CTF3", MDK Workshop, CERN, Geneva, April 2001

[10] F. Di Maio, A. Risso "The CERN-PS Equipment Access Library. Software Specifications", PS/CO/Note 93-87 (Spec), February 94.

[11] http://wwwpsco.cern.ch/

[12] http://wwwpsco.cern.ch/private/mw/RDA/cppapi-1.1/html/index.html

[13] A. Sharma, A.D. Ghodke, G. Singh, Abdurrahim, "Optics Design of Transfer Line-2 for CTF3", CTF3 note 091, (2008)

[14] C. Welsch et al, "Longitudinal Beam Profile Measurements at CTF3 Using a Streak Camera", Journal of Instrumentation, 1, (2006), P09002

[15] A. Ghigo et al, "Commissioning and First Measurements on the CTF3 Chicane", Proceedings of PAC 2005, USA, pp.785

[16] A. Dabrowski et al, "Non Destructive Single Shot Bunch Length Measurements for the CLIC Test Facility 3", PAC 2007, USA, pp. 4069

[17] http://laacg1.lanl.gov/laacg/services/serv_codes.phtml#parmela

[18] A. Yeremian, R.Miller, R.Ruth, H. H. Braun, G. Geschonke,L. Groening, L. Rinolfi, L. Thorndahl, I. Wilson, F. Zhou, "CTF3 Drive-Beam Injector Design",EPAC'02, June 2002, Paris.

[19] https://savannah.cern.ch/projects/placet/

[20] http://www.mathsoft.com

[21] E. Adli, R. Corsini, A.E. Dabrowski, D. Schulte, H. Shaker, P.K. Skowronski, F. Tecker, R. Tomas, "Status of an Automatic Steering for the CLIC Test Facility 3", Proceedings LINAC 2008

[22] Karl L.F. Bane, "Wakefields of Sub-Picosecond Electron Bunches", SLAC-PUB-11829, April 2006.

[23] Karl L.F. Bane, M. Timm, T. Weiland, "The Short-range Wakefields in the SBLC Linac", PAC 1997, May 1997.

[24] R. Zennaro, "Study of the Validity of K. Bane's Formulae for the CLIC Accelerator Structure", EPAC 2008, June 2008.

[25] http://mad.web.cern.ch/mad/

[26] A. Latina, E. Adli, H. Burkhardt, G. Rumolo, D. Schulte, R. Tomas, CERN, Geneva, Switzerland, Y. Renier, LAL, Paris, France, “Recent improvements in the Tracking Code PLACET’’, EPAC 2008, June 2008

[27] M. Borland; Argonne National Laboratory Advanced; Photon Source Report No.LS-287, 2000.

[28] D.Alesini, R.Corsini, D.Filippetto, A.Ghigo, F.Marcellini, B.Preger, “CTF3 Bunch Length Measurement with the 1.3 GHz RF Deflector”, CTFF3-010, Frascati, January 2007.

[29] P. Emma, J. Frisch, P. Krejcik, “A Transverse RF Deflecting Structure for Bunch Length and Phase Space Diagnostics’’, LCLS-TN-00-12, SLAC, August 2000.

[30] D. Alesini, C. Biscari, R. Corsini, A. Ghigo, F. Marcellini, “Longitudinal phase space characterization of the CTF3 beam with the RF Deflector”, EPAC 2004, Lucerne, Switzerland.

[31] P. Emma, “Electron Bunch Measurements with a Transverse RF Deflector”, ICFA XFEL 2004 Workshop, SLAC, July 2004

[32] Brazing done by S. Mathot, CERN EN/MME.

[33] http://www.ni.com/labview/

6. APPENDIX I: PULSE COMPRESSOR SIMULATION CODE IN C++

```
#include <math.h>
#include <complex.h>
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <eqp/eqp.h>c
#include <eqp/eqp_def.h>
#include "cpgplot.h"

#define ON  1
#define OFF 0

int ProcStat=OFF,signStat=OFF;
char *externalFile;
char kNumber[80];
class Input
{
   float T_pulse,T_start,T_step;
   float *kAmp,*mPhase,*cAmp,*cPhase,*corPhase,*corPhase2;
   int high,memoryStatus,res,plsOption;
   Equipment smp1,smp2,smp3,smp4,smp5;
   void delete_all()
   {
      if(memoryStatus)
      {
         delete kAmp;
         delete mPhase;
         delete cAmp;
         delete cPhase;
         delete corPhase;
         delete corPhase2;
         memoryStatus=OFF;
      }
      else
      {
         cout << "Try to delete the memory that not alocated yet" <<
endl;
```

```
         exit(1);
      }
   }
   public:
   float corStart,corFinal;
   Input(float T_pulse,float T_start,float T_step);
   ~Input()
   {
      delete_all();
   }
   float show_T_start()
   {
      return (T_start);
   }
   float show_T_pulse()
   {
      return (T_pulse);
   }
   float show_T_step()
   {
      return (T_step);
   }
   int show_high()
   {
      return ((int)(T_pulse/T_step)+1);
   }
   float amplitude(float t);
   float phase(float t);
   float outAmp(float t);
   float outPhase(float t);
   float correctPhase(float t);
   void readAgain();
   void readAgain(float T_pulse,float T_start,float T_step);
   void readAgain2();
   void improvePhase(float *inputPhase);
   float Input::improvePhase(float input);
};
Input::Input(float T_pulse,float T_start,float T_step)
{
   this->T_pulse=T_pulse;
   this->T_start=T_start;
   this->T_step=T_step;
   cout << this->T_step<<endl;
   memoryStatus=OFF;
   plsOption=0;
   readAgain(T_pulse,T_start,T_step);
   //readAgain2();
}
void Input::readAgain(float T_pulse,float T_start,float T_step)
{
   if(memoryStatus)   delete_all();
   char smp_name1[16]="CK.SVPKI";
   strcat(smp_name1,kNumber);
   strcat(smp_name1,"A-C");
   char smp_name2[16]="CK.SVWFGMKS";
   strcat(smp_name2,kNumber);
   strcat(smp_name2,"-C");
   char smp_name3[16]="CK.SVPSI";
   strcat(smp_name3,kNumber);
   strcat(smp_name3,"A-C");
   cout << smp_name3 << endl;
```

```
   char smp_name4[16]="CK.SVPSI";
   strcat(smp_name4,kNumber);
   strcat(smp_name4,"P-C");
   char smp_name5[16]="CK.SVPKI";
   strcat(smp_name5,kNumber);
   strcat(smp_name5,"P-C");
   double interv;
   double
high1,start1,high2,start2,high3,start3,high4,start4,high5,start5;
   smp1 = EqpCreateByName (smp_name1);
   if (smp1 == NULL)
   {
      printf("Error!\n"); exit(1);
   }
     smp2 = EqpCreateByName (smp_name2);
     if (smp2 == NULL)
   {
         printf("Error!\n"); exit(1);
   }
   smp3 = EqpCreateByName (smp_name3);
   if (smp3 == NULL)
   {
      printf("Error!\n"); exit(1);
   }
     smp4 = EqpCreateByName (smp_name4);
     if (smp4 == NULL)
   {
         printf("Error!\n"); exit(1);
   }
     smp5 = EqpCreateByName (smp_name5);
     if (smp5 == NULL)
   {
         printf("Error!\n"); exit(1);
   }

     interv=T_step;
   double startT=T_start,startT2=T_start-700;
   res = EqpWrite (smp1, EQP_INTERV, plsOption, &interv, 1, NULL, 0);
   res = EqpRead (smp1, EQP_DELAY,   plsOption, &start1,  1, NULL, 0);
   res = EqpWrite (smp1, EQP_STRT,   plsOption, &startT,  1, NULL, 0);
   res = EqpRead (smp1, EQP_HIGH,   plsOption, &high1,   1, NULL, 0);
   cout << interv << "hello" <<endl;
   printf("%4.0f data from %f every %f ns\n", high1,start1,interv);
   res = EqpWrite (smp2, EQP_INTERV, plsOption, &interv, 1, NULL, 0);
   res = EqpRead (smp2, EQP_DELAY,   plsOption, &start2,  1, NULL, 0);
   res = EqpWrite (smp2, EQP_STRT,   plsOption, &startT,  1, NULL, 0);
   res = EqpRead (smp2, EQP_HIGH,   plsOption, &high2,   1, NULL, 0);
   printf("%4.0f data from %f every %f ns\n", high2,start2,interv);
   res = EqpWrite (smp3, EQP_INTERV, plsOption, &interv, 1, NULL, 0);
   res = EqpRead (smp3, EQP_DELAY,   plsOption, &start3,  1, NULL, 0);
   res = EqpWrite (smp3, EQP_STRT,   plsOption, &startT,  1, NULL, 0);
//   res = EqpWrite (smp3, EQP_STRT,   plsOption, &startT2,  1, NULL,
0);
   res = EqpRead (smp3, EQP_HIGH,   plsOption, &high3,   1, NULL, 0);
   printf("%4.0f data from %f every %f ns\n", high3,start3,interv);
   res = EqpWrite (smp4, EQP_INTERV, plsOption, &interv, 1, NULL, 0);
   res = EqpRead (smp4, EQP_DELAY,   plsOption, &start4,  1, NULL, 0);
   res = EqpWrite (smp4, EQP_STRT,   plsOption, &startT,  1, NULL, 0);
   res = EqpRead (smp4, EQP_HIGH,   plsOption, &high4,   1, NULL, 0);
   printf("%4.0f data from %f every %f ns\n", high4,start4,interv);
   res = EqpWrite (smp5, EQP_INTERV, plsOption, &interv, 1, NULL, 0);
```

```
   res = EqpRead (smp5, EQP_DELAY,   plsOption, &start5,  1, NULL, 0);
   res = EqpWrite (smp5, EQP_STRT,   plsOption, &startT,  1, NULL, 0);
//   res = EqpWrite (smp5, EQP_STRT,   plsOption, &startT2,  1, NULL,
0);
   res = EqpRead (smp5, EQP_HIGH,   plsOption, &high5,   1, NULL, 0);
   printf("%4.0f data from %f every %f ns\n", high5,start5,interv);

   high=show_high();
   cout << high << endl;
   if(high1<high || high2<high || start1>T_start || start2>T_start ||
high3<high || high4<high || start3>T_start || start4>T_start||
   high5<high || start5>startT2)
   {
      cout << "Error in not adaption between TS and T_pulse with
devices" << endl;
      exit(1);
   }
   if(high1<high || high2<high || start1>T_start || start2>T_start ||
high3<high || high4<high || start3>T_start || start4>T_start||
   high5<high || start5>T_start)
   {
      cout << "Error in not adaption between TS and T_pulse with
devices" << endl;
      exit(1);
   }
   if (!(kAmp =new float[high]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
   if (!(mPhase =new float[high]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
   if (!(cAmp =new float[high]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
   if (!(cPhase =new float[high]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
   if (!(corPhase =new float[high]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
   if (!(corPhase2 =new float[high]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
   memoryStatus=ON;
   readAgain();
}
void Input::readAgain()
{
   res = EqpSetProperty (smp1, EQP_AQN, EQP_TYPE_FLOAT, high);
```

```
   res = EqpRead (smp1, EQP_AQN, plsOption, kAmp, high, NULL, 0);
   res = EqpSetProperty (smp2, EQP_AQN, EQP_TYPE_FLOAT, high);
   res = EqpRead (smp2, EQP_AQN, plsOption, mPhase, high, NULL, 0);
   res = EqpSetProperty (smp3, EQP_AQN, EQP_TYPE_FLOAT, high);
   res = EqpRead (smp3, EQP_AQN, plsOption, cAmp, high, NULL, 0);
   res = EqpSetProperty (smp4, EQP_AQN, EQP_TYPE_FLOAT, high);
   res = EqpRead (smp4, EQP_AQN, plsOption, cPhase, high, NULL, 0);
   res = EqpSetProperty (smp5, EQP_AQN, EQP_TYPE_FLOAT, high);
   res = EqpRead (smp5, EQP_AQN, plsOption, corPhase, high, NULL, 0);
   improvePhase(mPhase);
   FILE *fp;
   fp=fopen(externalFile,"r");
   float a,b,c;
   fscanf(fp,"%f %f %f",&a,&b,&c);
   cout << a << "  "<<b << "  "<<c << endl;
   if(c!=T_step) { cout << "not match between correction file and
program setting" << endl;exit(0);}
   int number=(b-a)/c+1;
   corStart=a;
   corFinal=b;
   cout << number << endl;
   for(int i=0;i<number;i++)
   {
      fscanf(fp,"%f %f",&a,&b);
      corPhase2[i]=b;
   }
   fclose(fp);
}
void Input::readAgain2()
{
   high=show_high();
   if (!(kAmp =new float[high]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
   if (!(mPhase =new float[high]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
   if (!(cAmp =new float[high]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
   if (!(cPhase =new float[high]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
   memoryStatus=ON;
   float amp,x,Te=850;
   float t;
   float ph;
   float af=1.1;
   float T_jump=4600,T_process=4700,phi_jump=124;
   for(int i=0;i<high;i++)
   {
      t=i*T_step;
      x=t/Te;
```

```
      if (x<=1) amp=3.119*x-3.457*x*x+1.338*x*x*x;
      else if(x>1 && t<=T_pulse) amp=1;
      else amp=0;

      if(t<T_jump) ph=0;
      else if(t>=T_jump && t<=T_process) ph=(t-
T_jump)*phi_jump/(T_process-T_jump);
      else if(t>T_process && t<=T_pulse)   ph=(exp((t-
T_process)*af/(T_pulse-T_process))-1)*(180-phi_jump)/(exp(af)-
1)+phi_jump;
      else ph=180;
      kAmp[i]=amp*amp;
      mPhase[i]=ph;
   }

}

float Input::improvePhase(float input)
{
   static float
correction[19]={0,8.2,15.2,23,32.2,42.8,55,71.2,90.2,111.2,132.2,150.2
,165.8,179.5,189.8,199.5,206.8,214,219.8};
   int index;
   index=(int)(input/10.0);
   if(index>17) index=17;
   return ((correction[index]+(correction[index+1]-
correction[index])/10.0*(input-index*10.0)));
}

void Input::improvePhase(float *inputPhase)
{
   for(int i=0;i<high;i++)
   {
      inputPhase[i]=improvePhase(inputPhase[i]);
   }
}
float   Input::amplitude(float t)
{
   int index=(int)((t-T_start)/T_step);
   if (t<T_start)
   {
      cout << "t is less than T_start" << endl;
      exit(1);
   }
   return sqrt(kAmp[index]);
}
float Input::phase(float t)
{
//   int index=(int)((t-T_start+700)/T_step);
   //if(t<3300) t=3300;
   //if(t>6780) t=6780;
   int index=(int)((t-T_start)/T_step);
   if (t<T_start)
   {
      cout << "t is less than T_start" << endl;
      exit(1);
   }
   //return (-Phase[index]*M_PI/180);
   //return (corPhase[index]/16.6*M_PI/180);
   //return (corPhase[index]*M_PI/180);
```

```
//   if (t>corStart && t<corFinal )return ((mPhase[index]-
correctPhase(t))*M_PI/180);
//   else return (mPhase[index]*M_PI/180);
   return (mPhase[index]*M_PI/180);
}
float   Input::outAmp(float t)
{
   int index=(int)((t-T_start)/T_step);
   if (t<T_start)
   {
      cout << "t is less than T_start" << endl;
      exit(1);
   }
   //return sqrt(cAmp[index]);
   return (cAmp[index]);
}
float Input::outPhase(float t)
{
   int index=(int)((t-T_start)/T_step);
   if (t<T_start)
   {
      cout << "t is less than T_start" << endl;
      exit(1);
   }
   return cPhase[index];
}
float Input::correctPhase(float t)
{
   int index=(int)((t-corStart)/T_step);
   if (t<T_start || t>corFinal)
   {
      cout << "t is less than T_start" << endl;
      exit(1);
   }
   return (corPhase2[index]);
}
class Process
{
   Input *in;
   float T_jump,T_process,T_endproc,phi_jump,t;
   float *newPhase,*cAmp,*cPhase;
   float Q0,w,tau,beta,alfa,delW;
        float_complex vcp,vc,vcn;
          float_complex v3_begin,v3_end;
   int memoryStatus;

   void allocMemory();
   float_complex equation(float t);
   float_complex equation2(float t);
   float_complex equation2(float t,float ph);
        float norm(float_complex com)
        {
     return(real(com)*real(com)+imag(com)*imag(com));
   }
        float_complex polar(float am,float te)
        {
     return(am*float_complex(cos(te),sin(te)));
   }
   public:
   Process(Input *in,float T_jump,float T_process,float
T_endproc,float phi_jump,float Q0,float beta,float w,float delW);
```

```
   ~Process()
   {
      delete newPhase;
      delete cAmp;
      delete cPhase;
   }
   float   phase(float t);
   void runCheck();
   void runProcess();
   float outAmp(float t)
   {
      int index=(int)((t-in->show_T_start())/in->show_T_step());
      if (t<in->show_T_start())
      {
         cout << "t is less than T_start" << endl;
         exit(1);
      }
      return (cAmp[index]);
   }
   float outPhase()
   {
      int index=(int)((t-in->show_T_start())/in->show_T_step());
      if (t<in->show_T_start())
      {
         cout << "t is less than T_start" << endl;
         exit(1);
      }
      return (cPhase[index]);
   }
   float deltaW();
};
Process::Process(Input *in,float T_jump,float T_process,float
T_endproc,float phi_jump,float Q0,float beta,float w,float delW)
{
   this->in=in;
   this->T_jump=T_jump;
   this->T_process=T_process;
   this->T_endproc=T_endproc;
   this->phi_jump=phi_jump;
   this->Q0=Q0;
   this->beta=beta;
   this->w=w;
   this->delW=delW;
   memoryStatus=OFF;
   allocMemory();
   tau=2*Q0/(1+beta)/w;
   alfa=2*beta/(1+beta);
}
void Process::allocMemory()
{
   int phasesize = (int)((T_endproc - T_process)/in->show_T_step()  +
1.0);
   if (!(newPhase =new float[phasesize]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
   if (!(cAmp =new float[in->show_high()]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
```

```
   }
   if (!(cPhase =new float[in->show_high()]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
}
void Process::runCheck()
{
   float t;
   float_complex vr;

   t=in->show_T_start();
   vcp=vc=float_complex(0,0);
   cAmp[0]=cPhase[0]=0;
   t+=in->show_T_step();
   vc=vcp+in->show_T_step()/tau*(alfa*polar(in->amplitude(t-in-
>show_T_step()),in->phase(t-in->show_T_step()))-
vcp*(float_complex(1,0)+float_complex(0,1)*tau*delW));
   vr=vc-polar(in->amplitude(t),in->phase(t));
   cAmp[1]=norm(vr);
   cPhase[1]=arg(vr);
   t+=in->show_T_step();
   for(int i=2;t<=in->show_T_pulse()+in->show_T_start();t+=in-
>show_T_step(),i++)
   {
      vcn=equation(t);
      vcp=vc;
      vc=vcn;
      vr=vc-polar(in->amplitude(t),in->phase(t));
      cAmp[i]=norm(vr);
      cPhase[i]=arg(vr);
   }
}
void Process::runProcess()
{
   float nor1,nor2,normBase,t;
   float maxPh,minPh,accuracy;
   float_complex vr;

   t=in->show_T_start();
   vcp=vc=float_complex(0,0);
   cAmp[0]=cPhase[0]=0;
   t+=in->show_T_step();
   vc=vcp+in->show_T_step()/tau*(alfa*polar(in->amplitude(t-in-
>show_T_step()),in->phase(t-in->show_T_step()))-
vcp*(float_complex(1,0)+float_complex(0,1)*tau*delW));
   vr=vc-polar(in->amplitude(t),phase(t));
   cAmp[1]=norm(vr);
   cPhase[1]=arg(vr);
   t+=in->show_T_step();
   int i;
   for(i=2;t<=T_process;t+=in->show_T_step(),i++)
   {
      vcn=equation2(t);
      vcp=vc;
      vc=vcn;
      vr=vc-polar(in->amplitude(t),phase(t));
      cAmp[i]=norm(vr);
      cPhase[i]=arg(vr);
   }
```

```
   v3_begin=vr;
   normBase=norm(vc-polar(in->amplitude(t-in->show_T_step()),phase(t-
in->show_T_step())));
   for(;t<=T_endproc;t+=in->show_T_step(),i++)
   {
      //minPh=phase(t-in->show_T_step());
      minPh=phi_jump*M_PI/180;
      maxPh=M_PI;
      accuracy=(maxPh-minPh)/(T_endproc-T_process)*in-
>show_T_step()/20;
                nor1=norm(equation2(t,minPh)-polar(in-
>amplitude(t),minPh));
                nor2=norm(equation2(t,maxPh)-polar(in-
>amplitude(t),maxPh));
      if((nor1-normBase)*(nor2-normBase)>0)
         if((nor1-normBase)>0)      maxPh=minPh;
         else   minPh=maxPh;
      while((maxPh-minPh)>accuracy)
      {
         maxPh=(maxPh+minPh)/2;
                        nor1=norm(equation2(t,minPh)-polar(in-
>amplitude(t),minPh));
                        nor2=norm(equation2(t,maxPh)-polar(in-
>amplitude(t),maxPh));
         if((nor1-normBase)*(nor2-normBase)>0)
         {
            maxPh=2*maxPh-minPh;
            minPh=(maxPh+minPh)/2;
         }
      }
      int index=(int)((t-T_process)/in->show_T_step());
      newPhase[index]=(minPh+maxPh)/2.0;
      vcn=equation2(t);
      vcp=vc;
      vc=vcn;
      vr=vc-polar(in->amplitude(t),phase(t));
      cAmp[i]=norm(vr);
      cPhase[i]=arg(vr);
   }
   v3_end=vr;
   for(;t<in->show_T_pulse()+in->show_T_start();t+=in-
>show_T_step(),i++)
   {
      vcn=equation2(t);
      vcp=vc;
      vc=vcn;
      vr=vc-polar(in->amplitude(t),phase(t));
      cAmp[i]=norm(vr);
      cPhase[i]=arg(vr);
   }
}
float   Process::phase(float t)
{
   float ph;
   if(t<=in->show_T_start()) ph=(T_jump-in->show_T_start())*delW;
   else if(t>in->show_T_start() && t<T_jump) ph=(T_jump-t)*delW;
//   else if(t>=T_jump && t<=T_process) ph=(t-
T_jump)*phi_jump/(T_process-T_jump)*M_PI/180;
   else if(t>=T_jump && t<=T_process)
ph=phi_jump*M_PI/180*sin(M_PI/2.0/(T_process-T_jump)*(t-T_jump));
   else if(t>T_process && t<=T_endproc)
```

```
      ph=newPhase[(int)((t-T_process)/in->show_T_step())];
   else ph=newPhase[(int)((T_endproc-T_process)/in->show_T_step())];
//   else ph=(t-T_jump)*delW;

   return(ph);
}
float_complex Process::equation(float t)
{
   return (equation2(t,in->phase(t-in->show_T_step())));
}
float_complex Process::equation2(float t)
{
   return (equation2(t,phase(t-in->show_T_step())));
}
float_complex Process::equation2(float t,float ph)
{
   float_complex a,b,c,a2,b2,c2,result,vg,vgp,vgn;
   float w0=w-delW,h=in->show_T_step();
   float_complex ir=float_complex(1,0);
   float_complex ii=float_complex(0,1);

   a=ir+ii*(float)((w*w-w0*w0)*tau/(2.0*w));
   b=ir*tau-ii/w;
   c=-ii*(float)(tau/(2.0*w));
   a2=a-c*(float)(2.0/(h*h));
   b2=-b/(float)(2.0*h)+c/(float)(h*h);
   c2=b/(float)(2.0*h)+c/(float)(h*h);
   vgp=polar(in->amplitude(t-h),ph);
   vg=polar(in->amplitude(t),ph);
   vgn=polar(in->amplitude(t+h),ph);
   result=-vc*a2-vcp*b2+alfa*vg-ii*(vgn-vgp)*(float)(alfa/w/(2.0*h));
   result/=c2;
   return (result);
//   return(vcp+2*in->show_T_step()/tau*(alfa*polar(in->amplitude(t-
in->show_T_step()),ph)-
vc*(float_complex(1,0)+float_complex(0,1)*tau*delW)));
}
float Process::deltaW()
{
   return (arg(v3_end)-arg(v3_begin))/(in->show_T_pulse()+in-
>show_T_start()-T_process);
}
class Output
{
   float *outT,*outAmp1,*outAmp2,*outWFG;
   Input *in;
   Process *proc;
   public:
   Output(Input *in,Process *proc)
   {
      this->in=in;
      this->proc=proc;
      if (!(outT =new float[in->show_high()]))
      {
         printf("Not enough memory to allocate buffer\n");
         exit(1);  /* terminate program if out of memory */
      }
      if (!(outAmp1 =new float[in->show_high()]))
      {
         printf("Not enough memory to allocate buffer\n");
         exit(1);  /* terminate program if out of memory */
```

```
      }
      if (!(outAmp2 =new float[in->show_high()]))
      {
         printf("Not enough memory to allocate buffer\n");
         exit(1);  /* terminate program if out of memory */
      }
      if (!(outWFG =new float[in->show_high()*2+2]))
      {
         printf("Not enough memory to allocate buffer\n");
         exit(1);  /* terminate program if out of memory */
      }
      read();
   }
   float convertPhase(float ph);
   void read()
   {
      int i=0;
      for(i=0;i<in->show_high();i++)
      {
         float t=in->show_T_start()+i*in->show_T_step();
         outT[i]=t;
         outAmp1[i]=in->outAmp(t);
         outAmp2[i]=proc->outAmp(t)*0.45;
         //outAmp2[i]=in->phase(t)*180.0/M_PI+35;
         if(ProcStat)
         {
            outWFG[i*2]=outT[i];
            //if(t>in->corStart && t<in->corFinal)
outWFG[i*2+1]=convertPhase(proc->phase(t)*180.0/M_PI+in-
>correctPhase(t));
            //else outWFG[i*2+1]=convertPhase(proc-
>phase(t)*180.0/M_PI);
            if(!signStat)   outWFG[i*2+1]=convertPhase(proc-
>phase(t)*180.0/M_PI);
            else      outWFG[i*2+1]=convertPhase(200-(proc-
>phase(t)*180.0/M_PI));
         }
      }
      if(ProcStat)
      {
         outWFG[i*2]=outT[i-1]+600;
         outWFG[i*2+1]=outWFG[1];
      }
   }
   void show();
   void write();
};
void Output::show()
{
//   if (cpgopen("?") < 1)
   if (cpgopen("/XWINDOW") < 1)
   {
      cout << "Error in chart" << endl;
      exit(1);
   }
   cpgenv(in->show_T_start(), in->show_T_pulse()+in->show_T_start(),
0, 30, 0, 0);
   //cpgpt(in->show_high(), (const float *)outT, (const float
*)outAmp,9);
   cpgsci(2);
```

```
   cpgline(in->show_high(), (const float *)outT, (const float
*)outAmp1);
   cpgsci(7);
   cpgline(in->show_high(), (const float *)outT, (const float
*)outAmp2);
   FILE *fp;
   fp=fopen("newout.txt","w");
   for(int i=0;i<in->show_high();i++)
   {
//      if(!ProcStat)      fprintf(fp,"%f   %f   %f   %f   %f   %f
%f\n",outT[i],in->amplitude(outT[i]),in->phase(outT[i]),proc-
>phase(outT[i]),outWFG[i*2+1],outAmp1[i],outAmp2[i]);
//      fprintf(fp,"%f  %f\n",outT[i],in->correctPhase(outT[i]));
      if(!ProcStat)      fprintf(fp,"%f  %f\n",outT[i],in-
>phase(outT[i]));
      else fprintf(fp,"%f   %f   %f   %f   %f   %f\n",outT[i],in-
>amplitude(outT[i]),in->phase(outT[i]),proc-
>phase(outT[i]),outAmp1[i],outAmp2[i]);
   }
   fclose(fp);
}
float Output::convertPhase(float ph)
{
   static float
correction[19]={0,8.2,15.2,23,32.2,42.8,55,71.2,90.2,111.2,132.2,150.2
,165.8,179.5,189.8,199.5,206.8,214,219.8};
   float result;

   if(ph<0)
   {
      cout << "phase can not be less tan zero" << endl;
      exit(0);
   }
   int i=0;
   //if (ph>=correction[20]) return(180);
   for(i=0;ph>=correction[i];i++);
   i--;
   result=i*10+10/(correction[i+1]-correction[i])*(ph-correction[i]);
   return result;
}
void Output::write()
{
   int res;
   int plsOption = 0;
   char eqp_name[16] = "CK.WFGMKS";
   Equipment eqp;
   strcat(eqp_name,kNumber);

   eqp = EqpCreateByName (eqp_name);
   if (eqp == NULL)
   {
      printf("Error!\n");
      exit(1);
   }
   res = EqpSetProperty (eqp, EQP_CCV, EQP_TYPE_FLOAT, in-
>show_high()*2+2);
   for(int i=0;i<=in->show_high();i++)   cout << outWFG[i*2] << "   "
<< outWFG[i*2+1] << endl;
     res = EqpWrite (eqp, EQP_CCV, plsOption, outWFG, in-
>show_high()*2+2, NULL, 0);
   cout << "It's work" << endl;
```

```
   if (res != 0)
   {
      printf("Set CCV property failed for %s.\n", eqp_name);
   }
}
int  main(int argc,char *argv[])
{
   float beta,f0=2.99855,Q0,delw,phi,startT,finalT;
   float
dbeta=5,dQ0=187000,ddelw=133,dphi=110,dstartT=1500,dfinalT=7500;
   float sJump,dsJump=6000,fJump,dfJump=6060,fFlat,dfFlat=6800;
   char str[80],dkNumber[80]="13";
   //int WriteStat=OFF;

   //externalFile=argv[1];
   printf("Enter klystron number[%s]:",dkNumber);
   gets(str);
   if(!strcmp(str,"")) strcpy(kNumber,dkNumber);
   else strcpy(kNumber,str);

   printf("Enter Beta[%f]:",dbeta);
   gets(str);
   if(!strcmp(str,"")) beta=dbeta;
   else beta=atof(str);

   printf("Enter deltaW[%f]:",ddelw);
   gets(str);
   if(!strcmp(str,"")) delw=ddelw;
   else delw=atof(str);

   printf("Enter Q0[%f]:",dQ0);
   gets(str);
   if(!strcmp(str,"")) Q0=dQ0;
   else Q0=atof(str);

   printf("Enter ProcStatus[OFF]:");
   gets(str);
   if(!strcmp(str,"ON")) ProcStat=ON;

   printf("Enter startT[%f]:",dstartT);
   gets(str);
   if(!strcmp(str,"")) startT=dstartT;
   else startT=atof(str);

   printf("Enter finalT[%f]:",dfinalT);
   gets(str);
   if(!strcmp(str,"")) finalT=dfinalT;
   else finalT=atof(str);

   if(ProcStat)
   {
      printf("Enter phaseJump[%f]:",dphi);
      gets(str);
      if(!strcmp(str,"")) phi=dphi;
      else phi=atof(str);

      printf("Enter startJump[%f]:",dsJump);
      gets(str);
      if(!strcmp(str,"")) sJump=dsJump;
      else sJump=atof(str);
```

```
      printf("Enter finishJump[%f]:",dfJump);
      gets(str);
      if(!strcmp(str,"")) fJump=dfJump;
      else fJump=atof(str);

      printf("Enter FinishFlatting[%f]:",dfFlat);
      gets(str);
      if(!strcmp(str,"")) fFlat=dfFlat;
      else fFlat=atof(str);

      printf("Enter SignStatus[+]:");
      gets(str);
      if(!strcmp(str,"-")) signStat=ON;

   }
   else
   {
      phi=dphi;
   }
   Input inp(finalT-startT,startT,20);
//   Process* proc = new
Process(&inp,5800,5820,6800,phi,Q0,beta,2*M_PI*f0,delw*2*M_PI*1e-6);
   Process* proc = new
Process(&inp,sJump,fJump,fFlat,phi,Q0,beta,2*M_PI*f0,delw*2*M_PI*1e-
6);

   if(!ProcStat)   proc->runCheck();
   else      proc->runProcess();
   Output out(&inp,proc);
   if(ProcStat)   out.write();
   out.show();
   cout << proc->deltaW()<< endl;
   delete proc;
   cpgclos();
   return 0;
}
```

7. APPENDIX II: PROGRAM TO REMOVE THE RIPPLING OF PULSE COMPRESSOR OUTPUT

```
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <eqp/eqp.h>
#include <eqp/eqp_def.h>

double improvePhase(double input)
{
   static double
correction[19]={0,8.2,15.2,23,32.2,42.8,55,71.2,90.2,111.2,132.2,150.2
,165.8,179.5,189.8,199.5,206.8,214,219.8};
   int index;
   index=(int)(input/10.0);
   if(index>17) index=17;
   return ((correction[index]+(correction[index+1]-
correction[index])/10.0*(input-index*10.0)));
}
double convertPhase(double ph)
{
   static double
correction[19]={0,8.2,15.2,23,32.2,42.8,55,71.2,90.2,111.2,132.2,150.2
,165.8,179.5,189.8,199.5,206.8,214,219.8};
   double result;

   if(ph<0)
   {
      cout << "phase can not be less tan zero" << endl;
      exit(0);
   }
   int i=0;
   for(i=0;ph>=correction[i];i++);
   i--;
   result=i*10+10/(correction[i+1]-correction[i])*(ph-correction[i]);
   return result;
}
```

```
main(int argc,char *argv[])
{

   char str[80],dkNumber[80]="13",kNumber[80];

   printf("Enter klystron number[%s]:",dkNumber);
   gets(str);
   if(!strcmp(str,"")) strcpy(kNumber,dkNumber);
   else strcpy(kNumber,str);

   //char smp_name[16]="CK.SVPEI1330P-C";
   char smp_name[16]="CK.SVPKI";
   strcat(smp_name,kNumber);
   strcat(smp_name,"P-C");
   char eqp_name[16]="CK.WFGMKS";
   strcat(eqp_name,kNumber);
   char eqp_name2[16]="CK.COMPMKS";
   strcat(eqp_name2,kNumber);
   char oper[5];
   int res,plsOption = 0,high,high2;
   double interv,dinterv=20,start,angle;
   double startT,dstartT=2200,base,dbase=90;
   double finalT,dfinalT=6800;
   double *corPhase,*out;
   Equipment eqp,eqp2,smp;

   eqp = EqpCreateByName (eqp_name);
   if (eqp == NULL)
   {
      printf("Error!\n");
      exit(1);
   }

   eqp2 = EqpCreateByName (eqp_name2);
   if (eqp2 == NULL)
   {
      printf("Error!\n");
      exit(1);
   }

   smp = EqpCreateByName (smp_name);
   if (smp == NULL)
   {
      printf("Error!\n"); exit(1);
   }

   printf("Enter interval[%f]:",dinterv);
   gets(str);
   if(!strcmp(str,"")) interv=dinterv;
   else interv=atof(str);

   printf("Enter start time[%f]:",dstartT);
   gets(str);
   if(!strcmp(str,"")) startT=dstartT;
   else startT=atof(str);

   printf("Enter final time[%f]:",dfinalT);
   gets(str);
   if(!strcmp(str,"")) finalT=dfinalT;
   else finalT=atof(str);
```

```
   printf("Enter base angle[%f]:",dbase);
   gets(str);
   if(!strcmp(str,"")) base=dbase;
   else base=atof(str);

   printf("Enter operation(Flat,Correctionfile,0):");
   gets(oper);

   high2=(finalT-startT)/interv;
   res = EqpWrite (smp, EQP_INTERV, plsOption, &interv, 1, NULL, 0);
   res = EqpRead (smp, EQP_DELAY,   plsOption, &start,  1, NULL, 0);
   res = EqpWrite (smp, EQP_STRT,   plsOption, &startT,  1, NULL, 0);
   res = EqpSetProperty (smp, EQP_HIGH, EQP_TYPE_INT, 1);
   res = EqpRead (smp, EQP_HIGH,   plsOption, &high,   1, NULL, 0);
   res = EqpRead (eqp2, EQP_CCV, plsOption, &angle, 1, NULL, 0);

   cout << high <<"  "<<high2 << endl;
   if(startT<start || high2>high)
   {
      cout << "Error in not adaption between TS and T_pulse with
devices" << endl;
      exit(1);
   }
   high=high2;
   if (!(corPhase =new double[high]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
   if (!(out =new double[high*2]))
   {
      printf("Not enough memory to allocate buffer\n");
      exit(1);  /* terminate program if out of memory */
   }
   res = EqpSetProperty (smp, EQP_AQN, EQP_TYPE_DOUBLE, high);
   res = EqpRead (smp, EQP_AQN, plsOption, corPhase, high, NULL, 0);
   FILE *fp;
   if(!strcmp(oper,"c"))
   {
      fp=fopen(argv[1],"w");
      if(fp==NULL)
      {
         printf("Error opening file\n");
         exit(1);  /* terminate program if out of memory */
      }
      fprintf(fp,"%f  %f  %f\n",startT,finalT,interv);
   }
   for(int i=0;i<high;i++)
   {
      out[i*2]=startT+i*interv;
      if(!strcmp(oper,"0"))   out[i*2+1]=base;
      else if(!strcmp(oper,"c"))
      {
         out[i*2+1]=corPhase[i];
      }
      else
      {
         out[i*2+1]=convertPhase(corPhase[i]*10+improvePhase(base));
      }
      cout << out[i*2] << "   " << out[i*2+1] << endl;
```

```
      if(!strcmp(oper,"c")) fprintf(fp,"%f
%f\n",out[i*2],out[i*2+1]);
   }
   if(strcmp(oper,"c"))
   {
      res = EqpSetProperty (eqp, EQP_CCV, EQP_TYPE_DOUBLE, high*2);
        res = EqpWrite (eqp, EQP_CCV, plsOption, out, high*2, NULL,
0);
      if (res != 0)
      {
         printf("Set CCV property failed for %s.\n", eqp_name);
      }
   }
   if(!strcmp(oper,"c"))
   {
      fclose(fp);
   }
   delete corPhase;
   delete out;
}
```

8. APPENDIX III: PLACET MODEL CODE (RUN.TCL)

```
# settings

set input_file  "coord_entrance_AS0530_sb.prn"
set output_file "output.dump"
set phase_offset 4.415


set script_dir .

set match(charge) 2.04375e+10
set match(sigma_z) 1650


 # here we switched off the wakefields
# source $script_dir/ctf3_beam-wakes0.tcl
# source $script_dir/ctf3_basic_single-wakes0.tcl

 # here we have the wakefields
 source $script_dir/ctf3_beam.tcl
 source $script_dir/ctf3_basic_single.tcl


set e_initial 0.020

set e0 $e_initial

Girder
TclCall -script {
  BeamDump -file before.dump
}
source $script_dir/linac.tcl
TclCall -script {
  global output_file
  BeamDump -file $output_file
  Octave {
      B=load '$output_file';
      sigZ=std(B(:,4));
      disp(sigZ);
      B=load 'before_chicane.dump';
```

```
      sigZ=std(B(:,4));
      disp(sigZ);
      }

}

set beamline ctf3beamline
BeamlineSet -name $beamline


set n_total [lindex [exec wc -l $input_file] 0]
set n_slice [expr $n_total / 8]
set n 8

calc beam.dat $match(charge) -3 3 $match(sigma_z) $n_slice
InjectorBeam "beam0" -bunches 1 \
          -macroparticles [expr $n] \
          -particles $n_total \
          -energyspread 0.0 \
          -ecut 3.0 \
          -e0 $e_initial \
          -file beam.dat \
          -chargelist {1.0} \
          -charge 1.0 \
          -phase 0.0 \
          -overlapp [expr -390*0.3/1.3] \
          -distance [expr 0.3/1.3] \
          -alpha_x 1.0\
          -alpha_y 1.0\
          -beta_x 1.0\
          -beta_y 1.0\
          -emitt_x 1.0\
          -emitt_y 1.0

set l {}
lappend l {1.0 0.0 0.0}
SetRfGradientSingle "beam0" 0 $l

set fp [open $input_file r]
set data [read $fp]
close $fp


set sum_z 0.0
set n 0.0
set sum_z2 0.0

set data [split $data "\n"]
set fp [open "particles.tmp" w]
foreach line $data {
  set line [concat $line]
  if { [llength $line] >= 6 } {
    set freq    2.998e9
    set clight  2.9979246e+08
    set lambda  [expr $clight / $freq]
    set x  [expr [lindex $line 0] * 1e4]
    set xp [expr [lindex $line 1] * 1e3]
    set y  [expr [lindex $line 2] * 1e4]
    set yp [expr [lindex $line 3] * 1e3]
    set z  [expr ([lindex $line 4] + $phase_offset) * $lambda / 360.0
* 1e6]
```

```
    set e  [expr [lindex $line 5] * 1e-3]
    set sum_z  [expr $sum_z + $z]
    set sum_z2 [expr $sum_z2 + $z*$z]
    set n [expr $n + 1.0]

    puts $fp "$e $x $y $z $xp $yp"
  }
}
close $fp
exec sort -b -g -k 4,4 particles.tmp > particles.in

BeamRead -file particles.in -beam "beam0"

set z_stdev [expr sqrt($sum_z2/$n - ($sum_z/$n)*($sum_z/$n))]

puts "tracking... $z_stdev $sum_z $sum_z2 $n "

FirstOrder 1

TestNoCorrection -beam beam0 -emitt_file dummy -machines 1 -survey
Zero -bpm_res 0.0
```

9. APPENDIX IV: PLACET MODEL CODE (LINAC.TCL)

```
set sbend_synrad 0
set quad_synrad 0
set mult_synrad 0
set phase_15 0
set sbend_csr 0
set ccc 1.0

set em0 0.511e-3
set cc 1

set e1 0.120

set l_cell 0.03332
set l_cplr 0.026

set E4  20.0
set E6  39
set E7  56
set E8  71
set E12 85
set E13 98
set E14 111
set E15 122

set EGAIN 15.943;

set deltaE5  [expr $E6 - $E4]
set deltaE6  [expr $E7 - $E6]
set deltaE7  [expr $E8 - $E7]
set deltaE11 [expr $E12 - $E8]
set deltaE12 [expr $E13 - $E12]
set deltaE13 [expr $E14 - $E13]
set deltaE15 [expr $E15 - $E14]

set af5 [expr $deltaE5/$EGAIN]
set af6 [expr $deltaE6/$EGAIN]
set af7 [expr $deltaE7/$EGAIN]
set af11 [expr $deltaE11/$EGAIN]
set af12 [expr $deltaE12/$EGAIN]
```

```
set af13 [expr $deltaE13/$EGAIN]
set af15 [expr $deltaE15/$EGAIN]

set CLIGHT 2.99792458e+8
set ECQA 0.25
set FQA  [expr $CLIGHT*1E-6*$ECQA]
set ECQB 0.25
set FQB  [expr $CLIGHT*1E-6*$ECQB]
set ECQC 0.056
set FQC  [expr $CLIGHT*1E-6*$ECQC]
set ECQD 0.056
set FQD  [expr $CLIGHT*1E-6*$ECQD]
set ECQE 0.03095
set FQE  [expr $CLIGHT*1E-6*$ECQE]
set ECQF 0.016
set FQF  [expr $CLIGHT*1E-6*$ECQF]
set ECQG [expr 2.4/0.300/200]
set FQG  [expr $CLIGHT*1E-6*$ECQG]
set ECQH [expr 2.049/0.380/150]
set FQH  [expr $CLIGHT*1E-6*$ECQH]
set ECQI [expr 0.55/0.359/55]
set FQI  [expr $CLIGHT*1E-6*$ECQI]
set ECQJ [expr 3.2/0.400/450]
set FQJ  [expr $CLIGHT*1E-6*$ECQJ]

set iqda0605 2.258046366
set iqfb0610 2.416338141
set iqdb0705 1.947326127
set iqfc0710 16.44704893
set iqdb0805 2.453281961
set iqfc0810 20.7770903
set iqdb0815 2.464599055
set iqdb0905 2.510580409
set iqfc0910 21.16584923

set iqdb1005 2.515722787
set iqfc1010 21.19870263
set iqdb1015 2.514850087
set iqdb1105 2.500133919
set iqfc1110 21.08192506
set iqdc1205 13.33009136
set iqfd1210 25.18510055
set iqdd1305 15.42328149
set iqfd1310 29.12981856
set iqdd1405 17.52960648
set iqfd1410 33.09815472
set iqdd1505 17.53501239
set iqfd1510 33.10787947

set iqdd0110 7.97
set iqfd0130 3.56
set iqfd0150 7.97
set iqdd0220 0
set iqfe0230 0
set iqdf0245 0
set iqfe0250 0
set iqfd0310 7.58
set iqfd0330 3.38
set iqdd0350 7.58
```

```
set LQS 0.124
set LQL 0.22


puts $af5
puts $af6
puts $af7
puts $af11
puts $af12
puts $af13
puts $af15


set file [open "nominal_voltage.dat"]
set data [read $file]
close $file
set voltage [split $data \n]


### here begins the lattice

Drift -name "DCAV" -length 0.05088

### Cavity 5-1
set factor $af5
AccCavity -name "CAV5$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV5$i" -length $l_cell -type 0 -phase 0 -
gradient [expr $factor * [lindex $voltage $i] / $l_cell * 1e-3]
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV5$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
puts "energy : $e0"
### End of Cavity 5-1

Drift -name "DCAV" -length 0.05088
Drift -name "D0560_1" -length 0.06
Drift -name "D0560_2" -length 0.02
Drift -name "DCAV" -length 0.05088

### Cavity 5-2
set factor $af5
AccCavity -name "CAV5$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV5$i" -length $l_cell -type 0 -phase 0 -
gradient [expr $factor * [lindex $voltage $i] / $l_cell * 1e-3]
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV5$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
puts "energy : $e0"
### End of Cavity 5-2
```

```
Drift -name "DCAV" -length 0.05088
Drift -name "D0590" -length 0.13
Bpm -name "CL.BPM0590" -length 0.2
Drift -name "D0605" -length 0.148
set K0605 [expr -$FQA*$iqda0605*$LQS*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K0605 [expr $K0605*$correction]
puts [expr $K0605/$e0]
Quadrupole -name "CL.QDA0605" -synrad $quad_synrad -length $LQS -
strength $K0605
Drift -name "D0610" -length 0.298
set K0610 [expr $FQB*$iqfb0610*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K0610 [expr $K0610*$correction]
puts [expr $K0610/$e0]
Quadrupole -name "CL.QFB0610" -synrad $quad_synrad -length $LQL -
strength $K0610
Drift -name "D0615" -length 0.298
set K0615 [expr -$FQA*$iqda0605*$LQS*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K0615 [expr $K0615*$correction]
puts [expr $K0615/$e0]
Quadrupole -name "CL.QDA0615" -synrad $quad_synrad -length $LQS -
strength $K0615
Drift -name "D0620" -length 0.349
# HCORRECTOR -name "CL.DHC0620" -length 0
# VCORRECTOR -name "CL.DVC0620" -length 0
Drift -name "D0630" -length 0.079
Drift -name "DCAV" -length 0.05088

### Cavity 6-1
set factor $af6
AccCavity -name "CAV6$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV6$i" -length $l_cell -type 0 -phase 0 -
gradient [expr $factor * [lindex $voltage $i] / $l_cell * 1e-3]
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV6$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
puts "energy : $e0"
### End of Cavity 6-1

Drift -name "DCAV" -length 0.05088
Drift -name "D0660_1" -length 0.07
Drift -name "D0660_2" -length 0.01
Drift -name "DCAV" -length 0.05088

### Cavity 6-2
set factor $af6
AccCavity -name "CAV6$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV6$i" -length $l_cell -type 0 -phase 0 -
gradient [expr $factor * [lindex $voltage $i] / $l_cell * 1e-3]
```

```
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV6$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
puts "energy : $e0"
### End of Cavity 6-2

Drift -name "DCAV" -length 0.05088
Drift -name "D0690" -length 0.14
Bpm -name "CL.BPM0690" -length 0.2
Drift -name "D0705" -length 0.1
set K0705 [expr -$FQB*$iqdb0705*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K0705 [expr $K0705*$correction]
puts [expr $K0705/$e0]
Quadrupole -name "CL.QDB0705" -synrad $quad_synrad -length $LQL -
strength $K0705
Drift -name "D0710" -length 0.25
set K0710 [expr $FQC*$iqfc0710*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K0710 [expr $K0710*$correction]
puts [expr $K0710/$e0]
Quadrupole -name "CL.QFC0710" -synrad $quad_synrad -length $LQL -
strength $K0710
Drift -name "D0715" -length 0.25
set K0715 [expr -$FQB*$iqdb0705*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K0715 [expr $K0715*$correction]
puts [expr $K0715/$e0]
Quadrupole -name "CL.QDB0715" -synrad $quad_synrad -length $LQL -
strength $K0715
Drift -name "D0720" -length 0.301
# HCORRECTOR -name "CL.DHC0720" -length 0
# VCORRECTOR -name "CL.DVC0720" -length 0
Drift -name "D0730" -length 0.079
Drift -name "DCAV" -length 0.05088

### Cavity 7-1
set factor $af7
AccCavity -name "CAV7$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV7$i" -length $l_cell -type 0 -phase 0 -
gradient [expr $factor * [lindex $voltage $i] / $l_cell * 1e-3]
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV7$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
puts "energy : $e0"
### End of Cavity 7-1

Drift -name "DCAV" -length 0.05088
Drift -name "D0760_1" -length 0.07
Drift -name "D0760_2" -length 0.01
Drift -name "DCAV" -length 0.05088
```

```
### Cavity 7-2
set factor $af7
AccCavity -name "CAV7$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV7$i" -length $l_cell -type 0 -phase 0 -
gradient [expr $factor * [lindex $voltage $i] / $l_cell * 1e-3]
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV7$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
puts "energy : $e0"
### End of Cavity 7-2

Drift -name "DCAV" -length 0.05088
Drift -name "D0790" -length 0.14
Bpm -name "CL.BPM0790" -length 0.2
Drift -name "D0805" -length 0.1
set K0805 [expr -$FQB*$iqdb0805*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K0805 [expr $K0805*$correction]
puts [expr $K0805/$e0]
Quadrupole -name "CL.QDB0805" -synrad $quad_synrad -length $LQL -
strength $K0805
Drift -name "D0810" -length 0.25
set K0810 [expr $FQC*$iqfc0810*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K0810 [expr $K0810*$correction]
puts [expr $K0810/$e0]
Quadrupole -name "CL.QFC0810" -synrad $quad_synrad -length $LQL -
strength $K0810
Drift -name "D0815" -length 0.25
set K0815 [expr -$FQB*$iqdb0815*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K0815 [expr $K0815*$correction]
puts [expr $K0815/$e0]
Quadrupole -name "CL.QDB0815" -synrad $quad_synrad -length $LQL -
strength $K0815
Drift -name "D0820" -length 0.065
# HCORRECTOR -name "CL.DHC0820" -length 0
# VCORRECTOR -name "CL.DVC0820" -length 0
Drift -name "D0823" -length 0.1509
Bpm -name "CL.WCM0823" -length 0
Drift -name "D0830" -length 0.1841
Drift -name "CL.DRI0830" -length 1.22
Drift -name "D0860_1" -length 0.05
Drift -name "D0860_2" -length 0.05
Drift -name "CL.DRI0860" -length 1.22
Drift -name "D0890" -length 0.1
Bpm -name "CL.BPM0890" -length 0.2
Drift -name "D0905" -length 0.1
set K0905 [expr -$FQB*$iqdb0905*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K0905 [expr $K0905*$correction]
puts [expr $K0905/$e0]
Quadrupole -name "CL.QDB0905" -synrad $quad_synrad -length $LQL -
strength $K0905
Drift -name "D0910" -length 0.25
```

```
set K0910 [expr $FQC*$iqfc0910*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K0910 [expr $K0910*$correction]
puts [expr $K0910/$e0]
Quadrupole -name "CL.QFC0910" -synrad $quad_synrad -length $LQL -
strength $K0910
Drift -name "D0915" -length 0.25
set K0915 [expr -$FQB*$iqdb0905*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K0915 [expr $K0915*$correction]
puts [expr $K0915/$e0]
Quadrupole -name "CL.QDB0915" -synrad $quad_synrad -length $LQL -
strength $K0915
Drift -name "D0920" -length 0.113
# HCORRECTOR -name "CL.DHC0920" -length 0
# VCORRECTOR -name "CL.DVC0920" -length 0
Drift -name "D0930" -length 0.287
Drift -name "CL.DRI0930" -length 1.22
Drift -name "D0960_1" -length 0.05
Drift -name "D0960_2" -length 0.05
Drift -name "CL.DRI0960" -length 1.22
Drift -name "D0990" -length 0.1
Bpm -name "CL.BPM0990" -length 0.2
Drift -name "D1005" -length 0.1
set K1005 [expr -$FQB*$iqdb1005*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1005 [expr $K1005*$correction]
puts [expr $K1005/$e0]
Quadrupole -name "CL.QDB1005" -synrad $quad_synrad -length $LQL -
strength $K1005
#Quadrupole -name "CL.QDB1005" -synrad $quad_synrad -length $LQL -
strength [expr 2.32355729*$e0]
Drift -name "D1010" -length 0.25
set K1010 [expr $FQC*$iqfc1010*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1010 [expr $K1010*$correction]
puts [expr $K1010/$e0]
Quadrupole -name "CL.QFC1010" -synrad $quad_synrad -length $LQL -
strength $K1010
Drift -name "D1015" -length 0.25
set K1015 [expr -$FQB*$iqdb1015*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1015 [expr $K1015*$correction]
puts [expr $K1015/$e0]
Quadrupole -name "CL.QDB1015" -synrad $quad_synrad -length $LQL -
strength $K1015
#Quadrupole -name "CL.QDB1015" -synrad $quad_synrad -length $LQL -
strength [expr -0.4649436977*$e0]
Drift -name "D1020" -length 0.301
# HCORRECTOR -name "CL.DHC1020" -length 0
# VCORRECTOR -name "CL.DVC1020" -length 0
Drift -name "D1030" -length 0.689
Bpm -name "CL.MTV1030" -length 0
Drift -name "D1040" -length 0.63
Drift -name "D1060_1" -length 0.05
Drift -name "D1060_2" -length 0.05
Drift -name "CL.DRI1060" -length 1.22
Drift -name "D1090" -length 0.1
Bpm -name "CL.BPM1090" -length 0.2
Drift -name "D1105" -length 0.1
set K1105 [expr -$FQB*$iqdb1105*$LQL*1e-3]
```

```
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1105 [expr $K1105*$correction]
puts [expr $K1105/$e0]
Quadrupole -name "CL.QDB1105" -synrad $quad_synrad -length $LQL -
strength $K1105
Drift -name "D1110" -length 0.25
set K1110 [expr $FQC*$iqfc1110*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1110 [expr $K1110*$correction]
puts [expr $K1110/$e0]
Quadrupole -name "CL.QFC1110" -synrad $quad_synrad -length $LQL -
strength $K1110
Drift -name "D1115" -length 0.25
set K1115 [expr -$FQB*$iqdb1105*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1115 [expr $K1115*$correction]
puts [expr $K1115/$e0]
Quadrupole -name "CL.QDB1115" -synrad $quad_synrad -length $LQL -
strength $K1115
Drift -name "D1120" -length 0.301
# HCORRECTOR -name "CL.DHC1120" -length 0
# VCORRECTOR -name "CL.DVC1120" -length 0
Drift -name "D1130" -length 0.079
Drift -name "DCAV" -length 0.05088

### Cavity 11-1
set factor $af11
AccCavity -name "CAV11$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV11$i" -length $l_cell -type 0 -phase 0 -
gradient [expr $factor * [lindex $voltage $i] / $l_cell * 1e-3]
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV11$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
puts "energy : $e0"
### End of Cavity 11-1

Drift -name "DCAV" -length 0.05088
Drift -name "D1160_1" -length 0.07
Drift -name "D1160_2" -length 0.01
Drift -name "DCAV" -length 0.05088

### Cavity 11-2
set factor $af11
AccCavity -name "CAV11$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV11$i" -length $l_cell -type 0 -phase 0 -
gradient [expr $factor * [lindex $voltage $i] / $l_cell * 1e-3]
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV11$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
```

```
puts "energy : $e0"
### End of Cavity 11-2

Drift -name "DCAV" -length 0.05088
Drift -name "D1190" -length 0.14
Bpm -name "CL.BPM1190" -length 0.2
Drift -name "D1205" -length 0.1
set K1205 [expr -$FQC*$iqdc1205*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1205 [expr $K1205*$correction]
puts [expr $K1205/$e0]
Quadrupole -name "CL.QDC1205" -synrad $quad_synrad -length $LQL -
strength $K1205
Drift -name "D1210" -length 0.25
set K1210 [expr $FQD*$iqfd1210*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1210 [expr $K1210*$correction]
puts [expr $K1210/$e0]
Quadrupole -name "CL.QFD1210" -synrad $quad_synrad -length $LQL -
strength $K1210
Drift -name "D1215" -length 0.25
set K1215 $K1205
puts [expr $K1215/$e0]
Quadrupole -name "CL.QDC1215" -synrad $quad_synrad -length $LQL -
strength $K1215
Drift -name "D1220" -length 0.301
# HCORRECTOR -name "CL.DHC1220" -length 0
# VCORRECTOR -name "CL.DVC1220" -length 0
Drift -name "D1230" -length 0.079
Drift -name "DCAV" -length 0.05088

### Cavity 12-1
set factor $af12
AccCavity -name "CAV12$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV12$i" -length $l_cell -type 0 -phase 0 -
gradient [expr $factor * [lindex $voltage $i] / $l_cell * 1e-3]
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV12$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
puts "energy : $e0"
### End of Cavity 12-1

Drift -name "DCAV" -length 0.05088
Drift -name "D1260_1" -length 0.07
Drift -name "D1260_2" -length 0.01
Drift -name "DCAV" -length 0.05088

### Cavity 12-2
set factor $af12
AccCavity -name "CAV12$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV12$i" -length $l_cell -type 0 -phase 0 -
gradient [expr $factor * [lindex $voltage $i] / $l_cell * 1e-3]
```

```
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV12$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
puts "energy : $e0"
### End of Cavity 12-2

Drift -name "DCAV" -length 0.05088
Drift -name "D1290" -length 0.14
Bpm -name "CL.BPM1290" -length 0.2
Drift -name "D1305" -length 0.1
set K1305 [expr -$FQD*$iqdd1305*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1305 [expr $K1305*$correction]
puts [expr $K1305/$e0]
Quadrupole -name "CL.QDD1305" -synrad $quad_synrad -length $LQL -
strength $K1305
Drift -name "D1310" -length 0.25
set K1310 [expr $FQD*$iqfd1310*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1310 [expr $K1310*$correction]
puts [expr $K1310/$e0]
Quadrupole -name "CL.QFD1310" -synrad $quad_synrad -length $LQL -
strength $K1310
Drift -name "D1315" -length 0.25
set K1315 $K1305
puts [expr $K1315/$e0]
Quadrupole -name "CL.QDD1315" -synrad $quad_synrad -length $LQL -
strength $K1315
Drift -name "D1320" -length 0.301
# HCORRECTOR -name "CL.DHC1320" -length 0
# VCORRECTOR -name "CL.DVC1320" -length 0
Drift -name "D1330" -length 0.079
Drift -name "DCAV" -length 0.05088

### Cavity 13-1
set factor $af13
AccCavity -name "CAV13$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV13$i" -length $l_cell -type 0 -phase 0 -
gradient [expr $factor * [lindex $voltage $i] / $l_cell * 1e-3]
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV13$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
puts "energy : $e0"
### End of Cavity 13-1

Drift -name "DCAV" -length 0.05088
Drift -name "D1360_1" -length 0.07
Drift -name "D1360_2" -length 0.01
Drift -name "DCAV" -length 0.05088

### Cavity 13-2
set factor $af13
```

```
AccCavity -name "CAV13$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV13$i" -length $l_cell -type 0 -phase 0 -
gradient [expr $factor * [lindex $voltage $i] / $l_cell * 1e-3]
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV13$i" -length $l_cplr -type 0 -phase 0 -gradient
[expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
puts "energy : $e0"
### End of Cavity 13-2

Drift -name "DCAV" -length 0.05088
Drift -name "D1390" -length 0.14
Bpm -name "CL.BPM1390" -length 0.2
Drift -name "D1405" -length 0.1
set K1405 [expr -$FQD*$iqdd1405*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1405 [expr $K1405*$correction]
puts [expr $K1405/$e0]
Quadrupole -name "CL.QDD1405" -synrad $quad_synrad -length $LQL -
strength $K1405
Drift -name "D1410" -length 0.25
set K1410 [expr $FQD*$iqfd1410*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1410 [expr $K1410*$correction]
puts [expr $K1410/$e0]
Quadrupole -name "CL.QFD1410" -synrad $quad_synrad -length $LQL -
strength $K1410
Drift -name "D1415" -length 0.25
set K1415 $K1405
puts [expr $K1415/$e0]
Quadrupole -name "CL.QDD1415" -synrad $quad_synrad -length $LQL -
strength $K1415
Drift -name "D1420" -length 0.301
# HCORRECTOR -name "CL.DHC1420" -length 0
# VCORRECTOR -name "CL.DVC1420" -length 0
Drift -name "D1430" -length 0.079
Drift -name "CL.ACS1430" -length 1.22
Drift -name "D1460_1" -length 0.07
Drift -name "D1460_2" -length 0.01
Drift -name "CL.ACS1460" -length 1.22
Drift -name "D1490" -length 0.14
Bpm -name "CL.BPM1490" -length 0.2
Drift -name "D1505" -length 0.1
set K1505 [expr -$FQD*$iqdd1505*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1505 [expr $K1505*$correction]
puts [expr $K1505/$e0]
Quadrupole -name "CL.QDD1505" -synrad $quad_synrad -length $LQL -
strength $K1505
Drift -name "D1510" -length 0.25
set K1510 [expr $FQD*$iqfd1510*$LQL*1e-3]
set correction [expr 1.0/sqrt(1-($em0/$e0)*($em0/$e0))]
set K1510 [expr $K1510*$correction]
puts [expr $K1510/$e0]
Quadrupole -name "CL.QFD1510" -synrad $quad_synrad -length $LQL -
strength $K1510
```

```
Drift -name "D1515" -length 0.25
set K1515 $K1505
puts [expr $K1515/$e0]
Quadrupole -name "CL.QDD1515" -synrad $quad_synrad -length $LQL -
strength $K1515
Drift -name "D1520" -length 0.301
# HCORRECTOR -name "CL.DHC1520" -length 0
# VCORRECTOR -name "CL.DVC1520" -length 0
Drift -name "D1530" -length 0.079
Drift -name "DCAV" -length 0.05088

### Cavity 15-1
set factor $af15
AccCavity -name "CAV15$i" -length $l_cplr -type 0 -phase $phase_15 -
gradient [expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV15$i" -length $l_cell -type 0 -phase
$phase_15 -gradient [expr $factor * [lindex $voltage $i] / $l_cell *
1e-3]
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV15$i" -length $l_cplr -type 0 -phase $phase_15 -
gradient [expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
puts "energy : $e0"
### End of Cavity 15-1

Drift -name "DCAV" -length 0.05088
Drift -name "D1560_1" -length 0.04
Drift -name "D1560_2" -length 0.03
Drift -name "D1560_3" -length 0.01
Drift -name "DCAV" -length 0.05088

### Cavity 15-2
set factor $af15
AccCavity -name "CAV15$i" -length $l_cplr -type 0 -phase $phase_15 -
gradient [expr $factor * [lindex $voltage 0] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 0] * 1e-3]]
for {set i 1} {$i<33} {incr i} {
      AccCavity -name "CAV15$i" -length $l_cell -type 0 -phase
$phase_15 -gradient [expr $factor * [lindex $voltage $i] / $l_cell *
1e-3]
      set e0 [expr $e0 + [expr $factor * [lindex $voltage $i] * 1e-3]]
}
AccCavity -name "CAV15$i" -length $l_cplr -type 0 -phase $phase_15 -
gradient [expr $factor * [lindex $voltage 33] / $l_cplr * 1e-3]
set e0 [expr $e0 + [expr $factor * [lindex $voltage 33] * 1e-3]]
SetReferenceEnergy $e0
puts "energy : $e0"
### End of Cavity 15-2

#set e1 [expr $e0*1.015]
set e1 $e0
puts $e0
puts $e1
set nbins_val 40
set nhalffiler_val 5
set nsectors_val 20
set filterorder_val 1
```

```
Drift -name "DCAV" -length 0.05088
Drift -name "D1590" -length 0.14
Bpm -name "CL.BPM1590" -length 0.2
Drift -name "D1597" -length 0.3435
Bpm -name "CL.WCM1597" -length 0
Drift -name "DEND" -length 0.3754

TclCall -script { BeamDump -file before_chicane.dump }

Drift -name "DR0100REAL" -length 0.090233
set KC0110 [expr -$iqdd0110*$FQD*$LQL*1.0e-3*$ccc]
puts [expr $KC0110/$e0]
Quadrupole -name "QDD0110" -synrad $quad_synrad -length $LQL -strength
$KC0110
Drift -name "DR0129" -length 1.2
set KC0130 [expr $iqfd0130*$FQD*$LQL*1.0e-3*$ccc]
puts [expr $KC0130/$e0]
Quadrupole -name "QFD0130" -synrad $quad_synrad -length $LQL -strength
$KC0130
Drift -name "DR0149" -length 0.2
set KC0150 [expr $iqfd0150*$FQD*$LQL*1.0e-3*$ccc]
puts [expr $KC0150/$e0]
Quadrupole -name "QFD0150" -synrad $quad_synrad -length $LQL -strength
$KC0150
Drift -name "DR0151" -length 0.186
Bpm -name "BPM0155" -length 0
# HCORRECTOR -name "CDHE0160" -length 0
# VCORRECTOR -name "CDVE0160" -length 0
Drift -name "DR0209" -length 0.54916997
Sbend -csr $sbend_csr -csr_nbins $nbins_val -csr_nhalffiler
$nhalffiler_val -csr_filterorder $filterorder_val -csr_nsectors
$nsectors_val -csr_charge [expr $match(charge)*1.602e-19*$cc] -name
"BHC0210" -synrad $sbend_synrad -length 0.561 -angle -0.31416 -e0 $e1
-E1 -0.15708 -E2 -0.15708 -six_dim 1
Drift -name "DR0214" -length 0.30666106
Bpm -name "BPM0215" -length 0
Drift -name "DR0216" -length 0.14251
set KC0220 [expr -$iqdd0220*$FQD*$LQL*1.0e-3*$ccc]
puts [expr $KC0220/$e0]
Quadrupole -name "QDD0220" -synrad $quad_synrad -length $LQL -strength
$KC0220
Drift -name "DR0225" -length 0.27
set KC0230 [expr $iqfe0230*$FQE*0.38*1.0e-3*$ccc]
puts [expr $KC0230/$e0]
Quadrupole -name "QFE0230" -synrad $quad_synrad -length 0.38 -strength
$KC0230
Drift -name "DR0235" -length 0.629589
Sbend -csr $sbend_csr -csr_nbins $nbins_val -csr_nhalffiler
$nhalffiler_val -csr_filterorder $filterorder_val -csr_nsectors
$nsectors_val -csr_charge [expr $match(charge)*1.602e-19*$cc] -name
"BHC0240" -synrad $sbend_synrad -length 0.561 -angle 0.31416 -e0 $e1 -
E1 0.15708 -E2 0.15708 -six_dim 1
Drift -name "DR0241" -length 0.28631
Bpm -name "BPI0242" -length 0
Drift -name "DR0244" -length 0.1905
set KC0245 [expr -$iqdf0245*$FQF*0.328*1.0e-3*$ccc]
puts [expr $KC0245/$e0]
Quadrupole -name "QDF0245" -synrad $quad_synrad -length 0.328 -
strength $KC0245
# VCORRECTOR -name "CDVE0245" -length 0
Drift -name "DR0249" -length 0.476
```

```
set KC0250 [expr $iqfe0250*$FQE*0.84*0.19*1.0e-3*$ccc]
puts [expr $KC0250/$e0]
Quadrupole -name "QFE0250" -synrad $quad_synrad -length 0.19 -strength
$KC0250
Quadrupole -name "QFE0250" -synrad $quad_synrad -length 0.19 -strength
$KC0250
Drift -name "DR0251" -length 0.476
set KC0255 $KC0245
puts [expr $KC0255/$e0]
Quadrupole -name "QDF0255" -synrad $quad_synrad -length 0.328 -
strength $KC0255
# HCORRECTOR -name "CDHE0255" -length 0
Drift -name "DR0257" -length 0.1905
Bpm -name "BPI0258" -length 0
Drift -name "DR0259" -length 0.28631
Sbend -csr $sbend_csr -csr_nbins $nbins_val -csr_nhalffiler
$nhalffiler_val -csr_filterorder $filterorder_val -csr_nsectors
$nsectors_val -csr_charge [expr $match(charge)*1.602e-19*$cc] -name
"BHC0260" -synrad $sbend_synrad -length 0.561 -angle 0.31416 -e0 $e1 -
E1 0.15708 -E2 0.15708 -six_dim 1
Drift -name "DR0265" -length 0.629589
set KC0270 $KC0230
puts [expr $KC0270/$e0]
Quadrupole -name "QFE0270" -synrad $quad_synrad -length 0.38 -strength
$KC0270
Drift -name "DR0275" -length 0.27
set KC0280 $KC0220
puts [expr $KC0280/$e0]
Quadrupole -name "QDD0280" -synrad $quad_synrad -length $LQL -strength
$KC0280
Drift -name "DR0283" -length 0.18851
Bpm -name "BPM0285" -length 0
Drift -name "DR0287" -length 0.26066106
Sbend -csr $sbend_csr -csr_nbins $nbins_val -csr_nhalffiler
$nhalffiler_val -csr_filterorder $filterorder_val -csr_nsectors
$nsectors_val -csr_charge [expr $match(charge)*1.602e-19*$cc] -name
"BHC0290" -synrad $sbend_synrad -length 0.561 -angle -0.31416 -e0 $e1
-E1 -0.15708 -E2 -0.15708 -six_dim 1
Drift -name "DR0299" -length 0.7351706
set KC0310 [expr $iqfd0310*$FQD*$LQL*1.0e-3*$ccc]
puts [expr $KC0310/$e0]
Quadrupole -name "QFD0310" -synrad $quad_synrad -length $LQL -strength
$KC0310
Drift -name "DR0315" -length 0.2
set KC0330 [expr $iqfd0330*$FQD*$LQL*1.0e-3*$ccc]
puts [expr $KC0330/$e0]
Quadrupole -name "QFD0330" -synrad $quad_synrad -length $LQL -strength
$KC0330
Drift -name "DR0333" -length 0.18918
Bpm -name "BPM0335" -length 0
Drift -name "DR0338" -length 1.01082
set KC0350 [expr -$iqdd0350*$FQD*$LQL*1.0e-3*$ccc]
puts [expr $KC0350/$e0]
Quadrupole -name "QDD0350" -synrad $quad_synrad -length $LQL -strength
$KC0350
```

10. APPENDIX V: NOMINAL VOLTAGE IN EACH CELL OF ACCELERATING CAVITY IN DBA (NOMINAL_VOLTAGE.DAT)

```
.1403
.3501
.3453
.3403
.3353
.3301
.3248
.3193
.3136
.3078
.3018
.2956
.2891
.2825
.2756
.2684
.2610
.2533
.2453
.2369
.2283
.2193
.2098
.2000
.1897
.1790
.1678
.1561
.1438
.1309
.1174
.1032
.0882
.0216
```