Fachhochschule Aachen
— Abteilung Jülich —

Fachbereich 9: Physikalische Technik
Studiengang und Studienrichtung: Physikalische Technik

Diplomarbeit

von Michela Müller

Ein System zur Erfassung und Dekonvolution von Signalen erzeugt durch Gammastrahlen in segmentierten Germaniumdetektoren.

Diese Arbeit wurde betreut von:

Prof. Dr. M. Müller-Veggian, FH Aachen
Dr. D. Bazzacco, INFN Padova

Die Arbeiten wurden durchgeführt am

Istituto Nazionale di Fisica Nucleare
— sezione di Padova —

Padova, Juni 1999.

Fachhochschule Aachen
— Abteilung Jülich —

Department 9: Physikalische Technik
Matter and subject of studies: Applied physics

Diploma thesis

by Michela Müller

A set-up for the acquisition and deconvolution of signals produced by gamma-rays in segmented germanium detectors.

Supervisors:

Prof. Dr. M. Müller-Veggian, FH Aachen
Dr. D. Bazzacco, INFN Padova

The work was carried out at the

Istituto Nazionale di Fisica Nucleare
— sezione di Padova —

Padova, June 1999.

Diese Arbeit ist selbständig angefertigt und verfaßt.
Es sind keine anderen als die angegebenen Quellen
und Hilfsmittel verwendet worden.

This work was made and composed by myself.
No other as the declared sources
and aids are used.

Padova, den: _____

_____

(Michela Müller)

Zusammenfassung

Detektor-Systeme, die in der modernen Gammaspektroskopie benutzt werden, sind zusammengesetzt aus einer großen Zahl von Germaniumdetektoren, umrundet von einem dicken Bismut-Germanat-Schild (BGO) zur Unterdrückung des Compton-Untergrundes. Die höchste auf diese Art erreichte absolute Detektoreffizienz ist etwa 10 % für γ-Strahlen mit einer Energie von 1 MeV. Höhere Effizienzen können nur durch Vermeidung des BGO-Schildes erreicht werden, so daß der gesamte Raumwinkel um die Strahlenquelle mit Germaniumdetektoren abgedeckt wird. Solch ein Array kann die benötigte Leistung (im Sinne von hoher Detektoreffizienz und geringem Compton-Untergrund) nur erreichen, wenn es in einem ortsempfindlichen Modus mit guter Ortsauflösung betrieben werden kann. Nur so werden die Positionen jener Punkte gefunden, wo die Gammastrahlen ihre Energie abgeben. Die Rekonstruktion der Abfolge von Wechselwirkungen, die zur Identifikation eines Ereignisses führt, wird ,,gamma-ray-tracking'' genannt. Ein auf diese Art arbeitendes Germanium-Array könnte eine absolute Detektoreffizienz von etwa 50 % erreichen und wäre dann das wichtigste Experimentiergerät des nächsten Jahrzehnts zur Untersuchung der Kernstruktur.

Ein wichtiger einleitender Schritt für diese Entwicklungen ist, die erreichbare Ortsauflösung in einem realen Detektor zu testen. Diese Diplomarbeit beschäftigt sich mit dem Aufbau der Hardware und den Software-Tools zur Durchführung eines solchen Tests mit einem hochsegmentierten, großvolumigen Germaniumdetektor, der in Italien von der *MARS*-Forschungsgruppe gekauft wurde. Die Hardware besteht aus Flash-ADCs mit hoher Samplingrate, um die Signale des Detektors zu erfassen. Zum Aufdecken der ursprünglichen Signalform führt die entwickelte Software die Dekonvolution der erfaßten Signale aus, welche die Information über die Positionen der Energieabgabe enthält. Die ,,response function'' der benutzten Vorverstärker, die eine wichtige Information für die Dekonvolution darstellt, wurde ebenfalls ermittelt. Das System wurde mit einem existierenden unsegmentierten Detektor getestet, und das Signal wurde nach der Dekonvolution mit Ergebnissen aus Modellrechnungen verglichen.

Nachdem einige Probleme im Zusammenhang mit elektronischem Rauschen gelöst waren, hat sich die Dekonvolution als stabil erwiesen und die rekonstruierten Signale haben plausible Formen. So ist der Weg frei für spätere Messungen mit dem segmentierten Detektor.

Abstract

The detection systems used in the modern γ-ray spectroscopy are composed by a large number of germanium detectors surrounded by a thick bismuth germanate (BGO) shield for the suppression of the Compton background. The highest absolute detection efficiency obtained in this way is about 10 % for a gamma-ray energy of $1\,MeV$. Higher efficiencies can be obtained only by avoiding the use of the BGO shield, in order to fully cover the solid angle around the radiation source with germanium detectors. Such an array can provide the needed performance (in terms of high detection efficiency and low Compton background in its response function) only if it can be operated in a position sensitive mode, with good spatial precision. Only in this way the positions of the points are found where the gamma-rays release their energy. The reconstruction of the sequence of interactions leading to the detection of a transition is called gamma-ray-tracking. A germanium-array working in this way could have an absolute detection efficiency of about 50 % and would be the most important experimental tool for nuclear structure studies for the next decade.

An important preliminary step for these developments is to test the position resolution performance of a real detector. This thesis work is concerned with the set-up of the hardware and the software tools to perform such a test on a highly-segmented germanium detector with large volume, which has been purchased in Italy by the *MARS* collaboration. The hardware consists of flash-ADCs with a high sampling rate to collect the signals from the detector. The developed software performs the deconvolution of the collected signals to recover the original shape which contains the information about the positions of the energy release points. The response function of the used preamplifiers, which is an important piece of information for the deconvolution, has also been derived. The system has been tested with an existing unsegmented detector and the deconvoluted signals are compared with results from model calculations.

After some problems related to electronic noise were solved the deconvolution has proven to be stable and the reconstructed signals have plausible shapes. So the way is free for later measurements with the segmented detector.

Contents

Chapter 1

Introduction

Most of the knowledge about the excitation structure of the atomic nucleus has been obtained by means of the precise detection of the electro magnetic radiation (gamma rays) emitted when an excited nucleus changes from one state to another one lying at lower energy. As the density of excited states is typically very high, the emission spectrum is rather complicated, so that the energy resolution of the experimental devices used to detect it is extremely important.

The most significant advance in nuclear spectroscopy was made about 30 years ago when the semiconductor radiation detectors for gamma rays were first introduced. With their exceptional energy resolution these detectors allow, in fact, to distinguish individual transitions even in very complicated spectra. Both silicon and germanium can be used, but for the most important energy range between $\approx 50\,keV$ and $\approx 10\,MeV$ high purity germanium is the best choice. The volume of the first germanium detectors was of only a few cubic centimetres but a continuous progress in the crystal production technology has made it possible to increase it, and therefore the detection efficiency, by about two orders of magnitude yielding the modern semi-coaxial high-purity crystals.

In the same time, the introduction of the concept of Compton background suppression by means of a vetoing shield which surrounds the germanium crystal, has brought a significant improvement of the response function: peak-to-background ratios of about 60 % at $1\,MeV$ are typical of present day detector systems.

The need to increase the detection efficiency in order to be able to detect weakly populated excitation structures, has resulted in the construction, in the last decade, of the so-called multi-array systems, where as many as economically affordable Compton shielded high-purity large-volume germanium detectors are closely packed around the radiation source filling almost completely the available solid angle. Typical example of these detector systems are *GASP*, *EUROGAM*, *GAMMASPHERE*

and *EUROBALL*.

GASP has been built at the National Laboratories in Legnaro ("*Laboratori Nazionale di Legnaro*", *LNL*, Padova, Italy) of the Italian National Institute of Nuclear Physics ("*Istituto Nazionale di Fisica Nucleare*", *INFN*) in 1992 and has an efficiency of 3 % with 40 detectors at the standard reference energy of 1.33 MeV.

EUROGAM was operated in 1993 at Daresbury (United Kingdom) and then at Strasbourg (France) in 1995 and had an efficiency of about 5 % with about 50 detectors.

GAMMASPHERE was built in Berkeley (United States) starting in 1993 and has finally reached an efficiency of about 10 % in 1998 when all its 110 detectors were put in operation.

EUROBALL, with its 10 % efficiency, is the result of an European collaboration among Denmark, France, Germany, Italy, Sweden and the United Kingdom. It has been in operation at *LNL* in 1997 and 1998 and is now installed at the "*Institut de Recherces Subatomique*" (*IReS*) in Strasbourg, France. EUROBALL is composed of 72 detectors but, as several of them are composite, the total number of germanium crystals is 239.

The larger and larger number of detectors used in these arrays is due to the fact that up to 30 gamma rays can be emitted in de-excitation cascade before an excited nucleus reaches its ground state. As these gammas are almost simultaneous and their emission directions more or less isotropic, the solid angle of each individual germanium crystal must be very small to have a sufficiently small probability that two or more of them enter the same detector where they cannot be anymore distinguished (summing effect).

These big detector systems have provided a lot of new knowledge about important aspects of nuclear physics like super-deformation and structure of exotic nuclei far from the valley of stability, paying for the efforts spent by a lot of people in their construction and operation. However, there are many other theoretically predicted phenomena (like for example hyper-deformation) that have not been found even with these devices and, clearly, much larger detection efficiencies are needed in order to progress further in the study of nuclear structure.

It seems however that 10 % is the maximum reachable value with the present day technology which is based on the largest high-purity germanium crystals but needs the Anticompton shield to improve the peak-to-background ratio. Even in the most efficient arrays the Anticompton shield occupies more than 50 % of the solid angle so that we are at least a factor of 2 away from the theoretical limit of $4\,\pi$.

Already in the project phase of the present generation of gamma-ray arrays it

has been verified, by means of Monte Carlo simulations, that it is not economically affordable to build a $4\,\pi$ detector out of only germanium crystals (a so called germanium shell) because to obtain the desired performance more than 1000 large volume germanium crystals are needed.

1.1 Evolution towards a γ-ray tracking array

A few years ago it was suggested that a compact (and therefore relatively cheap) detector could be built if the individual crystals composing it can be operated in a three dimensional position sensitive mode, like it is done since long time in the big particle detectors of the high energy physics. In this case it would be possible to determine the history of each gamma while it is absorbed in the germanium crystal (usually in a sequence of several Compton scattering interactions) distinguishing it from the other transitions of the de-excitation cascade and avoiding the use of the Compton shield. This way of working is called "gamma-ray tracking" and is being investigated by the *GRETA* group in the United States (which proposed first this new concept) and by the *EUROBALL* community in Europe, within the *TMR* network project "Development of Gamma-Ray Tracking Detectors for $4\,\pi$ Gamma-Ray Arrays".

The position sensitivity of a big semi-coaxial germanium crystal is obtained by the simultaneous use of two new technologies:

First of all the outer surface of the detector is electrically segmented into many (20-50) pixels which give a rough indication of the region of the crystal where some energy has been released; Secondly the rising edge of the signals from the segments is carefully studied because its detailed shape contains information about the radial position of the point of energy release and about its position with respect to the borders of the segment.

The evolution of the detector set-ups used in gamma-ray spectroscopy is shown schematically in Figure 1.1 taking as example the history of two γ-rays detected in a classical Compton shielded array (A), in a germanium shell (B) and in a compact tracking array (C). If considered separately, transition "1" would be detected correctly in all three cases; for (B) and (C) this is achieved by summing the signals from neighbouring crystals. Transition "2" is rejected by the Compton shielded detector because a part of its energy is deposited in the shield surrounding the germanium crystal. On the other side, if the two γ-rays arrive simultaneously both of them are rejected in case (A); in case (B) they are wrongly interpreted as one transition with an energy corresponding to the sum of the two individual energies. Only if the

position and energy of the individual interaction points can be determined, as in the case of the tracking array, is it possible to decide to which gamma-ray they belong and to recover correctly the event. Also the Doppler-broadening, as a side-effect of this innovation, will be reduced by the detection of the first gamma-hit, because the identification of its position corresponds to a good measurement of the emission angle.

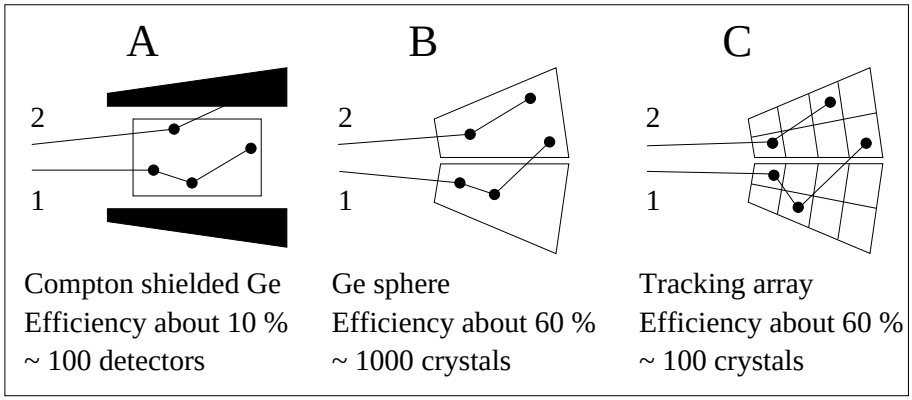
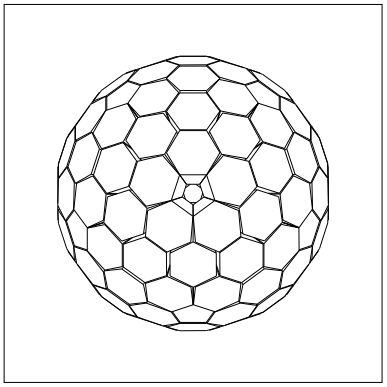


Figure 1.1: Advantages of a tracking array.

1.2 The *MARS* project

In Italy these studies are funded by the *INFN* in a research and development project called *MARS* (*Mini ARray of Segmented detectors*) and this diploma thesis reports on work done at the Department of Physics of the Padova University and at *LNL* to prepare for the tests of a prototype germanium detector with the outer surface segmented in 25 parts.

The main goal of the *MARS* project is to decide if the principle of γ-ray tracking works with real detectors. This will be done with test measurements performed on the segmented prototype detector and comparing their results with predictions obtained from Monte Carlo simulations of the same detector. In the case of a positive result the next step will be the construction, in collaboration with the other involved European countries, of a new γ-ray array to be used at *LNL*. In the following figure 1.2 a few of the considered configurations for this future tracking array are presented. As an indication of their complexity the total number of segments is given for each of them, considering that each germanium crystal is segmented in 25 parts. Of course all numbers must be scaled up if the experience with the *MARS* prototype will show that more segments are needed to obtain the desired position resolution.

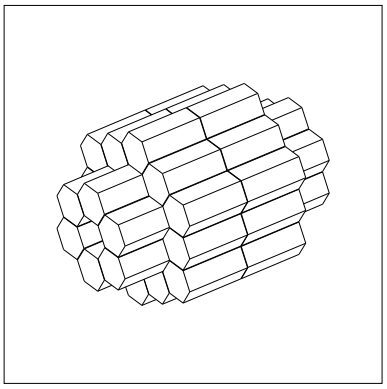


"The nicest"

Spherical with 120 detectors:

- 30% photopeak-efficiency
- 45% peak to total radiation
- 3000 segments

To produce it one needs 250 kg of germanium but after cutting the crystals to the proper shape only 120 kg remain. This configuration would be very expensive both for the amount of used material and for the number of electronics channels.

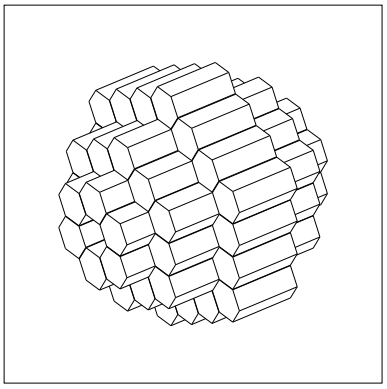


"The cheapest"

Cylindrical with 36 detectors:

- 30% photopeak-efficiency
- 45% peak to total radiation
- 900 segments

110 kg germanium used of originally 140 kg, one loses only 30 kg.



"The most effective"

Cylindrical with 54 detectors:

- 35% photopeak-efficiency
- 50% peak to total radiation
- 1350 segments

140 kg germanium used of originally 180 kg, one loses only 40 kg.

Figure 1.2: Possible *MARS* geometries.

It is important to consider that the referred efficiencies were calculated as response to individual transitions and without gamma-ray tracking. With a good tracking algorithm the peak to total should increase up to 60 % with a small loss of efficiency. Developing efficient tracking algorithms is another part of the *MARS* project. (The pictures and calculation results shown in this section are taken from the paper of the most recent *TMR* user meeting [13]. The number of segments is recalculated to fit the prototype.)

1.3 The Prototype Ge Detector

The final gamma-tracking array, in whichever version, will be very expensive. Before constructing it, one has to find out, by means of smaller experiments, if it would work correctly. For this reason the *Mars* group has purchased a prototype version of the planned segmented detectors from *Eurisys* in France.

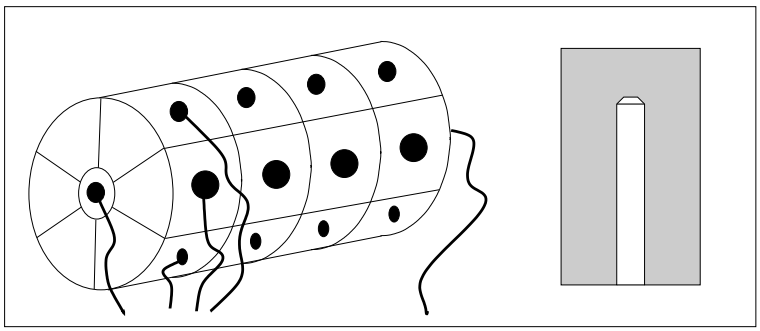


Figure 1.3: The prototype detector.

The crystal of this prototype detector has a cylindrical shape, although in the final array it will be probably hexagonal. Its diameter is $72\,mm$ and the length is $90\,mm$. Its approximate weight is about $2\,kg$. Like almost all large volume germanium detectors its geometry is not true coaxial because the inner hole stops about $1\,cm$ from the front of the crystal. This so called semi-coaxial geometry is currently used to increase the reliability of the detectors as it avoids one of the two delicate passivated surfaces which separates the voltage difference applied between the inner hole and the outer cylindrical surface. The loss of symmetry complicates the situation and it is more difficult to calculate the shape of the signals seen in the various segments. However this still can be done with numerical methods, and as the result of this extra effort is that the reliability of the detector increase.

The outer surface is segmented in 6 slices and 4 planes with an extra small segment in the front face; the total number of segments is therefore 25. The total

number of signals from this detector is 26 because also the inner hole is connected to obtain the sum of whatever happens in the whole germanium crystal. As it will be explained later the germanium crystal is operated at liquid nitrogen (LN_2) temperature and therefore the crystal is contained in a suitable cryostat. The 26 FETs (Field Effect Transistors) for the charge preamplifiers are also operated at low temperature in order to have the best energy resolution. This detector is mechanically compatible with the detectors used by *GASP* at *LNL* and can be easily inserted in the existing array to check the performance of the prototype in a realistic experimental situation.

Chapter 2

Nuclear radiations and HPGe detectors

Being used at liquid nitrogen temperatures the HPGe detectors are contained in a cryostat which shields them from radiation of charged particles. Only gamma and neutron radiation can enter into the crystal. Generally in a nuclear physics experiment several types of reactions take place in the same time. Some of them produce gamma or neutron radiation which can be registered with the detectors of interest in this text. Unless otherwise marked all the information in this chapter is taken from the book *Radiation detection and measurement* (G. F. Knoll) [7], which is also a source to learn more about them.

2.1 Nuclear radiations and ways to produce them

Beta decay

The most common source of fast electrons or positrons in radiation measurements is a radio isotope which decays by beta emission. The process is written schematically

$$ {}^{A}_{Z}X \longrightarrow {}^{A}_{Z+1}Y + \beta^{-} + \bar{\nu} \qquad {}^{A}_{Z}X \longrightarrow {}^{A}_{Z-1}Y + \beta^{+} + \nu \tag{2.1} $$

where X and Y are the initial and final nuclear species, and ν ($\bar{\nu}$) is the neutrino (antineutrino). As neutrinos and antineutrinos have an extremely small interaction probability with matter, they are undetectable for all practical purposes. The recoil nucleus Y has a very small energy, which normally cannot be detected by radiation detectors. Thus, the only significant ionising radiation produced by beta decay is the fast electron or beta particle itself, with energies roughly between a few keV and a few MeV.

Internal conversion

The internal conversion process begins with an excited nuclear state which may be formed by a preceding process, often beta decay of a parent species. The de-excitation of such states through emission of an electron from the atomic shell surrounding the nucleus is called internal conversion. The energy of the electron ranges from high keV up to MeV and is a good source for calibration because this effect causes mono energetic spectra with well known energies.

Alpha decay

Heavy nuclei are energetically unstable against the spontaneous emission of an alpha particle (or ${}^4He$ nucleus). The half-life of useful sources varies from days to many thousands of years. Because of a correlation between alpha particle energy and half-life of the parent isotope there are no useful isotopes with energies beyond about $6.5\,MeV$. If the energy is below about $4\,MeV$, the lifetime is very long and therefore the activity of a source becomes very small. The decay process is written schematically as

$$ {}^{A}_{Z}X \longrightarrow {}^{A-4}_{Z-2}Y + \alpha \tag{2.2} $$

where X and Y are the initial and final nuclear species.

Spontaneous fission

The fission process is the only spontaneous source of energetic heavy charged particles with mass greater than that of the alpha particle. All the heavy nuclei are, in principle, unstable against spontaneous fission into two lighter fragments. But because of the large potential barrier this process is only significant for some transuranic isotopes of very large mass number. Normally one of the fragments is absorbed in the material of the radiation source and only the other fragment escapes outside of the source. In most spontaneous fission processes also a number of fast neutrons is liberated resulting in a source for neutrons with an energy peak at about $1\,MeV$ but a significant yield of neutrons up to $10\,MeV$. If used as a neutron source, the isotope is generally encapsulated in a sufficiently thick container so that only the fast neutrons and gamma rays emerge from the source.

Radioisotope neutron sources

Because energetic alpha particles are available from the direct decay of a number of convenient radionuclides, it is possible to fabricate a small self-contained neutron source by mixing an alpha-emitting isotope with a suitable target material. The maximum efficiency is obtained when beryllium is chosen as the target, and neutrons are produced through the following reaction:

$$ \alpha + {}^{9}_{4}Be \longrightarrow {}^{12}_{6}C + n \tag{2.3} $$

In the case of beryllium the energy of the neutron is $5.71 MeV$. For alternative materials (B, F, C) the energy is lower.

Photoneutron sources

Some radioisotope gamma ray emitters can also be used to produce neutrons when combined with an appropriate target material. The resulting photoneutron sources require a gamma ray photon with an energy of at least the one of the escaped neutron to make one of the following reactions possible:

$$\gamma + {}^{9}_{4}Be \longrightarrow {}^{8}_{4}Be + n \qquad \gamma + {}^{2}_{1}H \longrightarrow {}^{1}_{1}H + n \tag{2.4}$$

The neutron energy for the ${}^{9}Be$ and ${}^{2}H$ case respectively is 1.666 MeV and 2.226 MeV. Other target nuclei with practical significance are not known.

Accelerators

Using particle accelerators it is possible to produce artificial radiation of high energy, that means ions, and electrons. Neutrons and gammas cannot be accelerated because they have no electric charges. The maximum kinetic energy that can be produced in the *Laboratori Nazionali di Legnaro (LNL)* of the I.N.F.N., is 15 MeV multiplied by the ionization charge of the accelerated particle species.

Gamma radiation

Almost all nuclear reactions and beta decay processes produce also gamma radiation. The energy spectrum varies very much from case to case. Beta decay is very useful to calibrate gamma-ray detectors as it produces characteristic lines with energy ranging from a few keV up to 6 MeV. The study of the spectrum of unknown isotopes produced with exotic reactions is the very reason why nuclear physicists are interested in building gamma counting arrays with increasing efficiency and detection power.

Annihilation radiation

A peculiar type of gamma radiation is produced when a positron and an electron meet together and are destroyed, leaving only energy. This process is called *annihilation* and generates two gamma rays of 0.511 MeV.

Bremsstrahlung

When fast electrons interact in matter, a part of their energy is converted into electro-magnetic radiation in the form of bremsstrahlung. The fraction of converted energy increases with the atomic number of the absorbing materials. The energy is continuous from some keV up to the energy of the electron itself. This effect is used for example in conventional X-ray tubes.

Characteristic X-rays

If an orbital electron in an atom falls back from an excited state this causes a X radiation of a defined wavelength. The spectra are well known but of low energies up to $0.1\,MeV$ for radium, for example. Physically, of course, there is no difference between X-rays and gamma radiation.

2.2 Interaction of gamma rays

Although several interaction mechanisms are known for gamma rays in matter, only three major types play an important role in germanium crystals and probably in radiation measurements in general. They are described in the following parts.

Photoelectric absorption

In the photoelectric absorption process, an incoming gamma ray photon with an energy E_γ undergoes an interaction with an absorber atom in which the photon completely disappears. In its place, an energetic *photoelectron* is ejected by the atom as a whole.

$$E_{e^-} = E_\gamma - E_b \tag{2.5}$$

where E_b represents the binding energy of the photoelectron in its original shell. Also Auger electrons or X-ray photons may be produced in this way inside the germanium crystal. The photoelectric effect cannot take place with free electrons.

Compton scattering

The interaction process of Compton scattering takes place between the incident gamma ray photon and an electron in the absorbing material. The incoming gamma ray photon is deflected through an angle θ with respect to its original direction. The photon transfers a portion of its energy to the electron, which is then known as a *recoil electron*. Because all angles of scattering are possible, the energy transferred to the electron can vary from zero to a large fraction of the gamma ray energy. The following equation describes this effect, using the symbols defined in figure 2.1:

$$E'_\gamma = \frac{E_\gamma}{1 + \frac{E_\gamma}{m_0 c^2}\,(1 - \cos\theta)} \tag{2.6}$$

where $m_0 c^2$ is the rest mass energy of the electron ($0.511\,MeV$). The angular distribution of scattered gamma rays is described by the "Klein-Nishina formula" which can be found in [7].

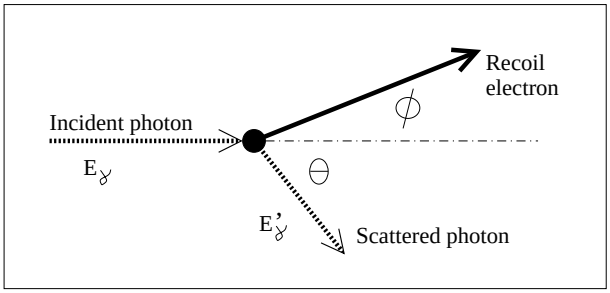


Figure 2.1: Sketch to explain the Compton scattering.

Pair production

If the gamma ray energy exceeds twice the rest mass energy of an electron (that is $1.02\,MeV$), the process of pair production is energetically possible: the gamma ray photon disappears and is replaced by an electron – positron pair. This process must take place in the coulomb field of a nucleus. The energy above the $1.02\,MeV$ needed to create the pair is shared by the positron and the electron as kinetic energy. Because the positron will subsequently annihilate after slowing down in the medium, two annihilation photons (each with an energy of $511\,keV$) are normally produced as secondary products of the interaction. As a practical matter, the probability of this interaction remains very low until the gamma ray energy approaches twice this value, and therefore pair production is predominantly confined to high energy gamma rays. No simple expression exists for the probability of pair production per nucleus, but its magnitude varies approximately as the square of the absorber atomic number.

Interaction of neutrons

Neutrons carry no charge and therefore cannot interact in matter by means of the coulomb force which dominates the energy loss mechanisms for charged particles and electrons. When a neutron interacts with an atomic nucleus, it may either totally disappear and be replaced by one or more secondary radiations, or its energy and/or direction can be changed. In contrast to gamma rays, the secondary radiations resulting from neutron interactions are almost always heavy charged particles. One can distinguish distinguish two kinds of interactions according to energy of the neutrons.

- Slow neutron interactions
  Because of the small kinetic energy of slow neutrons, very little energy can be transferred to the nucleus in elastic scattering. The slow neutron interactions of real importance are neutron-induced reactions which can create secondary

radiations of sufficient energy to be directly detected. In most materials, also germanium, the radiative capture reaction, also called (n, γ) reaction, is the most probable.

- Fast neutron interactions
  The probability of most neutron-induced reactions drops off rapidly with increasing neutron energy. But if the energy of the neutron is sufficiently high, inelastic scattering with nuclei can take place in which the recoil nucleus is elevated to one of its excited states during the collision. The nucleus quickly de-excites, emitting a gamma ray.

2.3 High energy-resolution gamma ray detectors

The main purpose of many applications of radiation detectors is to measure the energy distribution of the incident radiation in as much detail as possible. In the ideal case the response function of the detector should therefore be a single very narrow peak with an amplitude proportional to the energy of the incident gamma. However there are fundamental and practical limitations on the energy resolution one can achieve. A fundamental limit is the finite number of charge carriers produced by the absorbed radiation. This means that the peaks have a Gaussian form with a width proportional to the square root of the number of carriers. From this point of view semiconductor radiation detectors give the best energy resolution because of the very small energy needed to create electron–hole pairs, which is about ten times smaller than the energy needed to create a photon in a typical scintillation crystal. A practical limit to the energy resolution for a real detector is for example the electronic noise. The performance of a detector with respect to energy resolution is given by the *full width of half maximum (FWHM)* of the peak produced by some standard radioactive sources. In the following the discussion will be limited to germanium detectors as this is the standard kind used in high resolution gamma ray spectroscopy.

Gamma rays are absorbed in germanium mainly by means of photoelectric effect, Compton scattering and pair production. In the process of the photoelectric effect the whole energy of the incident gamma is given to an electron which in turns release it as electron–hole pairs as it slows down in the crystal. In this way a sharp peak is produced with an amplitude proportional to the energy of the incoming gamma. If the gamma undergoes Compton scattering, a secondary gamma, which can escape from the detector, is produced. If this happens, only a part of the energy of the incoming gamma is absorbed in the detector. This means that, beside a peak

coming from full absorption of all secondary gammas, also a continuous background is present in the spectrum.

Figure 2.2 shows the energy dependence of these three effects in germanium. The data to draw the figure is taken from [8]. Photoelectric absorption plays an important role for low energies, and the pair production is important for high energies. For energies from $200\,keV$ up to $7\,MeV$ the Compton scattering plays the most important role. Normally this is the energy range used in nuclear structure experiments, so it is worth to think about a suppression of the Compton scattering effect with escaped secondary gammas while registering the experimental data.

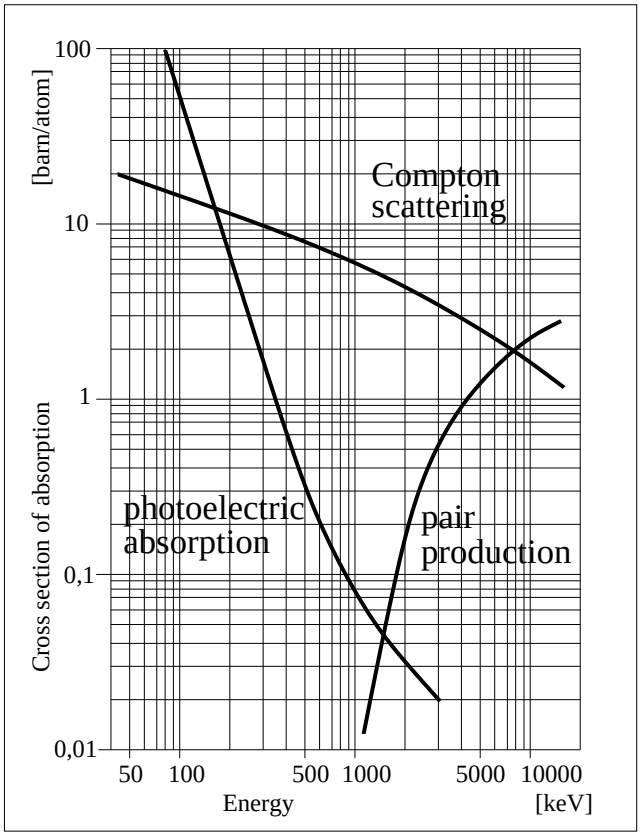


Figure 2.2: Energy dependence of interaction processes in Ge.

It is in fact rather common to surround the germanium crystal with an efficient scintillator (like NaI or BGO) coupled with photomultipliers to detect the escaping secondary radiation. With a proper anti coincidence circuit the acquisition of signals from the germanium detector can be inhibited if the surrounding scintillator has seen some signal at the same time. In this way up to 60 % of the background can be suppressed (see figure 1.1 A).

To obtain the best energy resolution germanium detectors must be operated at

liquid nitrogen temperature and therefore they are kept in a high vacuum aluminium case. This aluminium acts as an absorber and might be disturbing for incoming radiation of very low energy.

In typical nuclear reactions used to produce and study unstable nuclei, a few fast neutrons are also emitted. If these neutrons reach the detector they can hit one of the germanium atoms knocking it off its position and destroying the crystalline lattice at the interaction site. This process produces trapping sites for the electrons or holes generated by successive incoming gammas, so that part of the generated charge carriers are not anymore collected and the energy resolution is degraded. Fortunately a damaged crystal can be recovered by an annealing process at 110–120 ^0C. However, heating the detector up to such temperatures is a delicate and dangerous process which can even destroy the whole crystal or some of its parts. Therefore neutron damage should be avoided as much as possible.

2.4 Important effects inside a semiconductor

The periodic lattice of crystalline materials establishes allowed energy bands for electrons that exist within that solid. The energy of the electrons in the pure material must be confined to one of these bands which may be separated by gaps or ranges of forbidden energies. Figure 2.3 represents a simplified model. The lower band, called the *valence band* E_v, corresponds to those electrons that are bound to specific lattice sites within the crystal. The next higher-lying band is called the *conduction band* E_c and represents electrons that are free to migrate through the crystal. The two bands are separated by the *band gap* E_a, the size of which determines whether the material is classified as a semiconductor ($E_a \cong 1\,eV$) or an insulator ($E_a > 5\,eV$).

As at any temperature (above absolute zero) some thermal energy is shared by the electrons in the crystal, a few of them are elevated across the band gap into the conduction band, where they can move freely and can be collected by applying an electric field. This excitation process not only creates an electron in the otherwise empty conduction band, but it also leaves a vacancy (called a *hole*) in the otherwise full valence band. The hole, representing a net positive charge, will also tend to move in an electric field, but in a direction opposite that of the electron. The drift velocity in an electric field depends on the material and the temperature. The values for germanium are the following:

Electron mobility ($300\,K$):	0.39 x $10^4\,cm^2/Vs$
Electron mobility ($77\,K$):	3.6 x $10^4\,cm^2/Vs$
Hole mobility ($300\,K$):	0.19 x $10^4\,cm^2/Vs$
Hole mobility ($77\,K$):	4.2 x $10^4\,cm^2/Vs$

These values are given for low to moderate electric fields. At high electric field values, the drift velocity increases more slowly and reaches a *saturation value* which becomes independent of further increases in the electric field. Normally Ge detectors are operated with electric field values sufficiently high to result in saturated drift velocity for the charge carriers. These saturated velocities are of the order of $10^7\,cm/s$ (that is, it takes about $100\,ns$ to travel a distance of $1\,cm$) for both electrons and holes. Semiconductor detectors can therefore be among the fastest-responding of all radiation detector types if designed as a thin layer.

Semiconductors can be used as detectors because when they absorb the energy of an incoming radiation a certain number of electrons are displaced into the conduction bands. Part (a) of figure 2.3 shows a possible situation for a few electrons after a nuclear event. Part (b) shows the same situation but a short time later, after the secondary processes are finished and all the electrons are collected at the bottom of the conduction band, whereas the holes are collected at the top of the valence band. The dominant advantage of semiconductor detectors lies in the smallness of the ionization energy (for Ge at $77\,K$ it is $2.96\,eV$), which means that they will

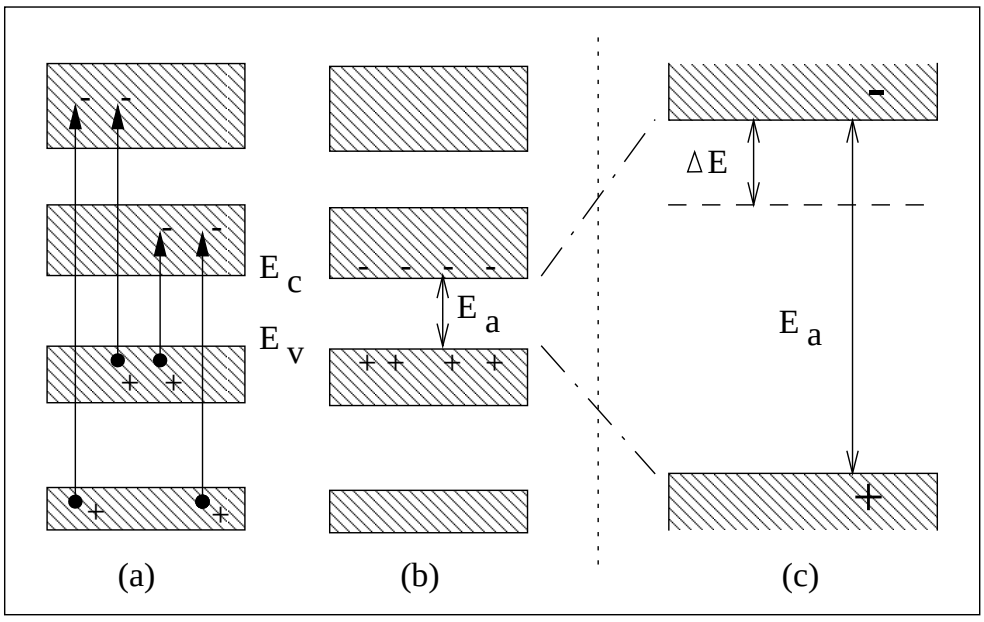


Figure 2.3: Band model of semiconductors.

provide a good energy resolution because of the large number of ions (electrons and holes) that are produced.

This picture is valid for a pure semiconductor. However, in any real material there is always a small amount of impurities which occupy substitutional sites in the lattice structure. If the impurity atom belongs to the group V elements of the periodic table, the extra electron (with respect to tetravalent Ge or Si) remains only lightly bound to the original impurity site and therefore contribute to the conduction band. These electrons are referred to as *donor electrons*. If instead of this the impurity is one of the group III elements the missing electron results in a free hole, known as *acceptor hole*, which contributes to the valence band. For practical semiconductors the type and amount of impurities is decided by the manufacturer and the following are possible materials that could be used:

n-type:	$Li\,(0.0093)$; $P\,(0.012)$; $As\,(0.013)$; $Sb\,(0.0096)$
p-type:	$B\,(0.01)$; $Al\,(0.01)$; $Ga\,(0.011)$; $In\,(0.011)$; $Tl\,(0.01)$; $Pt\,(0.04)$

The numbers in brackets are the energy distance from the corresponding band edge in eV. In part (c) of figure 2.3 is shown the effect of doping for a n-type doped crystal. Most electrons from the energy level of the impurity escape to the conduction band, because of the thermal motion. This results in much (roughly million times) more free electrons in the conduction band as holes in the valence band, therefore the movement of negative charges is favoured. That opens the possibility for the electrons to fill external holes (positive charges) coming from an electrical contact. In a p-type crystal the same is valid for the holes and the valence band respectively. The application as a Ge detector is described in the next section. For other electronic semiconductor devices one can refer to [6].

There are other impurities which are not useful because they have multiple energy levels inside the band gap of Ge, for example Cu, Ag, Au, Zn, Mn, Fe, Co, Ni. If these impurities exist inside the crystal they disturb the proper function. A special case is Pb whose energy distance from the band edge is $0.2\,eV$. In principle the use of it is possible, but the needed thermal energy with temperatures of several $100\,^0C$ does not allow any practical application. In this section only germanium was discussed. The same physical picture is valid also for other semiconductor materials like Si for example, of course with other values for the various quoted quantities.

2.5 High-Purity Germanium (HPGe) detectors

Germanium is almost always preferred to silicon as a material for gamma ray spectroscopy above about 100 keV. If germanium of usual semiconductor purity is used, it is impossible to create depletion depths thicker than a few mm.

To overcome this problem, the lithium drifting process has been developed roughly 30 years ago to artificially create an "intrinsic" zone whose thickness can reach $10 - 15\,mm$. The volume of these $Ge(Li)$ detectors can therefore be made sufficiently large to be of interest in gamma ray spectroscopy. The major practical disadvantage of $Ge(Li)$ detectors is that the lithium spatial distribution becomes unstable at room temperature. To avoid damage the detectors have to be stored always at a reduced temperature, typically through the use of a liquid nitrogen dewar. A single failure to keep the detector cold usually means that the device must be returned to the manufacturer for re-drifting. This process is expensive and does not guarantee that the original performance can be reached again.

In more recent years it has become possible to produce "High-Purity Germanium" (HPGe) with impurities less than 10^{10} atoms per cm^3. This "is perhaps the most highly purified and completely analysed material that has ever been produced" (G. F. Knoll). The residual acceptor or donor impurities, cause a p-type or n-type semiconductor respectively. Excessive leakage current prevents the use of any Ge detector at room temperature, but HPGe can be stored at room temperature between the uses. Nevertheless, the detectors are normally stored at low temperature to avoid mechanical problems or surface contamination caused by the cycles of warming and cooling. Both p-type and n-type HPGe detectors are currently used in high energy resolution spectroscopy. However, n-type is preferred in nuclear physics because it is less sensitive to damage induced by fast neutrons which are almost always produced in reactions induced by accelerated beams.

The basic configuration of a n-type Ge detector is shown in figure 2.4. A diode junction is formed at one end of the crystal by means of a highly doped p^+ layer: the diffusion of free electrons and free holes from the two sides creates a depleted region with a thickness that can be extended to the other side of the detector by the application of a negative voltage to the p contact (*reverse biasing*). On the other side of the crystal an electric contact is created by means of a highly doped n^+ layer. In commercial germanium detectors the p^+ region is created implanting a thin (a few μm) layer of boron while the n^+ is obtained by diffusion of a thick (roughly $1\,mm$) layer of lithium.

Large-volume n-type gamma-ray detectors are produced in a (semi)coaxial configuration with the thin boron layer on the outside and the thick radiation absorbing

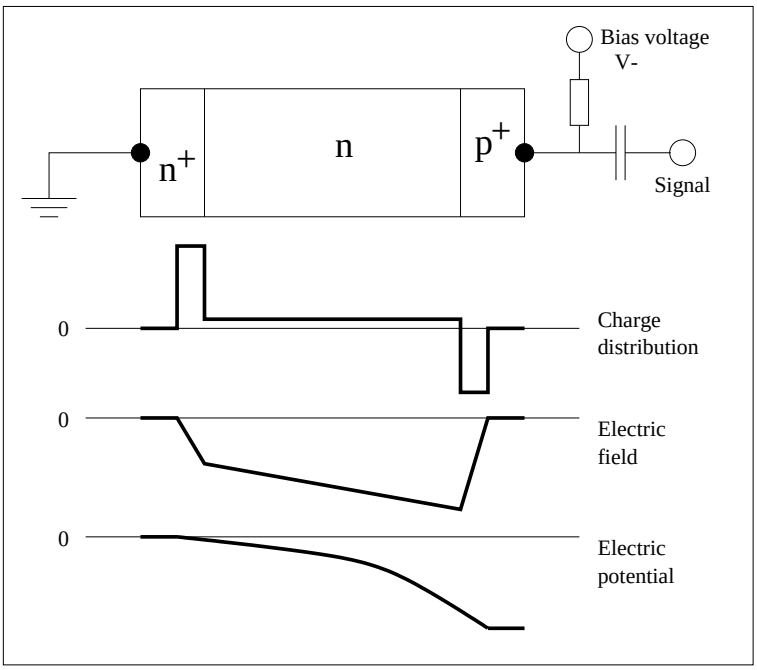


Figure 2.4: Configuration of an intrinsic germanium detector.

lithium on the inner hole. This is one of the reasons to prefer n-type instead of p-type detectors in gamma-ray spectroscopy because in the p-type case the thick lithium drifted layer (forming the p–n junction) would be on the outside and would therefore absorb a part of the incoming radiation.

In the figure are added also the charge distribution, the electric field and the electric potential. Reverse biasing requires that a negative voltage has to be applied to the p^+ contact. As one can see in the sketch the detector is usually over-biased with respect to the voltage needed to fully deplete the detector. So the minimum electric field is still sufficiently high to impart saturation velocity to the charge carriers, minimising the collection time and the effects due to recombination and trapping. Practical problems related to breakdown and surface leakage often limit the maximum voltage to values at which electrons but not holes will reach saturated drift velocity.

Large volume detectors are used in the so called *closed-end configuration* with a coaxial part and a closed one. The calculation of the electric field and the electric potential in such a geometry is very difficult but can be solved with the method of finite elements. Simplifying the geometry to a cylindric crystal (of r_2) with a cylindric hole (of r_1) closed formulas can be derived:

$$V_d = \frac{\rho}{2\epsilon}\left[r_1^2 \ln\left(\frac{r_2}{r_1}\right) - \frac{1}{2}\left(r_2^2 - r_1^2\right)\right] \tag{2.7}$$

$$E(r) = \frac{\rho}{2\epsilon}\, r - \frac{V + \frac{\rho}{4\epsilon}\,(r_2^2 - r_1^2)}{r\,\ln\,(r_2/r_1)} \tag{2.8}$$

These expressions show high field areas toward the outer regions of the HPGe detector, where is also the most of the detector volume. The higher field may not effect electron collection times but does aid in reducing hole collection time because holes seldom reach saturation. A reachable value for time resolution in coaxial HPGe crystals is $2\,ns$ [Canada 1977, by B. C. Robertson (Department of Physics, Queen's University, Kingston) and H. L. Malm (Aptec Engineering Ltd., Toronto)]. This time resolution is obtained with standard electronics.

The energy resolution of HPGe detectors is very good. For commercial detectors in the scientific field $150\,eV$ resolution at about $6\,keV$, $1\,keV$ at about $660\,keV$ and $1.7\,keV$ at $1.33\,MeV$ are possible. At low energies, the resolution is dominated by the electronic noise of the system, which depends on the detector capacitance.

2.6 The charge collection process in coaxial detectors

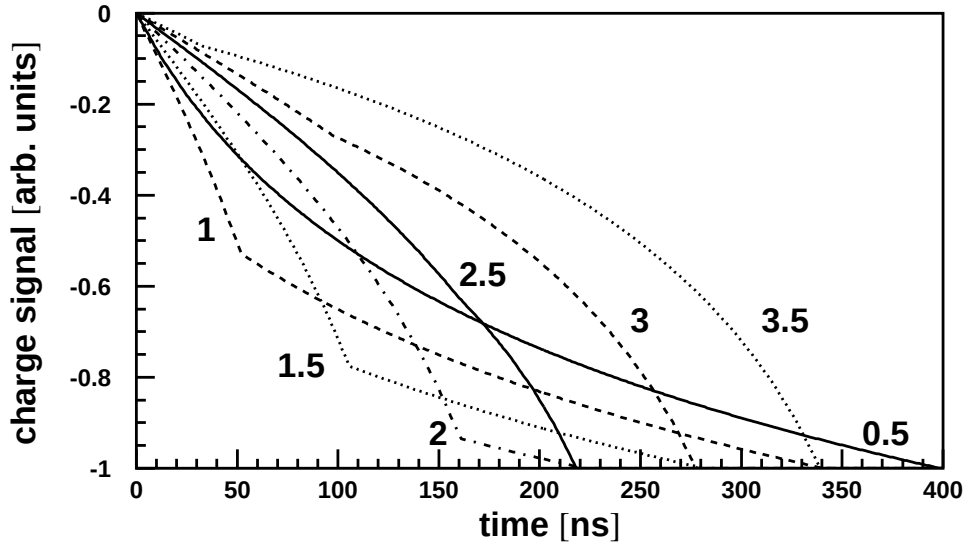


Figure 2.5: Leading edge of the output pulse for a coaxial detector at different radial positions (in cm).

As in other detectors in which signal carriers are collected over appreciable distances, the shape of the rising pulse edge from germanium detectors depends on the position where the charge carriers are formed inside the active volume. To a good approximation, very short-range particles create all the electron-hole pairs at one

location inside the detector. If that location is in the interior of the active volume, there will be unique and separate collection times for holes and electrons because each of them must travel a fixed distance to be collected. In the special case of an interaction near an edge of the active volume, the observed pulse shape is mainly to the motion of only one type of charge carrier. For charged radiations whose range is not small compared with the active volume, a distribution of collection times will result from the corresponding spatial distribution of the points at which holes and electrons are formed. The pulse shape generated by the collection of electrons and holes in a coaxial detector can be calculated in a simplified way using the following equation (valid only for $v_h\, t < r_0$):

$$Q(t) = \frac{q_0}{\ln(r_2/r_1)} \left[\ln\left(1 + \frac{v_e\, t}{r_0}\right) - \ln\left(1 - \frac{v_h\, t}{r_0}\right) \right] \tag{2.9}$$

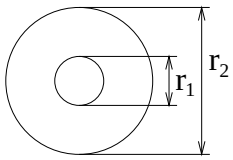


Figure 2.6: Definition of r_1 and r_2 for equation 2.9.

Like outlined in figure 2.6, r_1 and r_2 are the inner and outer radius of the crystal, v_e and v_h are the electron and hole drift velocities, q_0 is the maximum charge if all electrons and holes have been collected. The position where the charge carriers are formed is r_0. The first and second terms in the square brackets become constants [equal to $\ln(r_2/r_0)$ and $\ln(r_1/r_0)$] when the electrons or holes are collected, and a slope change occurs in the waveform at the corresponding times. Figure 2.5 shows the resulting output pulses for several different radial positions. It is important to notice that both calculation and figure are valid only assuming the following:

- All charge carriers are created at a fixed position. If multiple interactions (e.g., Compton scattering followed by photoelectric absorption) occur at different points in the detector crystal, the resulting total shape is obtained summing the shapes of the individual interactions.
- All charge carriers are assumed to be generated inside the active volume of the detector.
- The electric field is high enough to cause saturation of the drift velocity for both electrons and holes.

- Trapping and de trapping of the charge carriers are ignored.

The geometry of real detectors is however more complicated because, for reliability reasons, the inner hole is not going through the whole length of the cylindrical crystal. This so called closed-end configuration cannot be solved analytically but must be obtained with numerical methods. Figure 2.8 shows the pulses for interactions at different radii depths for a closed-end coaxial HPGe detector. These shapes have been calculated with programs developed at INFN Padova by Thorsten Kröll [12] using finite elements and weighting fields methods. As one can see the shapes are rather more complicated than those given before. Furthermore, the variation of the electric field strength depends on the purity of the Ge crystal, like shown in figure 2.7. The intrinsic crystal used to produce figure 2.5 has a high field strength in the inner region, falling down like $1/r$. The property of the field inside real HPGe crystals with an impurity concentration of 10^{10} atoms per cm^3 is opposite. This leads to the situation that the bottom part of figure 2.8 is not similar to figure 2.5 what one would suppose at first.

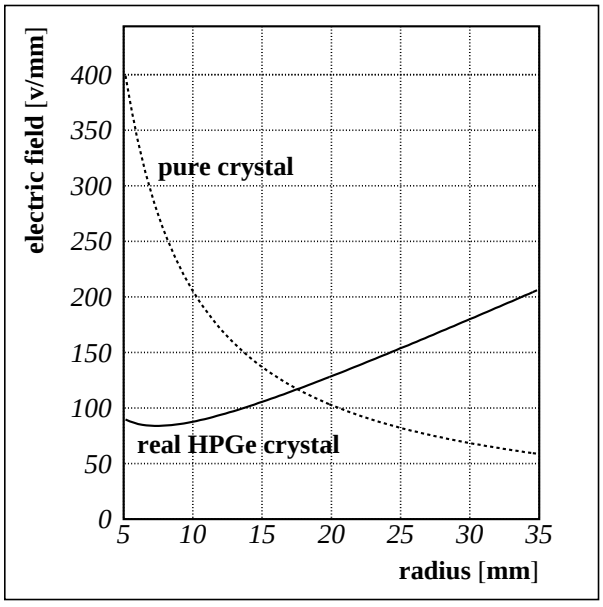


Figure 2.7: Electric field for a pure and a slight impure Ge coaxial crystal, calculated for $4000\,V$.

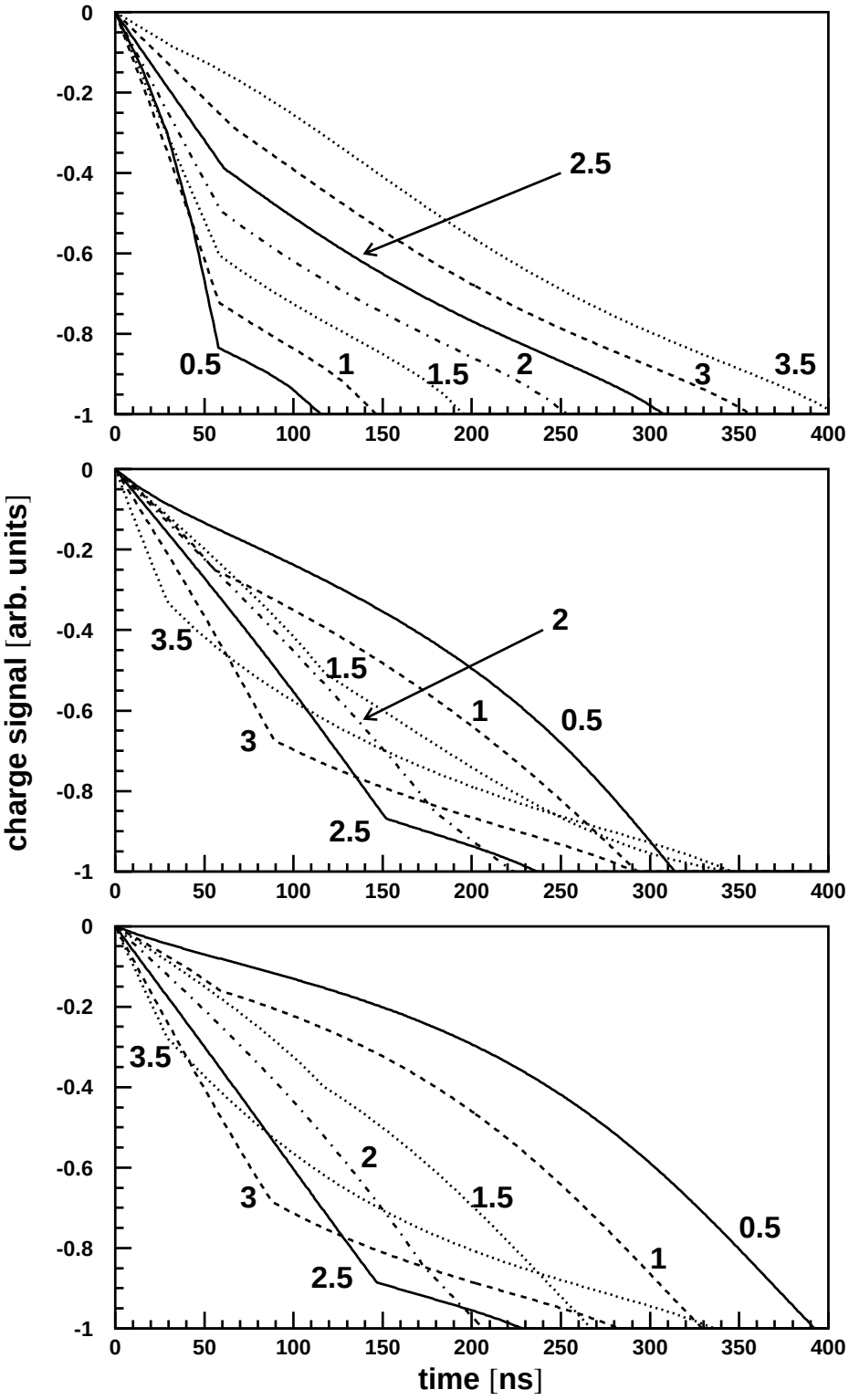


Figure 2.8: Simulated output pulses for a closed-end coaxial HPGe detector at different radial positions (in cm). Upper: in the closed front region. Middle: region of the beginning hole. Lower: center of the coaxial region.

Chapter 3

Deconvolution and solutions for our system

In every kind of measurement the input source signal is manipulated by the process of measurement which introduces some noise, thereby reducing the possibility to determine all features of the data under investigation. If the influence is really statistical, meaning "white noise", there is no possibility to calculate it away. In most cases however, many of the degrading operations are selective, in that their effects are more damaging to certain features of the data than to other. For example, the noise will be not really "white" because of the bandwidth of the used electronics. Furthermore, known dependencies on rise and delay times or amplitude of the signal can be used to construct a model of the manipulating operations. Once formulated in mathematical models the effects assist to purify the signal by calculating back the known manipulating terms. This kind of calculation is called *deconvolution*, and the following sections describe both the theory and the problems uncovered while programming the *MATLAB* [5] code for the present experiment.

Peter A. Jansson writes in [11] about the future: *"Linear deconvolution methods have served to educate us as to the pitfalls of the deconvolution problem. Their occasional successful applications both tantalised and discouraged us. ... The more-generally useful nonlinear methods have teamed with the powerful hardware that they demand to enhance future prospects for wide application of deconvolution methods. ... Computing time and cost will also be reduced through improvements in both algorithms and hardware. We anticipate a proper answer to the question, how far can we go? ... Future experimentalists will come to regard deconvolution as an essential tool in the standard repertoire. Through deconvolution they will make otherwise-expensive observations with inexpensive instruments. The limits of state-of-the-art experimental apparatus will be extended. This progress, in turn, will*

produce data that cannot fail to reveal phenomena that would otherwise lie hidden. The new experimental results will stimulate advances in physical theory, thereby providing incentives for better and better apparatus. Because of its applicability across disciplinary lines, deconvolution will play a key role in the advancement of all branches of science that yield to the tools of measurement."

3.1 The convolution integral

The transformation of the input signal with an operation producing a manipulated output signal can be described mathematically with the method of *convolution*. That is particularly useful if the operation is too difficult to describe with an analytical transfer function.

If the operation can be considered as a linear system with input $x(t)$ and output $y(t)$, it can be expressed as the limiting form of a linear combination of shifted pulses. From the mathematical point of view one deals with sums having an infinite number of terms, whereas for practical computations one takes only a finite number of combinations. The input is described as:

$$x(t) = \lim_{\Delta \to 0} \sum_{k=-\infty}^{+\infty} x(k)\, \delta(t - k\Delta)\, \Delta \tag{3.1}$$

If $\Delta \to 0$ the sum term becomes an integral.

$$x(t) = \int_{-\infty}^{+\infty} x(\tau)\, \delta(t - \tau)\, d\tau \tag{3.2}$$

The theory of Linear Time-Invariant (LTI) Systems defines $h_k(t) = h_\tau(t)$ as the response to a shifted unit impulse δ at time t, meaning $\delta(t - k\Delta)$ for the sum and $\delta(t - \tau)$ for the integral term. Because of the superposition property of linear terms it is allowed to change all the input unit impulses to output responses to get the whole output signal:

$$y(t) = \lim_{\Delta \to 0} \sum_{k=-\infty}^{+\infty} x(k)\, h_k(t)\, \Delta \tag{3.3}$$

Or again as an integral:

$$y(t) = \int_{-\infty}^{+\infty} x(\tau)\, h_\tau(t)\, d\tau \tag{3.4}$$

The last equation represents the general form of the response of a linear system in continuous time. If in addition to being linear the system is also time-invariant,

then $h_\tau(t) = h_0(t-\tau)$. For notational convenience the subscript is dropped and the *unit impulse response* $h(t)$ is defined as $h(t) = h_0(t)$. This results in the *convolution sum*:

$$y(t) = \sum_{k=-\infty}^{+\infty} x(k)\, h(t - k\Delta) = \sum_{k=-\infty}^{+\infty} h(k)\, x(t - k\Delta) \tag{3.5}$$

The corresponding *convolution integral* is:

$$y(t) = \int_{-\infty}^{+\infty} x(\tau)\, h(t-\tau)\, d\tau = \int_{-\infty}^{+\infty} h(\tau)\, x(t-\tau)\, d\tau \tag{3.6}$$

Two important annotations: The sum formula is now valid for each time probe t and it is not anymore necessary to form the limits. It is possible to take as much time probes t_i as needed for the one's own wishes. Secondly it is not anymore important whether the sum loop or integration loop is in the variable h or in x. Both right and left terms have the same results.

More detailed information about this derivation and LTI systems can be found in [10].

3.2 How to obtain the deconvolution

The next aim is to reach the inversion of the convolution, which is named *deconvolution*. The difficulty is that a simple algebraic transposition is impossible because of the product of the two functions $x(t)$ and $h(t)$ inside of the sum (or integral). However, it follows from the definition of the Fourier integral that:

> "The Fourier transform of the convolution of two functions is equal to the product of their individual Fourier transforms."

The uppercase letters in the following formulas mean the results after a Fourier transform (i.e. frequency domain) and lowercase letters mean terms before a Fourier transform or after an inverse transformation (i.e. time domain). In this manner it is possible to write the convolution integral or convolution sum as:

$$Y(s) = H(s)\, X(s) \tag{3.7}$$

Now, the Fourier transform of the deconvolution is simply:

$$X(s) = \frac{Y(s)}{H(s)} \tag{3.8}$$

That means: divide the Fourier transform of the known convolution by the Fourier transform of the response to get the Fourier transform of the deconvoluted signal. To

give a complete overview, the formulas of Fourier transform and inverse transform are included, both in sum and integral form:

$$F(s) = \lim_{\Delta \to 0} \sum_{t=-\infty}^{+\infty} f(t)\, e^{-2\pi j s t \Delta}\, \Delta \tag{3.9}$$

$$F(s) = \int_{-\infty}^{+\infty} f(t)\, e^{-2\pi j s t}\, dt \tag{3.10}$$

$$f(t) = \lim_{\Delta \to 0} \sum_{s=-\infty}^{+\infty} F(s)\, e^{+2\pi j s t \Delta}\, \Delta \tag{3.11}$$

$$f(t) = \int_{-\infty}^{+\infty} F(s)\, e^{+2\pi j s t}\, ds \tag{3.12}$$

To compute the deconvolution with electronic calculators, the sum versions with a finite amount of steps are used. For the experimental setup described in this work the program *MATLAB* is used to do this job.

3.3 Problems with the deconvolution

The deconvolution works well with artificial, completely noise-free "data". Unfortunately realistic signals with real noise tend to cause uncontrollable oscillations in the restored signal. For this reason it is important to make the signal as nice as possible before executing the deconvolving division formula.

In the first approach to the deconvolution, simulated pulses with added noise (taken from a real signal) were used. These pulses were first approximated by means of polynomials of degree ranging from 6 to 60. The resulting polynomial function has then been Fourier transformed. The original simulated data (without added noise) has been used as response function of the system and therefore also its Fourier transform has been calculated in order to obtain the deconvoluted signal (in the frequency domain) following equation 3.8. The inverse Fourier transform of this function should have given back a flat distribution representing ideally the residual noise after the polynomial approximation. However the obtained function was instead very much oscillating. This shows that the polynomial approximation alone is not sufficient for our purpose. We must exploit any a priori knowledge on the original signal to counter random and systematic influences on the data. This additional information allows a stricter filtering of experimental data by utilising a number of constraints:

- Prior knowledge of the physical limits to the values that the data may assume. For our electronics, the maximum frequency is 30 MHz and the rising edge of the input signal to be reconstructed is shorter than 500 ns, as can be seen for example in figures 4.15 to 4.17.

- If the amplitude of the signal at the end of the sampling interval is not the same as at the beginning the Fourier transform, which deals well with periodic signals, it interprets the difference as a sharp pulse. This problem is usually solved by means of a smooth curve extended across the end points. The data belonging to this extension are known as "noise". Given the fact that our signals have a rather flat top we preferred to extend them using their mirror shape. The signal used for the first tests of calculation is shown schematically in figure 3.1, where one can see also that flat regions must be added at the beginning and at the end to obtain a clearer deconvoluted signal without an overlap of the first harmonic. In this way we ensure equal values at the beginning and at the end of the sampling interval without adding noise. In a later version of the program the signal is differentiated, giving zero for the flat region at the end of the signal. In this way the mirroring is no more needed and only the flat regions must be added.

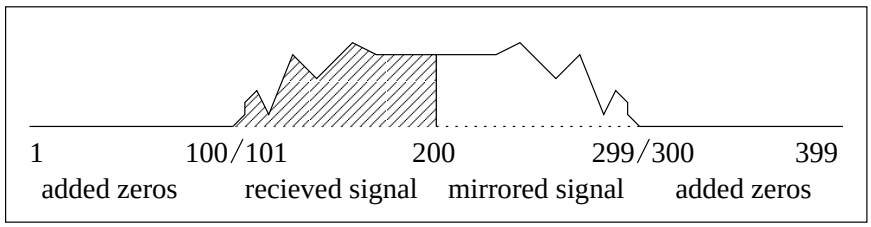


Figure 3.1: How to construct a symmetrical signal.

- Known symmetries of the output signal allow to set some terms of the Fourier spectrum to zero. In our case the mirroring of the signal causes a symmetry which allows only cosine terms in the spectrum.

- The constraint of "nonnegativity" says that the correctly restored function is not allowed to extend below a baseline (for example the zero line) where it would take non physical values. An arithmetical method to implement this constraint is described below. The actual data analysing program does not utilise this constraint because the following one is sufficient:

- The constraint of "finite extent" implies that no deviations from zero are allowed in those intervals on the spatial axis that lie outside the known extent of the original object. The implementation of the algorithm for this constraint into the program was the great break-through in reconstructing the input signals of the experiment.

For the last two constraints Samuel J. Howard [11] has developed mathematical solutions as described below.

3.4 The constraint of finite extent

If data exist only over a finite interval and are zero outside of it, the constraint of "finite extent" can be applied. At first the function to be restored is divided into two parts, $u(t)$ and $v(t)$: $u(t)$ refers to a low-pass filtered signal which is defined as "good" and whose Fourier transform is held fixed; $v(t)$ contains the rest of the signal which is defined as "bad" and whose values for the different time probes t are considered as independent variables. $N1$ and $N2$ are the boundaries of the nonzero region. The restored function $f(t) = u(t) + v(t)$ should have no deviations from zero outside of the interval given by these boundaries. Minimising the sum of the squared points outside the known interval result in $v(t)$ which best satisfy the constraint. Actually, recovering only a band of frequencies in $v(t)$ implies the additional constraint of holding all frequencies above this band equal to zero (what is a low pass filter). As a big advantage in opposition to directly setting to zero a range of frequencies, this method does not deform the signal inside the signal's boundary. Because of aliasing, the total number of independent coefficients should not be greater than the total set of data N. This means that the highest frequencies have to be set to zero without calculating the effect of this operation on the signal components $v(t)$, because for each signal probe in the time domain there exist two Fourier coefficients (sin and cos) in the frequency domain. In a linear system of equations there should be as many unknown variables as equations, and like described below there is one equation for each signal component of $v(t)$. The expression to be minimised is:

$$\sum_{t<N1,\, t>N2} [u(t) + v(t)]^2 = \sum_{t<N1,\, t>N2} \left\{ u(t) + A_b \cos\left[\frac{2\pi}{N} bt\right] + B_b \sin\left[\frac{2\pi}{N} bt\right] + A_{b+1} \cos\left[\frac{2\pi}{N}(b+1)t\right] + \cdots + B_{b+c-1} \sin\left[\frac{2\pi}{N}(b+c-1)t\right] \right\}^2 \tag{3.13}$$

To find the minimum of a function of several variables one takes the partial derivative with respect to each unknown coefficient and sets the result to zero. This gives the

system:

$$\begin{aligned}
\frac{\partial}{\partial A_b} \sum_{t<N1,\, t>N2} [u(t)+v(t)]^2 &= 0 \\
\frac{\partial}{\partial B_b} \sum_{t<N1,\, t>N2} [u(t)+v(t)]^2 &= 0 \\
\frac{\partial}{\partial A_{b+1}} \sum_{t<N1,\, t>N2} [u(t)+v(t)]^2 &= 0 \\
&\vdots \\
\frac{\partial}{\partial B_{b+c-1}} \sum_{t<N1,\, t>N2} [u(t)+v(t)]^2 &= 0
\end{aligned} \tag{3.14}$$

Considering in detail the derivative with respect to A_b it is solved like this:

$$\begin{aligned}
&\sum_{t<N1,\, t>N2} \frac{\partial}{\partial A_b} \left[u(t) + A_b \cos\left(\frac{2\pi}{N} bt\right) + \cdots \right]^2 \\
&= 2 \sum_{t<N1,\, t>N2} [u(t)+v(t)] \cos\left(\frac{2\pi}{N} bt\right) = 0
\end{aligned} \tag{3.15}$$

The complete set of equations to be solved for all the c chosen k values outside of the interval $N1$ and $N2$ is enumerated below.

$$\begin{aligned}
\sum_{t<N1,\, t>N2} [u(t)+v(t)] \cos\left[\frac{2\pi}{N} bt\right] &= 0 \\
\sum_{t<N1,\, t>N2} [u(t)+v(t)] \sin\left[\frac{2\pi}{N} bt\right] &= 0 \\
\sum_{t<N1,\, t>N2} [u(t)+v(t)] \cos\left[\frac{2\pi}{N} (b+1)t\right] &= 0 \\
&\vdots \\
\sum_{t<N1,\, t>N2} [u(t)+v(t)] \sin\left[\frac{2\pi}{N} (b+c-1)t\right] &= 0
\end{aligned} \tag{3.16}$$

It is a set of linear equations in the independent coefficients $v(t)$. Once the coefficients have been obtained, they are Fourier transformed and the resulting Fourier coefficients $F(b)$ to $F(b+c-1)$, with $F(s) = A_s + B_s$, are added to the lower ones taken from $u(t)$. Finally the inverse Fourier transform of the whole set $F(s)$ restores the signal.

If some Fourier coefficients of the signal response function have very small or zero values for certain frequencies s, it is important to set them a little bit away

from zero. Otherwise some very large values might be artificially constructed in the high-frequency band as the algorithm divides by something very small (or zero, resulting in an error).

3.5 The constraint of nonnegativity

This constraint is not included in the algorithm of this work because of the following two reasons: The part inside of the signal to be restored is far away from zero for all probes f(t). Therefore, if the deconvoluted signal becomes negative in the region where the original signal is far from zero, the result is too far away from reality to be taken in consideration, as one can see for example in figures 4.15 to 4.17. Several tests performed even with bad looking deconvolutions showed that once a non-oscillating solution is found it is unlikely to obtain negative values in the deconvoluted function. However, outside the time range of the deconvoluted signal but still inside the total considered time window, negative values occur. The constraint of nonnegativity is solved in an iterative method and therefore does not exploit well the ability of *MATLAB* to solve linear equations. On the other side, using only the constraint of finite extent produces up to 100 Fourier coefficients (in the actual used program version) with those of the highest frequencies corresponding to noise for any realistic signal. For this reason the implemented algorithm searches for the maximum value to find the signal and cuts the left and right sides after the first points of intersection with zero.

In practical applications with pulses produced by radioactive processes one does not deal with single signals but rather with time sequences of signals having, usually, a random time distribution. This means that it is not possible to use the constraint of finite extent between two signals because the time location of the individual ones is not known, and therefore one cannot define the time intervals (between the individual pulses) where the signal should be zero. In contrast with this it is sure that all values should not be negative as we are dealing with collected charge signals which are in principle of the same polarity. (If they are all negative, the constraint becomes one of nonpositivity.) In such a case it is more useful to add the constraint of nonnegativity to the constraint of finite extent. Using both constraints means having more independent variables because there are more signal probes $v(t)$ to calculate with, resulting in more Fourier coefficients to be taken into consideration in the system of equations.

The way to derive the constraint of nonnegativity is much longer than the last one, so in this section only the solution is mentioned. The whole derivation can be

found in [11]. The resulting system of equations is the following:

$$\begin{aligned}
A_b &= -\sum_{t=0}^{N-1}\left\{u(t)+v(t)-A_b\cos\left[\frac{2\pi}{N}bt\right]\right\} \\
&\times \cos\left[\frac{2\pi}{N}bt\right]\left\{\sum_{t=0}^{N-1}\cos^2\left[\frac{2\pi}{N}bt\right]\right\}^{-1} \\
B_b &= -\sum_{t=0}^{N-1}\left\{u(t)+v(t)-B_b\sin\left[\frac{2\pi}{N}bt\right]\right\} \\
&\times \sin\left[\frac{2\pi}{N}bt\right]\left\{\sum_{t=0}^{N-1}\sin^2\left[\frac{2\pi}{N}bt\right]\right\}^{-1} \\
A_{b+1} &= -\sum_{t=0}^{N-1}\left\{u(t)+v(t)-A_{b+1}\cos\left[\frac{2\pi}{N}(b+1)t\right]\right\} \\
&\times \cos\left[\frac{2\pi}{N}(b+1)t\right]\left\{\sum_{t=0}^{N-1}\cos^2\left[\frac{2\pi}{N}(b+1)t\right]\right\}^{-1} \\
&\vdots \\
B_{b+c-1} &= -\sum_{t=0}^{N-1}\left\{u(t)+v(t)-B_{b+c-1}\sin\left[\frac{2\pi}{N}(b+c-1)t\right]\right\} \\
&\times \sin\left[\frac{2\pi}{N}(b+c-1)t\right]\left\{\sum_{t=0}^{N-1}\sin^2\left[\frac{2\pi}{N}(b+c-1)t\right]\right\}^{-1}
\end{aligned} \tag{3.17}$$

The meaning of the letters for the constants and coefficients is the same as in the last section about the constraint of finite extent. One can see that the summation interval is written over the whole distance of the N data under investigation. This is not correct because the summation has to be taken only over those data points t for which the function $u(t)+v(t)$ is negative. That means that the algorithm has to test $u(t)+v(t)$ for negative values after each new iteration. A matrix solving method like the Gauss-Jordan reduction cannot be used without modification because for each iteration step the test might choose other data points to be included, meaning other entries in the matrix to be solved. One has to program an iterative algorithm.

Finally it is found in practice by Howard [11] that all that is necessary to implement successfully both constraints on the same set of data is to take the summation over all data points outside the known boundaries for the constraint of finite extent while summing the negative values only over the inner interval for the constraint of nonnegativity. This should be done in each iteration cycle for both constraints together to allow a proper test on the negativity mentioned above.

Chapter 4

Overview about the experimental setup

As explained in the previous chapter, the shape of the rising edge of the pulse produced at the electrodes of the detector depends on the position of the interaction. It is therefore, at least in principle, possible to use the obtained shape to derive the position of the interaction.

However, one must consider that real pulses are always obtained from the output of charge-preamplifiers which will also contribute, through their transfer function, to the shape of the signal. The response function of the preamplifier must be taken into account and the experimental shapes must be deconvoluted before comparing them with the calculated ones.

The prototype detector of the *MARS* experiment is equipped with 26 charge preamplifiers developed at the university of Köln and we have calculated the response function using the *Spice* simulation program.

The other part to be considered is how the shape of the signal can be collected and put in a form suited for being processed in a computer. This is usually done digitising the pulses by means of an ADC with a sufficient number of bits and a high enough sampling frequency to preserve the information contained in the pulse. For this purpose we have used a system from *Struck* which can digitise up to 96 channels in parallel in a crate. The ADC's have 8 bits and can sample the input signals every 10 ns.

4.1 The used Ge detector

For a test of the programs to control the experiment and to deconvolute the signals it was decided not to wait for the prototype detector but test it with one of the *ORTEC*

detectors used in the *GASP* experiment. These detectors are not segmented but the actual used equipment and programs are set up to support up to 96 segments, enough for any reasonable future prototype.

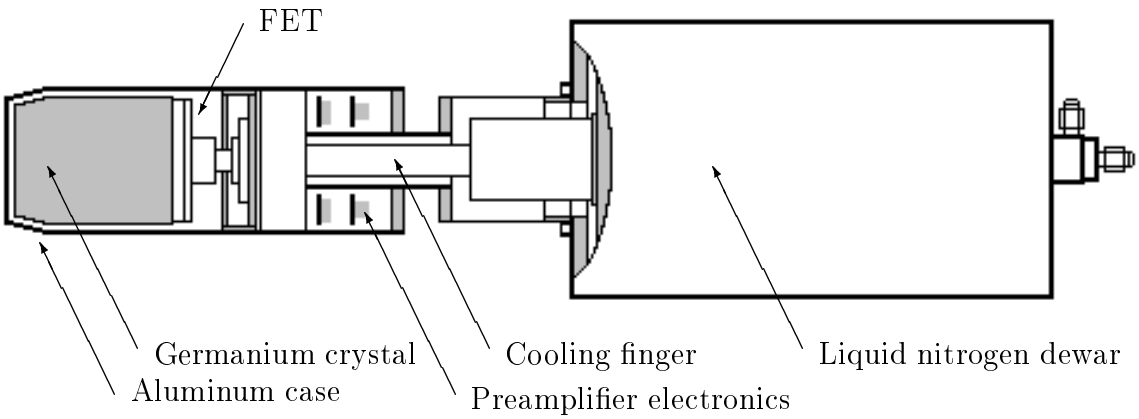


Figure 4.1: Diagram of the *ORTEC* HP*Ge* detector setup.

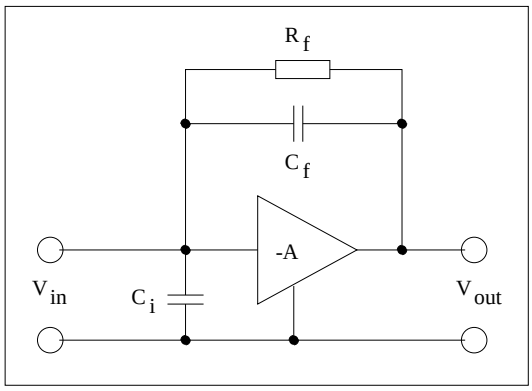
Figure 4.1, designed using a drawing from the *GASP* internet site (http://axpd30.pd.infn.it/GASP/pictures/HPGe.jpg), shows the configuration of the used *ORTEC* detector (Model No. *GMX-76240-S*) with the liquid nitrogen (LN_2) dewar and the preamplifier. The detector is very compact in order to be inserted into the Anticompton shield of a 4π γ-ray array. The electronics of the preamplifier is located very close to the crystal to minimise capacitance and to provide a cooling of the input field-effect-transistors, so to reduce the electronic noise.

4.2 Charge-sensitive preamplifiers

As described in a previous chapter the output of pulse-type radiation detectors like germanium crystals, is a burst of charges. The amount of charges in most cases is so small, that a sufficient voltage to amplify them with an acceptable signal-to-noise ratio is achievable only with a high impedance input of the preamplifier. For the same reason it is also necessary to minimise the capacitive loading, for example by minimising the cable length from the detector to the preamplifier. Concerning the noise performance of a detector system the preamplifier is the most critical piece of the electronics used to process the signals. Preamplifiers exist in two basic configurations: voltage-sensitive and charge-sensitive.

The voltage-sensitive preamplifier provides an output voltage proportional to the input voltage as long as the input capacitance is not changed. Unfortunately,

in semiconductor detectors the capacitance can change by changing the operating parameters.



Assume $A >> (C_i + C_f)/C_f$

$$V_{out} = -A\, V_{in}$$

$$V_{out} = -A\, \frac{Q}{C_i + (A+1)\, C_f}$$

$$V_{out} \cong -\frac{Q}{C_f}$$

Figure 4.2: Principle of a charge-sensitive preamplifier.

For this reason, as preamplifier for germanium detectors the charge-sensitive configuration is almost always chosen. Figure 4.2 shows the principle design and the significant equations of a charge-sensitive preamplifier. For this circuit, the output voltage is proportional to the total integrated amount of charge seen by the input terminals. Changes in the input capacitance have no longer an effect on the output voltage, as long as the input pulse is short compared with the time constant (decay rate) $R_f\, C_f$. Of course, the input capacitance C_i, shown in the figure, is not a part of the preamplifier but is due to the detector and the cable connecting it to the input field-effect-transistor.

To obtain the best resolution, the constant $\tau = R_f\, C_f$ has to be chosen as large as possible. In the ideal case of $R_f \to \infty$ the response function of the preamplifier is a step-like function which for a given pulse builds on the level produced by the preceding signals. Of course after a certain time the output of the preamplifier saturates and must be resented by a proper circuitry. (This configuration is known as *transistor reset* charge sensitive preamplifier.)

In practical *resistive feedback* preamplifiers the time constant is of the order of $1\, ms$ and is obtained with $R_f \approx 1\, G\Omega$ and $C_f \approx 1\, pF$. The gain voltage can be easily calculated knowing the energy required to produce one charge carrier: for germanium it is $\approx 55\, mV/(MeV \cdot pF)$.

The long output tail which is needed to obtain good energy resolution is usually removed by a second stage of the preamplifier through a differentiation giving a decay constant of roughly $50\, \mu s$. This stage contains also a zero-pole cancellation circuit and is used to decouple the output from the first "delicate" part of the preamplifier.

The "ideal" response function of a resistive feedback preamplifier is an exponential with zero rise time which does not perturb the time distribution of the input charges. In practice however the response function has a finite rise time (due to parasitic capacitance) and this means that the shape of the output signal is no more exactly similar to the input.

4.2.1 The preamplifier of Köln

The preamplifier for the 25 outer segments of the *MARS* prototype detector has been developed by George Pascovici of the *Institut für Kernphysik* at the *Universität zu Köln*. It is known as *SMD-PA* and is a very compact, low power, low noise, high speed preamplifier, specifically designed for pulse shape analysis using cooled detectors. The preamplifier includes a low noise FET input circuit placed in close contact with the *Ge* crystal (so called *cooled FET version*). Alternatively it is possible to place the FET directly on the *SMD-PA* main circuit board (so called *room temperature FET version*).

The sheet of the SMD-PA circuit is shown in figure 4.3. All references to pins, connectors and electronic components made in this section can be found in the figure. The first *cold part* contains the FET transistor and makes the first amplification at high impedance. After this, a stabilised complementary transistor amplification circuit placed on the circuit board changes the impedance to low. The output of this part is fed back to the input to close the amplification loop and stabilise the circuit. The third step in the signal path is a standard operational amplifier that can be used either in inverting or in non-inverting mode according to the desired polarity of the output signal. This amplifier is used to decouple the output from the first amplification loop, followed by another one to produce the proper output shape through a differentiation and pole zero cancellation.

In the case of highly segmented *Ge* detectors the preamplifier can be connected both to the inner and to the outer (segmented) electrodes. Of course, lowest noise and best rise time performance are achieved with the cooled FET version. In this case, the cooled FET circuit, placed in the detector cryostat at a temperature of about $150\,K$, is connected with the SMD-PA main circuit board by a 5 wire, shielded, flat cable assembly.

Adjustments:

P1 is used to optimise the FET drain current for a minimum noise level. P2 is used to optimise the conversion of the $1\,ms$ charge-sensitive stage decay time into $47.5\,\mu s$ decay time with minimum overshoots, or respectively undershoots. P3 is used to adjust the DC offset between $\pm 100\,mV$ (unterminated). Cv is used to

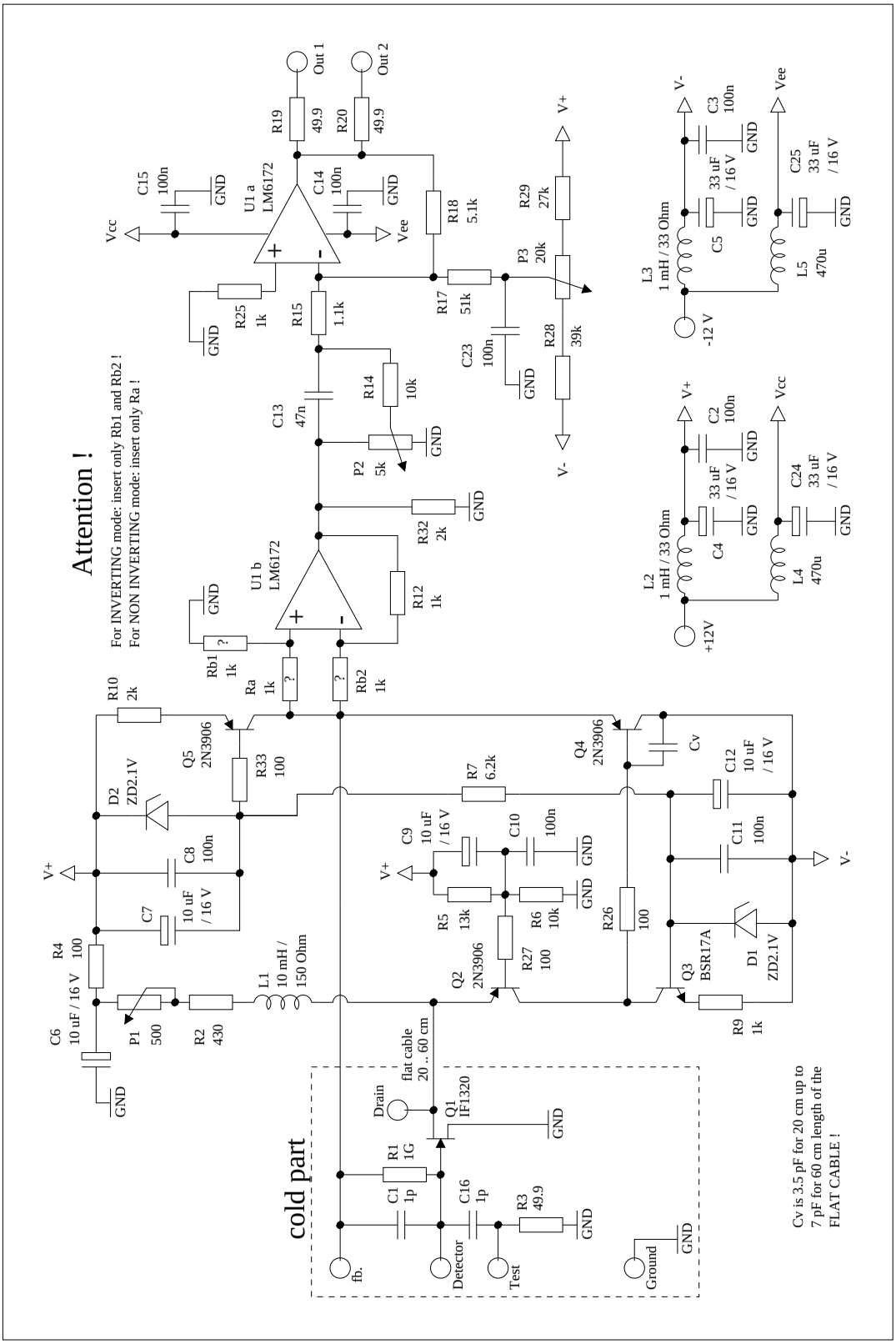


Figure 4.3: Sheet of the SMD-PA preamplifier.

optimise the rise time performance to get the minimum rise time without overshoots (undershoots), depending on the flat cable length.

Inputs:

The charge pulses from the detector are connected to the *Detector* input which has a high source impedance. The *Test* input has an impedance of $50\,\Omega$ and can be used to test the preamplifier and the following electronic circuitry by means of pulse generated signals.

Outputs:

Two independent outputs to be used as *energy* and *timing* signals with peak amplitudes linearly proportional to the charge input.

Performance:

The rise time is $17\,ns$ at $0\,pF$ with a slope of about $0.3\,ns/pF$. Additional rise time is caused by the flat cable length and is $5\,ns$ for $50\,cm$. The decay time is $47.5\,\mu s$, but can be factory adjusted to different values. The impedance is $50\,\Omega$ with a maximum output swing of $83\,mA$. The gain is $175\,mV/MeV$ unterminated. Also this is factory adjustable, up to $200\,mV/MeV$. The circuit integrates the input signal with a nonlinearity of less then $0.025\,\%$ for an output swing of $9.5\,V$ unterminated. Using analog amplifiers of good quality (like *Ortec 671* or *Silena 6711*) with 3 to $4\,\mu s$ shaping constant, the noise in equivalent FWHM (that is *Full Width of Half Maximum*) is less than $0.6\,keV$ at $0\,pF$ and less than $1.0\,keV$ at $30\,pF$. The noise caused by this preamplifier is more than 10 times lower than the noise caused by the digitisation in the *DL 312* module (described later), it is not important to consider it in the actual experiment. The power consumption of one *SMD-PA* is only $0.42\,W$. The physical dimensions are very small ($26 \times 50 \times 9\,mm$) and this is a big advantage for highly segmented counters, because each segment needs its own preamplifier.

4.2.2 The transfer function

The transfer function of the *SMD-PA* preamplifier has been obtained with the program *Spice* and is shown in figure 4.4 as solid lines. After models are defined for all semiconductor components and a net plan of all sheet circuits and connections of the preamplifier is designed, it is possible to simulate the output signal for each wanted input signal. The *Spice* program originally was manufactured to develop integrated circuits themselves. For our discrete circuit it was quite difficult to find proper models for all the used semiconductors and not all of them are provided by the manufacturers of the semiconductors or their representatives. Also some large resistors (roughly $1\,G\Omega$) and small capacitors (roughly $1\,pF$) had to be added into the received models to reach stable simulation runs without premature program

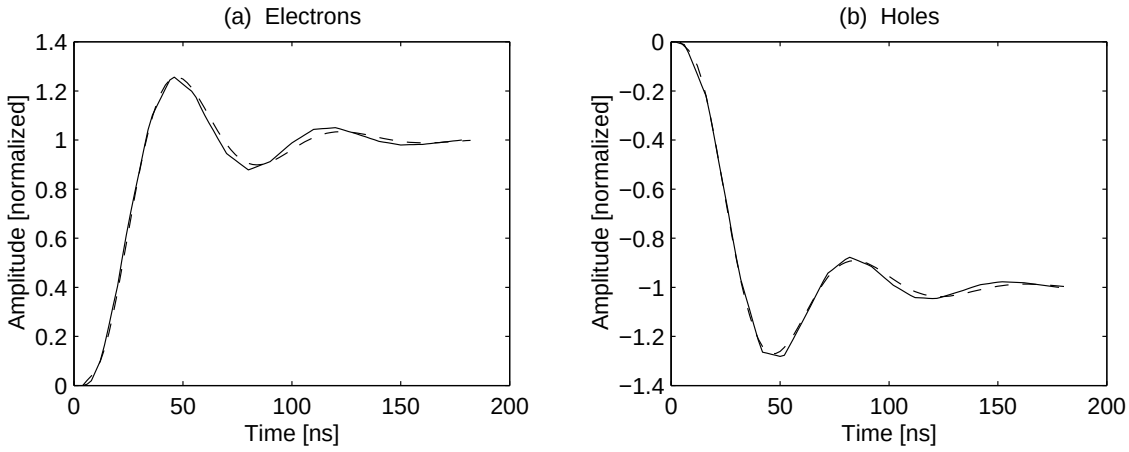


Figure 4.4: *SMD-PA*: Simulated impulse response (solid line) and fit (dashed line).

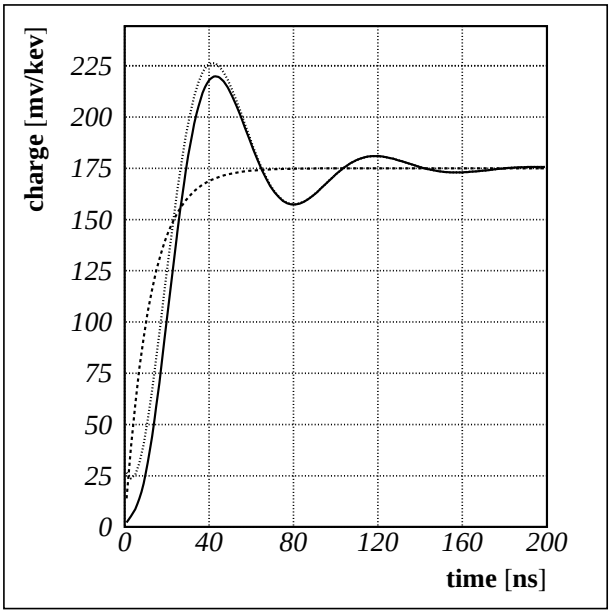


Figure 4.5: A model for the impulse response function.

aborts. The places where to put these components were found by looking through the core dump file created by *Spice*. Typically it was possible to find node points with impossible voltage values. In the electronic elements connected to these points, we found frequently a transistor with zero capacity between the base, collector and emitter contacts. Also some capacitors were problematic: The simulation run places necessarily an initial voltage to every node point and afterwards tries to find a static voltage distribution by solving the system of equations which belongs to the total circuit under investigation. If a transistor is switched off and is connected only to a capacitor or to another transistor, the charges cannot move from the corresponding node point. Inside of integrated circuits this situation is very common. In reality the problem does not exist because also capacitors and switched-off semiconductors have a resistance less than infinite. To avoid the described problems the *Spice* program uses a very low capacity to short cut the semiconductor connections and a very large resistor to short cut the capacitors, also if there are no such elements in the description file of the circuit. Setting these general limiting constants to lower (for resistors) and higher (for capacitors) values as the default setting produces a stable solution. This is recommended in the *Spice* manual [4] to solve such a kind of problem. Unfortunately, a stable simulation run was reached only by setting the limits near to the values mentioned above, which falsifies the circuit too much. For example the feedback loop in the first part of the circuit ($R_1 = 1\,G\Omega$ and $C_1 = 1\,pF$) sets the decay rate and this rate would change a lot by adding another $R = 1\,G\Omega$. For this reason the critical points causing errors were searched by hand to add capacitors or resistors selectively only in the places, where they were needed to enforce stable simulation runs. Whereas there were few problems with input of realistic impulses, it was quite difficult to reach a stable program execution for short pulses, needed to approximate the δ unit function. The modifications were not made in time-critical places of the sheet, therefore a measurable modification of the impulse response function is not expected. To be sure, if the final version of these preamplifiers will be mounted in the prototype detector, one could repeat the simulation runs with the modified sheet and compare them with the behaviour of a (spare) real one, of course not using a nearly delta spike but using a pulser signal. Such a test could also decide if the delivered models for "transistor types similar to the used ones" are sufficient to describe the real timing. In detail only the model of the *LM6172* operational amplifier was received by the manufacturer. *Philips* wrote that they do not have *Spice* models for the used transistors. *InterFET* sent, after three months of waiting, some advertisement for a new integrated charge preamplifier with a *Spice* model of that one instead of the used FET's. No model was found for the *BSR17A*

transistor, it was replaced by a *LM3904* which should be rather similar.

For the simulation of a δ unit function a narrow triangular pulse of 300.000 electrons (corresponding roughly to the amount of charge released by the detection of a $1\,MeV$ gamma ray in germanium) with a rising and a falling edge of each $0.1\,ns$ has been used. This is likely to be a good approximation because the input signal is at least one order of magnitude faster than needed by the following electronics. The same simulation is repeated for positive charges because the behaviour of the preamplifier does not need to be symmetrical. The output of the simulation can be used as impulse response $h(t)$ like described later.

In order to be able to perform the deconvolution process in a simple manner, we preferred not to directly use the transfer function derived with *Spice* because it is expressed in a form, which is not easily adaptable to different preamplifiers or even to the same preamplifier with slightly different components. In fact, for any even slight modification of the preamplifier circuit one has to introduce the full specification of the circuit and perform a full calculation with the problems described above. Instead we tried to look for a simple analytical expression with a few free parameters which can be easily modified to describe different situations. The obtained expression is not a rigorous model of the transfer function but expresses in an intuitive way the main effects producing the non ideal response to the δ unit function. The first important effect is that it takes a certain time to charge the always existing parasitic capacitance in the circuit. This is represented as a dashed line on figure 4.5 and can be described as:

$$y_1(t) = Y_0 \left(1 - e^{-t/\tau}\right) \tag{4.1}$$

Y_0 is the final value after $t \to \infty$ and τ is the time constant of how fast the capacitance is charged.

The second effect is that changing from one level to another (here from 0 to Y_0) in a circuit with a feed-back loop always causes oscillation. Sometimes it does not seem so because the oscillation is very much damped, but if a fast reaction is wanted one must choose a good compromise between velocity and overshoot. This oscillation is represented as a dotted line in figure 4.5 and can be described as:

$$y_2(t) = 1 + e^{-\delta t} \sin\left(\frac{2\pi}{T}\, t + \varphi_0 - \frac{\pi}{2}\right) \tag{4.2}$$

δ is the decay time coefficient, T the period of the oscillation and φ_0 is the phase angle. (To substract $\pi/2$ could be avoided by changing the program source code, there are no special reasons.)

Normally a time lag between the input and the reaction is the third important effect in the behaviour of amplifiers. In this special case the time lag is so small that

it can be taken into account by adjusting the phase angle so that for the first time the positive increase of y_1 and the negative increase of y_2 neutralise each other, a construction that can be done by a computer program automatically. This results in only three free parameters to be adjusted, one of which, T, can be measured in a simple way and does not affect the others.

The final function is obtained multiplying the two expressions: $y(t) = y_1(t)\, y_2(t)$. A comparison of the resulting function with the result obtained with *Spice* is shown in figure 4.4. As one can see, the difference is very small, and therefore in the analysis of the data we preferred using the "analytical" transfer function. The advantage of this method is that it is possible to adjust the impulse response if the real preamplifier differs a bit from the simulated one. It is possible to use the program even with preamplifiers whose impulse responses are unknown at first if one tries out the proper values for τ, δ and T. The following table shows the best values for the simulated *SMD-PA* preamplifier:

	τ	δ	T	(φ_0)
negative charges	$1.2 \cdot 10^{-8}\, s$	$2.8 \cdot 10^{7}\, 1/s$	$7.7 \cdot 10^{-8}\, s$	(5.691)
positive charges	$1.2 \cdot 10^{-8}\, s$	$2.7 \cdot 10^{7}\, 1/s$	$7.8 \cdot 10^{-8}\, s$	(5.764)

In the paper [9] is derived a very detailed mathematical model to simulate the impulse response of preamplifiers, including unwanted effects like noise. The result is almost the same, taking in consideration the "low" time resolution of $10\, ns$ of our sampling ADC system. For the calculations made by the *MATLAB* program for the present work, the above described model is sufficient.

4.3 General description of the equipment

The electronic equipment used to digitise the signals from the detector is described in some detail in the following paragraphs. It consists, roughly, of three subsystems:

- The Trigger which selects the signals to be processed.
- The ADCs which digitise the signals.
- The data acquisition (*DAQ*) which reads the digitised data and stores it into files on a disk of a connected computer.

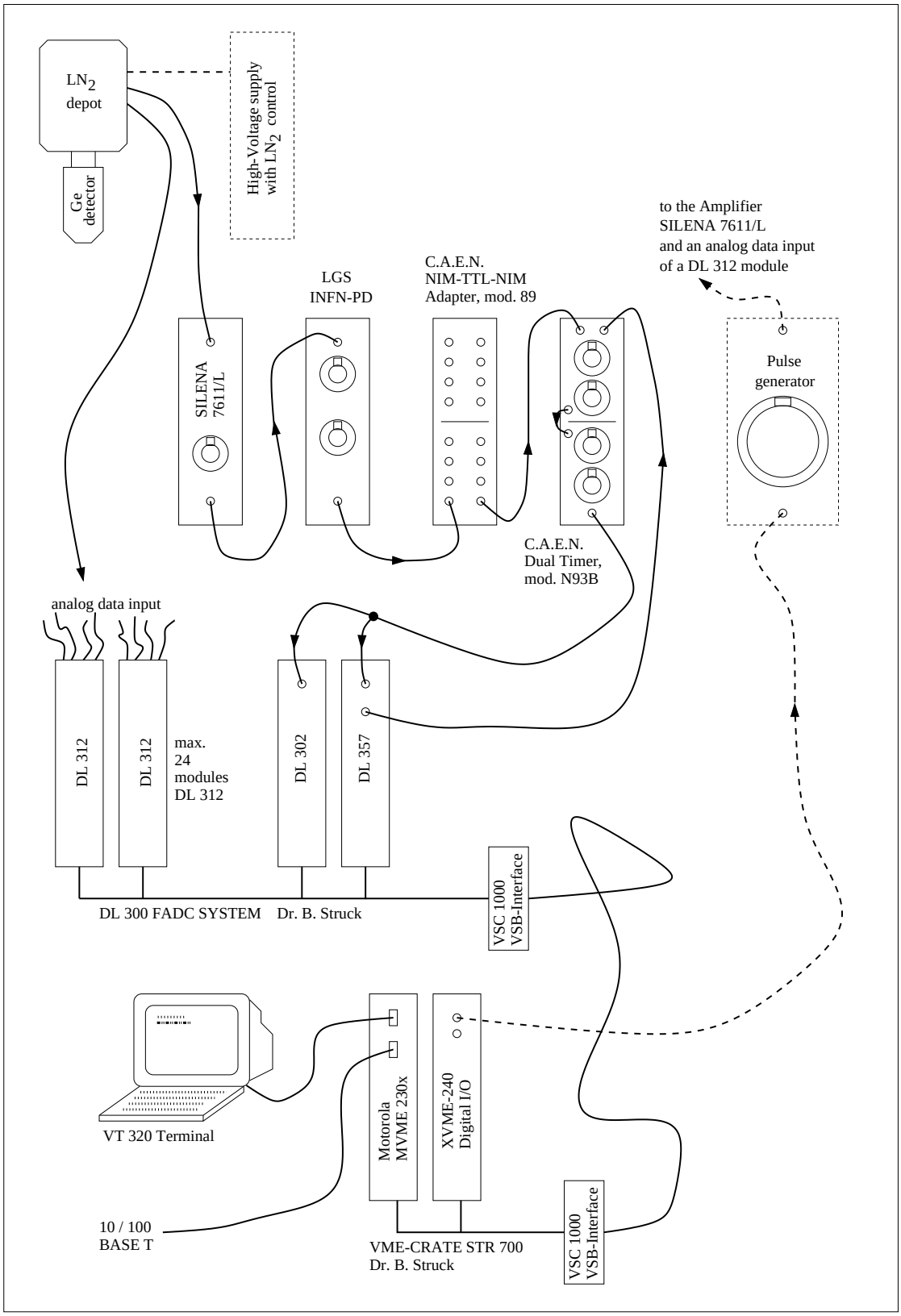


Figure 4.6: The signal path through the used equipment.

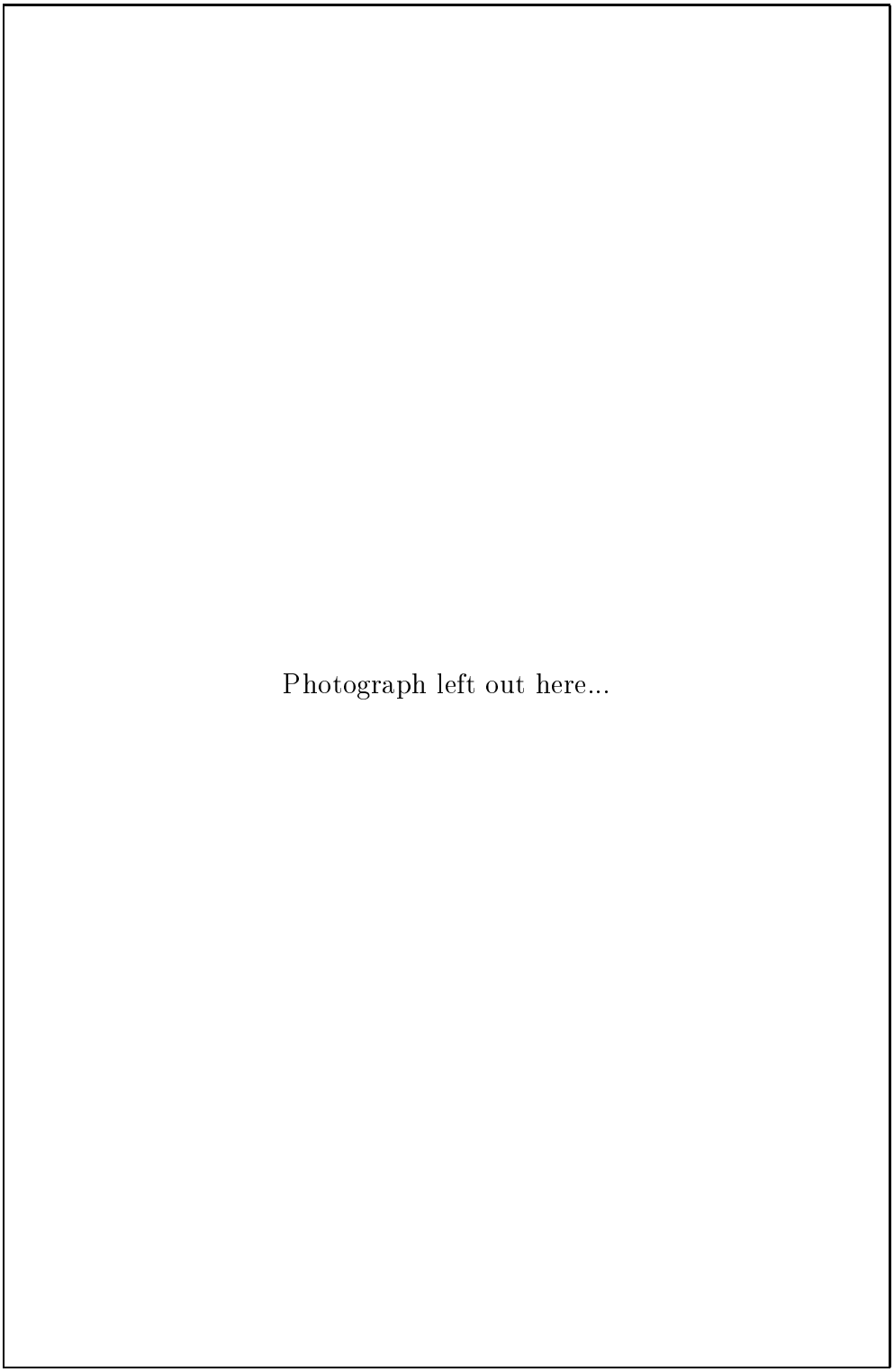


Figure 4.7: A photograph of the used equipment.

Figure 4.6 shows schematically the connections between the various modules; only elements relevant to this experiment are given. As one can see on the photograph 4.7, the set-up is rather complicated due to the fact that it was assembled from other experiments: a dedicated set-up would be more expensive than the prototype detector itself!

4.4 The Trigger subsystem

In any experiment, the trigger is designed to select which among the data produced by the detectors are interesting and should be taken into the data acquisition system.

In our case, the most simple trigger would be a signal, of a given amplitude, generated in the central (unsegmented) electrode of the tested detector. In fact a signal is always present in the central electrode independent of which of the segments fire. To detect the presence of the signal and, possibly, to select its energy, conventional analog electronic is used. The signal from the central electrode is connected to a spectroscopy amplifier (*SILENA, 7611/L*), used with a short shaping time. The output of the amplifier is sent to a linear gate circuit (*LGS INFN-PD*) which selects an energy range and provides a logic *TTL* pulse when a signal is present. This logic pulse is converted to the *NIM* standard (*C.A.E.N. NIM-TTL-NIM adapter, mod. 89*) and sent to a dual timer module (*C.A.E.N., mod. N93B*) where it is adjusted in time and width to provide the trigger pulse to the following ADC system. As the trigger should be inhibited while the ADC system is not ready to convert data, it is vetoed in the first channel of the dual timer by the busy signal from the *DL 357* module described below. The shaping time used in the amplifier ($0.5\,\mu s$) is less than optimum concerning the energy resolution, this is done to provide the trigger signal before the rise time of the pulse is shifted out of the 1024 byte long memory buffer of the ADC System.

4.5 The ADC subsystem

The output of the preamplifiers of the 25 segments are digitised by the *Struck* ADC subsystem described below. When used with non segmented detectors a second output of the preamplifier of the inner contact is connected to this system.

4.5.1 The *DL 312* modules

The DL 312 is a multi-channel wave form digitiser module based upon the AD 9002 Analog Device Flash-ADC chip. Each Flash-ADC has a resolution of 8 bit that means a need of 256 operational amplifiers to compare the input signal with the 256 reference voltages. (Of course there are not 256 separate amplifiers, they are integrated in special chips.) The module is especially designed to digitise and to store drift chamber signals. The band width is better than $30\,MHz$, the sampling rate up to 100 Mega samples per second (Ms/s). To increase the dynamic range up to 10 bits a non-linear transfer function is used. It is hyperbolic with the *a factor* 0.75. To re-linearise the signal the following formula is required:

$$U_i = \frac{C}{2^n - a\,C}\,U_0 \tag{4.3}$$

$a = 0.75$, $n = 8$ (8 bit ADC), C is the ADC output, U_i the input signal and finally U_0 is the calibration constant. To calculate U_0 it is possible to look in the Technical Manual but it is better to calibrate it directly, using a well-known input voltage. The range of input voltages is from zero to $200\,mV$ with an impedance of $100\,\Omega$. The advantage of the hyperbolic method is to provide a good resolution for higher voltage values and a bad resolution for low ones which are normally dominated by noise. An important fact to understand the programming reported later is that two channels are always read out in parallel because of the 16 bit *SCANBUS* of the system. If the number of used channels is odd the system seems to have a ghost channel which might have also ghost hits because of electronic noise. The program *sim2mat* is able to eliminate them.

Because of mechanical reasons it is possible to include up to 24 *DL 312* modules into the crate. Each module supports 4 digitising channels, so the maximum amount of channels is 96. In the preliminary experiment only one channel is used, but the software is developed to support all channels, for example the 26 ones needed for the prototype detector of *Eurisys*.

4.5.2 The *DL 302* module

The *DL 302* module is the central scanner controller of all the installed *DL 312* modules. The board contains the scanning circuitry and, in a pluggable "*personality board*", it could contain also an on-board hit-detector module. The version used here has no such module and does not need it because the *DL 357* module (see there) already manages the hit control. Essentially, the scanner consists of a programmable clock and a 16 bit address counter. After a start command, the scanner asserts

the system clock as well as address information for the data acquisition modules onto the digital *SCANBUS*. Because of the missing hit-detector the scanner can be operated only in the *Sampling Mode*: A signal selects all *DL 312* modules equally. These modules sample and digitise analog signals as described above and store them into their memories. The system clock is used as the sampling time base. It is possible to start scanning with automatic stop after the memory (1024 byte per channel) is full or, as programmed here, to start by software and continue until a stop signal is received (*Common Stop Mode*). The clock speed is adjustable to extern, $6.25\,MHz$, $12.5\,MHz$, $50\,MHz$ and $100\,MHz$. For pulse shape analysis performed in the experiments reported here $100\,MHz$ are necessary by all means.

4.5.3 The *DL 357* module

The *DL 357 SIM* module implements two major functions: The first is the interface between a *DL 300* ADC system (*SCANBUS*, 16 bit) and a *VMEBUS* system (32 bit). The second is the below explained zero suppressing readout controller with a local hit buffer.

The readout controller allows autonomous, processor independent readout. The zero suppressed data of all scanning channels are stored in a hit buffer of 64 kByte. (Because of a hardware problem only 32 kByte are usable, see below.) Assuming the *DL 302* has stopped and the data are stored in the *DL 312* module memories, the *SIM* starts to read out sequentially the digital ADC data with two channels in parallel. This data stream is compared on the fly with programmable threshold registers for start and stop of the hits. As long as more than two consecutive data words in either data byte are above threshold they are stored into the hit buffer memory. A programmable digital delay assures that 2 to 5 pre-samples and 3 to 0 post-samples (the sum is always 5) are stored. Additionally, the time and wire information and a marker word are inserted at the start of each hit blocklet to allow later hit separation and time reconstruction on a data processing computer. A total word counter and an event counter proceed each event data block for later control of it. Data readout stops if a programmable *end wire* address is reached. If the last wire is odd, also the next wire is read out! The *Common Stop Mode* is supported very efficiently by means of special online address calculations during scanning.

To start scanning it is useful to make use of the same signal which stops the sampling in the *DL 302* module. At the end of this scan procedure the *SIM* raises its *Hit Buffer ReaDY (HBRDY)* flag on. This signals to the connected *VMEBUS* computer that it can read out the buffered data. The computer can reset the flag and start the sequence again after reading the data. Overlapped operation is possible,

which means during hit buffer read the sampler is allowed to sample again, but the program *simread*, used here, does not support it, because in the planned experiments the readout time is not critical.

Our version of *DL 302* module has a hardware error. After a short period of use, probably for temperature reasons, the most significant bit of each data word is set to one. So it is impossible to address the upper 32 kByte of the hit buffer memory. Also there result some calculation problems to decide if the bit has to be reset to zero in the received data or not, to be resolved by the *sim2mat* program.

A problem to be solved by the *simread* program is that the hit buffer memory is cyclical, this means a hit event might begin near to the maximal address and finish at a lower address. To avoid recalculations, not simple because of the hardware error, the *scan range* and *scan offset* are reset before each event to avoid an overflow. With 1024 bytes per channel and 32 kbyte secure hit buffer memory it is impossible to have problems even for the theoretical case of maximal hit lengths on each channel, if one uses only 32 channels or less. In realistic experiments there should be no problems even with 96 channels, but the program fails without warning in the case of overflow. How to setup and control the registers of the *DL 302* and other *Struck* modules, and also how to solve the described problems in detail, can be found out by reading the source code in the appendix.

4.6 The DAQ subsystem

The local computer system used to collect the data from the ADCs is a *Motorola MVME230x* board with an added digital I/O board *XVME-240*. Because this is a normal *VMEBUS* system it could be changed without many problems with other systems of other companies.

As operating system is chosen *VxWorks*, a real-time operating system of high performance. The kernel uses an interrupt driven task scheduling algorithm, based on priorities of the tasks. Run time facilities include *POSIX* interfaces, networking, file system support, a command shell, symbol table and a dynamic linker. All *VxWorks* facilities are provided by libraries of C routines. This has the following effects:

- Any C routine can be invoked interactively, using the *Tornado* shell (if purchased).

- Any C routine can be spawned as a *VxWorks* task, either from host tools or from *VxWorks* applications of the user.

- C routines can be connected to an interrupt or timer.

This concept has the advantage that the program can be designed outside of the device on another computer. To test or run it there, one needs only to cross compile with the *VxWorks* compiler for the chosen host operating system. In the case of *Unix/Linux* the *GNU* tools are supported. Also the free *GNU* C compiler can be used if some libraries are changed with the ones of the *VxWorks* system. The resulting object code has to be uploaded to the *VMEBUS* system and this step finally results in the machine-readable (executable) code. Also the *GNU* C++ compiler is supported. Unfortunately the *<stream.h>* library of the used installation does not work, for this reason the functions *puts* and *scanf* instead of *cout* and *cin* (which are more powerful and easier to use) are chosen to create the application program *simread*. This could be run on the local *VMEBUS* system instead of the final version with *LabVIEW* support. The source code is shown in the appendix; also there one can find an explanation concerning the restrictions of the small version compared to a final version.

The *simread* program sets up all the hardware of the device, reads out the scanner and sends the data to a connected computer, without manipulating them. (This is done in the program *sim2mat*.) The user is able to effect the following:

- Reset and setup all the used hardware modules.
- Setup the parameters: First and last used scan wire, thresholds for begin and end of the hit events, memory limit to use for sampling.
- Choose a filename (and path) to write the data.
- Get a new set of data after a new experimental event (cyclical) and be able to create a pulse to produce an artificial event for test purposes.

The small version here asks the user for input on the local terminal or one connected via remote login. The final version to be developed will be controlled by the program *LabVIEW* running on another computer (also with other operating systems as *UNIX*, like PC's) connected via *TCP/IP* protocol. All the setup and control mechanisms depending on the measurement (meaning all subroutines), will be exactly the same, only the main routine will be changed and the code for networking will be added. *LabVIEW* is a powerful development tool to control instrumentation from a computer. It can read and write digital (or digitised) data from/to connected systems and offers to a human some control elements on a windows screen like for example measuring instruments and switch levers.

It follows a sample session for the bone-version of the program *simread*:

```
simread 1.0, scan data reader running ...

Please choose the path and filename for the scanned data.
~data/test.in

Please choose the last used sampling wire. 1 is the first wire in the
first DL312 module. There are 4 wires in one module. The last possible
wire is 96!
1

Please choose the maximum length of a sample. 50 is the minimum (because
less does not make sense) and 1023 is the maximum amount of memory.
1023

Please choose the threshold defining the start of hit event. The maximum
is 254 (but does not make sense). The minimum is 1 to take all!
15

Please choose the threshold defining the end of hit event. The maximum
is 254 (but does not make sense). The minimum is 1 to take all!
15

Please choose the number of pre-samples. Possible are the values 2,3,4,5.
5

Please choose one of the following options by typing only one of the
characters and 'enter':
s = simulate a hit by producing a pulse with the hardware
r = read out the hit memory and write the data into the chosen file
    (After a hit!)
q = quit the program
s
**** Simulated hit launched. ****

Please choose one of the following options by typing only one of the
characters and 'enter':
s = simulate a hit by producing a pulse with the hardware
r = read out the hit memory and write the data into the chosen file
    (After a hit!)
q = quit the program
r
**** Hit data written into file. ****

Please choose one of the following options by typing only one of the
characters and 'enter':
s = simulate a hit by producing a pulse with the hardware
r = read out the hit memory and write the data into the chosen file
    (After a hit!)
```

```
q = quit the program
q
Program finished by user command.
```

4.7 Data conversion: The program *sim2mat*

The data conversion, as already mentioned, is not done directly on the *MVME230x* system but on a connected *UNIX* computer server because of its much higher calculating power. The program *sim2mat* converts the raw data of the *SIM* module, received by the program *simread*, into a standard output code readable by the *MATLAB* program. If the option "S" is set, it also reconstructs the data that was in the *DL 312* modules. For every channel an *ASCII* file with 1024 numbers representing the hit memory of 1024 bytes is generated. The *LabVIEW* system can read these files and use them to simulate an oscilloscope output for each used channel. A later version of a program like *simread* is able to append several hundred hit data events in one file. For this purpose the length of a blocklet can be chosen with the option "A". In this case the program reads and interprets all data blocklets until it finds one with data not valid or the file ends. (If the first blocklet is not valid, it returns an error.) All blocklets are assembled to one big event which simplifies the later data processing with the *MATLAB* macros. Other options set the input and output filenames and the minimal height and length of a hit to be saved. At least the last used wire and the hardware error management can be chosen.

One can read how to use the program and what can be manipulated in the "help—information" of the sample session shown below. The more hidden features of the program are to re-change high and low words that are exchanged because of an incompatible hardware connection. Also a very rigorous control of the input data file prevents the working with faulty data. As communication interface with the user command line options are chosen because this allows to use the program also in batch-files without explicit user-I/O. Once the options are adjusted the program can be called multiple times without a further need of control. In this case the output should be piped into a file because nobody will read the output on the screen, and if there are problems the messages could be scrolled away too fast. The *MATLAB* macro *analyse.m* (described below), for example, uses this program in the background. Also *LabVIEW*, if it simulates an oscilloscope output, uses *sim2mat* several times without asking a user.

The following text is a sample session, at first with an invalid option to produce the help screen, later with a valid option ("2 scan wires used").

```
mueller% sim2mat nonsense
```

```
sim2mat 3.0, Scan data interpreter running...
The options are set to:
Input data file (SIM DL363) = test.in
Output data file (MATLAB) = test.m
Maxwire = 1   , Presemples = 5
minimal Length of hit = 3   , minimal Height of hit = 1
hardware Error flag = 0

HELP - INFORMATION
------------------
Possible options are: M=1 (preset) to M=96 for the last used
scan wire, counting starts with 1. P=2 to P=5 (preset) adjusts
the start of the hit and should be the same like set in the SIM!
L=3 (preset) to L=1000 suppresses 'hits' shorter than set here
what suppresses some noise. H=1 (preset) to H=254 suppresses
'hits' lower than set here to suppress noise, if it is set higher
as the Threshold in the SIM.
The SIM Hardware in our lab sets the most significant bit to one
after some time. This error is detected and resolved by this
program. But if you know that this hardware error may exist, it
is safer to set E=1 (instead of preset E=0). Because, if the
error exists only sometimes, the program could fail to detect it
(because it does not exist in that moment) and make mistakes
if it begins to exist a bit later. The only disadvantages of
resolving the hardware error are: 1) You can use only 32kB of
the 64kB SIM buffer for one event. 2) Eventcounts > 32767 are
a little bit ambiguous because there are always 4 possibilities
(Eventcount OR 10...0 10...0 binary).
The filenames are set with I= for the input SIM DL363 data file
and O= for the output MATLAB data file. You can use any valid
path but only less than 255 characters (preset I=test.in O=test.m).
If you add the parameter S= with a valid pathname (for example
S=test), this program generates for each scan wire a file like
'test.12' where the number after the dot is the wire number. These
files contain the whole scanning result in a format readable by the
LABVIEW system. To be compatible with all computers where LABVIEW
probably is running, please do not use more than 8 characters and
no dots or special keys in the filename.
If multiple SIM data pAckets shall be interpreted in one run, they
have to have all the same length. Set the packet length with A= and
the correct packet length (in words, 2 byte each). The whole file
has to be shorter than 1 Mbyte because of a limited buffer.
Attention: The 'A' and the 'S' option cannot be used together!

Example: sim2mat I=~counts/data.1 O=../hello M=5 P=4 L=50 H=20 E=1
         (it does not matter what flag you set first...)
```

```
mueller% sim2mat M=2

sim2mat 3.0, Scan data interpreter running...
The options are set to:
Input data file (SIM DL363) = test.in
Output data file (MATLAB) = test.m
Maxwire = 2  , Presemples = 5
minimal Length of hit = 3  , minimal Height of hit = 1
hardware Error flag = 0

O.K. - but hardware error detected! (resolved...)
```

To conclude this topic, one can see a listing of the file *test.m* produced with the above commands and without a real hit. The hits are indexed to separate them. All information is stored in variables which can be used directly by the *MATLAB* program. The "hit events" registered here are electronic noise. Because a real hit event would have a minimum length (for example 150) and height (here 50) one can avoid registering these virtual hits by setting the options L=150 and H=50. The sampling rate is $100\,MHz$, so the time distance between two counts is $10\,ns$. The information "`Hitstart(1) = 231;`" for example has to be interpreted as $2.31\,\mu s$ after the start of sampling.

```
Eventcount = 0;
Wirenumber(1) = 1;
Hitlength(1) = 3;
Hitstart(1) = 231;
Hit(1,1) = 7;
Hit(1,2) = 7;
Hit(1,3) = 2;
Wirenumber(2) = 1;
Hitlength(2) = 3;
Hitstart(2) = 239;
Hit(2,1) = 6;
Hit(2,2) = 6;
Hit(2,3) = 15;
Wirenumber(3) = 2;
Hitlength(3) = 14;
Hitstart(3) = 272;
Hit(3,1) = 14;
Hit(3,2) = 14;
Hit(3,3) = 2;
Hit(3,4) = 2;
Hit(3,5) = 15;
Hit(3,6) = 2;
Hit(3,7) = 7;
Hit(3,8) = 7;
```

```
Hit(3,9) = 14;
Hit(3,10) = 2;
Hit(3,11) = 2;
Hit(3,12) = 10;
Hit(3,13) = 2;
Hit(3,14) = 2;
NumberOfHits = 3;
```

4.8 Data analysis: The *MATLAB* macros

To obtain the pulse shape of the current signal (or of the voltage signal) at the detector outputs, the program *MATLAB* is used. In our case we make use of three powerful features, which are unpleasant to program in C++: polynomial fitting; solving of linear equation systems; the graphical output with automatic scaling of the axes. The special commands needed to understand the code without knowing anything about *MATLAB* are described in detail in the appendix, in front of the source code of the *MATLAB* macros. The data analysis is divided into three macros, which are programs to be run from the *MATLAB* input shell.

- The macro *analyse.m* controls the user I/O and launches the other programs when used.
- The macro *polish.m* adjusts the received data and frees them from the worst noise.
- The macro *decon.m* filters the data in different manners and deconvolutes the signal.

All these macros are described in the following sections, and also some difficulties concerning the deconvolution are mentioned.

In the following sections some graphical output is shown to explain the effects of the filters and deconvolution algorithms on the received data. To have a well defined starting-point, a test data set was constructed like shown in figure 4.8, where the x-axis shows the digitisation steps while the y-axis shows the amplitude, measured in ADC output steps. The vertical gain is the same for all four parts of the figure.

The shape of part (a) of the figure corresponds to a typical signal expected for an impulse caused by a single hit in the *Ge* crystal. We did not choose one of the simulated waveforms, like shown in figures 4.15 to 4.17, because their sharp edges are not to be expected in reality and cause artificial problems to the deconvolution. The used waveform is a smoothed one, but with a secondary hump to make it not too simple.

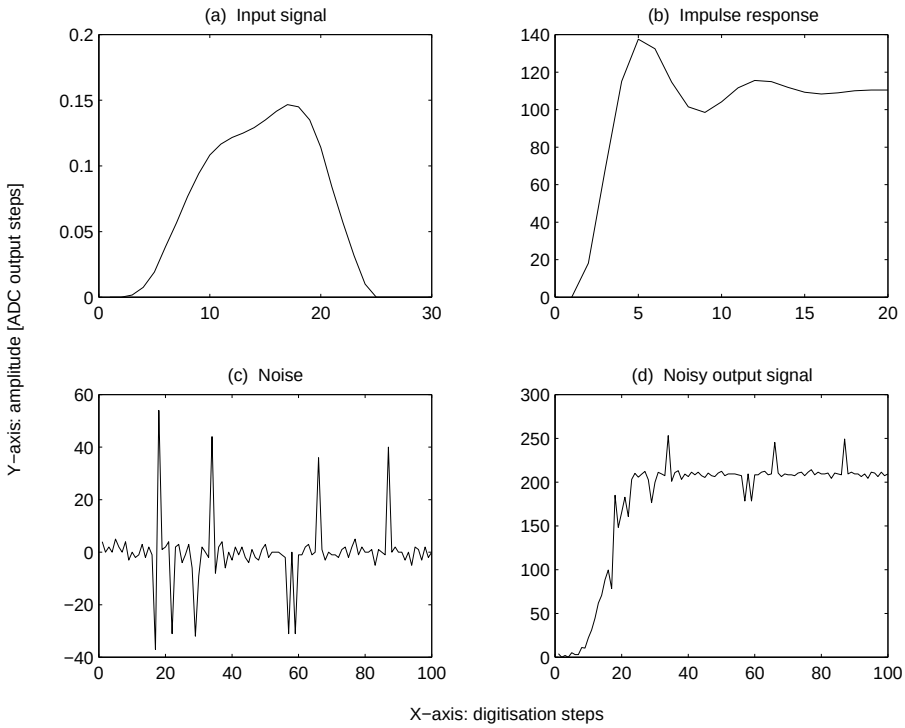


Figure 4.8: Construction of a noisy test data set.

Part (b) shows the response function for an input delta signal, as simulated with *Spice*, in steps of $10\,ns$.

Part (c) shows an example of real electronic noise measured with the *Struck* system. The sharp spikes of high amplitude, seen in the figure, cause big problems and it would be a great advantage to identify the part in the setup producing them so to avoid them. In the case of our test data this noise is guilty of producing an artificial third hump in the reconstructed input signal. The rest of the noise has an amplitude of roughly four vertical units. This noise is a hard test for the deconvolution algorithms taking in account that the maximum amplitude of the signal to be reconstructed is only about 0.15 vertical units. Deconvolution is very sensitive to noise, and for this reason every slight improvement in noise reduction results in much better reconstructed waveforms. As one can see in the section about the macro *decon.m*, the result achievable with this big amount of noise is acceptable but not too good. If better algorithms for deconvolution are not found in future (and at the moment there are no hints to expect them), the noise has to be reduced drastically if one wants to achieve reasonable results in real measurements.

The last part (d) of the figure shows the resulting test data set. The input signal shown in part (a) is convolved with the impulse response shown in part (b)

using directly the equation 3.5. The Fourier transform provided by *MATLAB* is not applied here but only in the deconvolution process in order to avoid errors which would be masked by the use of the same method in both directions. In the same spirit in the convolution we applied the impulse response simulated with *Spice* while in the deconvolution the approximate functions (4.1 and 4.2) are used in order to test their validity. Finally the noise of part (c) is added. One can see distinctly that the rising edge of the signal, which is the important part for the deconvolution, is damaged very much by the noise.

4.8.1 The central user interface

The only macro which has to be started directly by the user is *analyse.m* (typing "analyse" in the *MATLAB* input shell). This macro, at first, asks for a filename (and path if wished) where it can find the data to be investigated. The file has to contain valid *SIM* data blocklets, like the ones taken for example with the *SIMREAD* program. If the blocklets have got the same length, it is allowed to append them, what can be done for example with the standard *UNIX* command "cat". Consequently the next thing the macro has to request is the length of the *SIM* blocklets and possible control flags for the *sim2mat* program. Because it is very troublesome to do this every time one uses the macro, the version presented here has at the beginning of the source code a "user adjustable part", where the general constants can be set, so there remain no questions about general settings in the data analysis session. This method is implemented in the same manner in both of the other macros, it can be compared with the setting of ".rc" files under *UNIX*. After the needed information is obtained, the macro launches *sim2mat* to transfer the data in a format readable by *MATLAB* like described in a section above.

At this point the macro starts a loop until a user break ("Ctrl" + "C") or a user quit by answering "0" to the question: "Please choose the channel number to be worked up". Answering with a valid number results in a secondary loop to choose the hit number. Choosing nothing ("Return") results in the next hit found for the actual channel. Choosing a hit number belonging to another sampling channel switches also the channel, ignoring the first decision. If no more hits are found for the actual channel it returns to the first loop asking for a channel number.

Inside the loops the data can be processed. At first the raw data of the chosen hit are shown in a first graphical window. Then a second window is prepared and the macro *polish.m* is launched to adjust the data freeing them from the worst noise. The result is shown in the second window by the macro *polish.m*. After the data are presented in a graphical manner the program asks the user to decide if it is a

hit or only noise. Deciding the data are noise results in continuation of the loop described above. Deciding they are a hit (meaning caused by gamma radiation) results in further data processing. This macro switches to a third graphical window and launches the macro *decon.m* to deconvolute the adjusted, polished data. Finally, if the deconvoluted data are presented by the *decon.m* macro, the user is asked for a filename to save it. As example for the graphical output of the *MATLAB* macros is shown, in figure 4.9, real data measured with the *Struck* system.

The format of the stored data is *ASCII* with 50 real numbers, each number in one line. No other characters as numbers and the decimal point are added. All data processing software we know is able to read such a simple format, so the further data processing should not be a problem in the sense of compatibility. If the *analyse.m* program is quitted by the user it finishes with deleting all created temporary files.

This user interface using *MATLAB* is preliminary. Finally the user will control the *MATLAB* macros using the *LabVIEW* system, but programming a uniform *LabVIEW* user interface for the whole experiment is planned as a future work, not for this thesis.

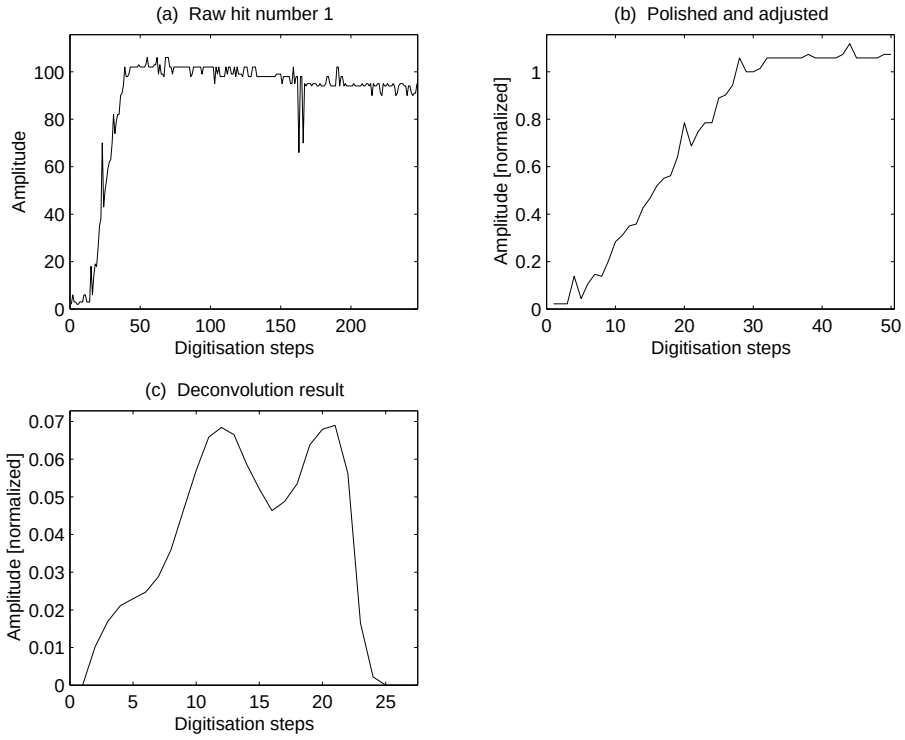


Figure 4.9: An example plot of the graphical output window.

4.8.2 Polishing the received signals

The macro *polish.m*, described here, is launched by the macro *analyse.m* like a function in C++. As input it expects a variable "Hitselected" set to the interesting hit number. It is stored in the file *hitselected.txt*. As output it produces an *ASCII* file with the name *polished.txt* containing the rising edge of the hit data in 50 integer numbers, to be processed afterwards by *decon.m*. Also it produces the same 50 data as graphical output for the prepared window in *analyse.m*. It is important to know that all variables are global in *MATLAB*. Our solution of this problem is resetting all variables in the beginning of each of the three macros and write the real global ones to the disk. This should not cause too much hard disk traffic because in modern computers the hard disks are buffered.

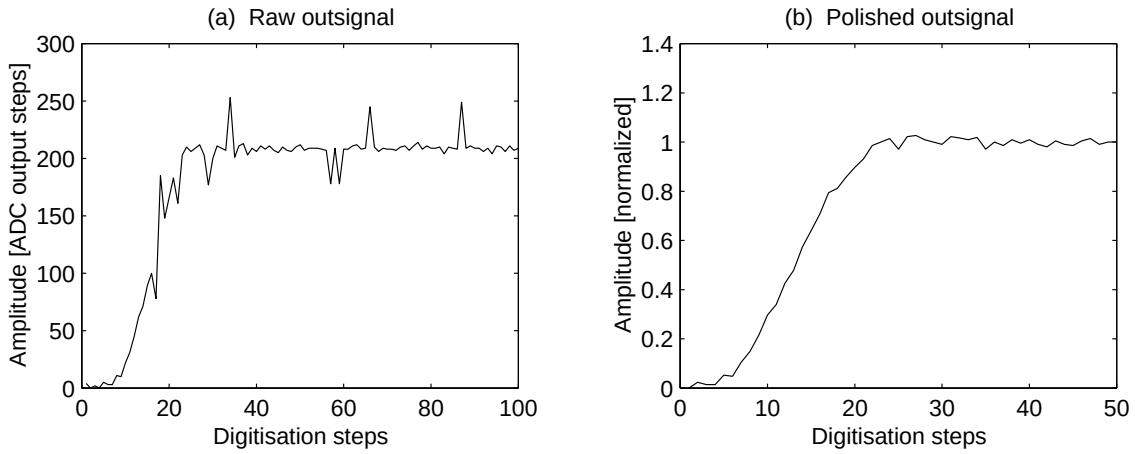


Figure 4.10: Polishing and adjusting the hit data.

As the first step *polish.m* searches the rising edge of the hit inside of its hit blocklet. A time gate of 20 digitisation steps is scrolled through the hit data. If the majority of ADC output counts is higher than 10 inside this window, the actual beginning of the time gate is taken as the beginning of the hit and the following 60 data are used for further calculation. To avoid program aborts, hits which are too short, are filled up at the end with the middle value of the last three data. Hits with a length of less than 20 are filled up with zeros also at the end because they are assumed to be only noise. To avoid working with such bad data, the *sim2mat* flag "L=60" (or greater) should be set.

The second step is to calculate a polynomial fit of 10th grade on the used data. To achieve a smoother result are added in front of the data 10 zeros, and after the end are added 10 numbers with the middle value of the last ten data. The fit is

compared with the original data and if the difference is more than 15, the original data is replaced by the fitted one in that point. The procedure described here is a quick and powerful solution to remove the large spikes, but should not be used to smooth the statistical noise. The reason is, that if too much data points are replaced the data set finally represents no longer the real data but the polynomial function. Most of the statistical noise is removed later in *decon.m*.

After the spikes are removed, the data are linearised using the equation 4.3 and then they are normalised setting the last value of the collected signal to one. Because the amplitude of the signal represents the energy of the gamma-ray under investigation, in a real nuclear experiment it should be saved before changing it. However in this work we are interested only in the pulse shapes and we do not care much about this parameter. Furthermore, the energy of the gamma rays was known and carefully selected using a linear gate circuit, so the amplitude is meaningless here and neither used nor saved.

In the next step these scaled data are adjusted a second time in order to begin with the first value above zero. This is not possible in a direct way because of the noise, so one searches the first amplitude equal or above 20 % of the maximum amplitude. This one is taken as the fifth sample.

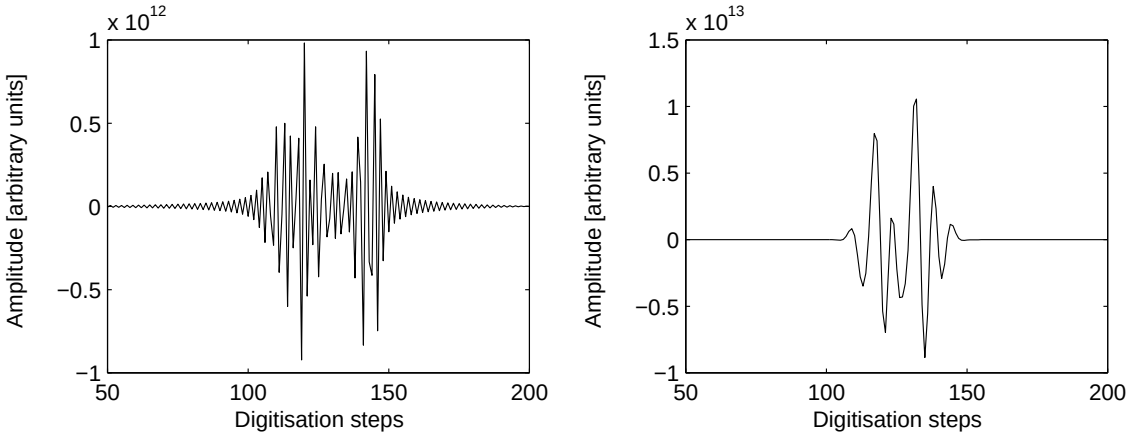
Finally any negative values, eventually created by the fitting, are removed by setting them to zero and the output is produced like described in the beginning. Figure 4.10 compares the raw data created by *analyse.m* with the polished and adjusted data leaving the *polish.m* macro.

4.8.3 Deconvolution of the signals

The macro *decon.m*, described here, is also launched by *analyse.m*. As input it needs a file *polished.txt* with 50 data values, normally produced by the *polish.m* macro. As output it produces an *ASCII* file, called *signal.txt*, with the 50 restored input signals. If the user wants to save the data, this file will be renamed later by the *analyse.m* macro. The macro *decon.m* produces also the same 50 data as graphical output for the prepared window in *analyse.m*.

As for the other two macros, all user adjustable constants can be found in the special marked beginning. One has to set the constants "Delta", "T" and "Tau" to the proper values for the equations 4.1 and 4.2. The constant "Filter" sets a low-pass filter. Finally the constant "Applybad" determines how many complex Fourier coefficients will be constructed by the constraint of finite extent. The significance of the settings in detail is described below.

In a first approach we used directly the scanned output data without differenti-



(a) 50 "good" coefficients

(b) 5 "good" coefficients

Figure 4.11: First deconvolutions without success.

ation, and only some filters were applied. They are shown in a symbolic manner in figure 3.1; zeros were added in front of the beginning and the whole signal is added in mirrored shape at the end. The same procedure is applied with the impulse response function. Figure 4.11 shows two deconvoluted signals, using a different number of Fourier coefficients: they present oscillations of great amplitudes, several decimal ranges greater than the original input signal. As one can see in the two figures, the constraint of finite extent is very powerful, as it reduces the oscillations by more than two orders of magnitude in the regions outside of the expected signal. These results are however rather useless because of the wild oscillations due to the always present statistical noise which is amplified too much in the deconvolution process.

In this example, using our test data set (figure 4.8), the tail of the signal has a length of roughly 120 samples. These samples, representing the amplified integral of the input signal, should be all the same in theory but are not because of the noise. Looking to the convolution formula (3.5), one can see that all signal samples (multiplied with the shifted response) are added up to produce the convolved signal. The same is valid for the deconvolution because it is only an inverse convolution, like derived in chapter 3. For the noisy tail of our signal this means that all the noise errors (multiplied with the response 1 for that signal region) are added to each point of the recalculated input signal. Roughly, 120 sampling errors multiplied by a middle error amplitude of 4 results in an additional error of about 480, 120 times greater than the original middle error amplitude (4 in this example). Clearly, deconvolution reacts badly to noise and in fact solving it with the 2 % of statistical

noise in the test signal has proven to be quite difficult. Like demonstrated in the figure 4.11 it seems to be impossible to deconvolute a signal with over 200 % of noise.

The solution to get rid of this problem is to differentiate both the sampled signal and the impulse response function. Deconvolution is the inverse operation of all effects happened to the signal, performed by comparing it with a δ-spike to which all the same effects happened. A Differentiation is just another manipulation and does not disturb the deconvolution of the input signal, if applied to both signal and response in the same manner.

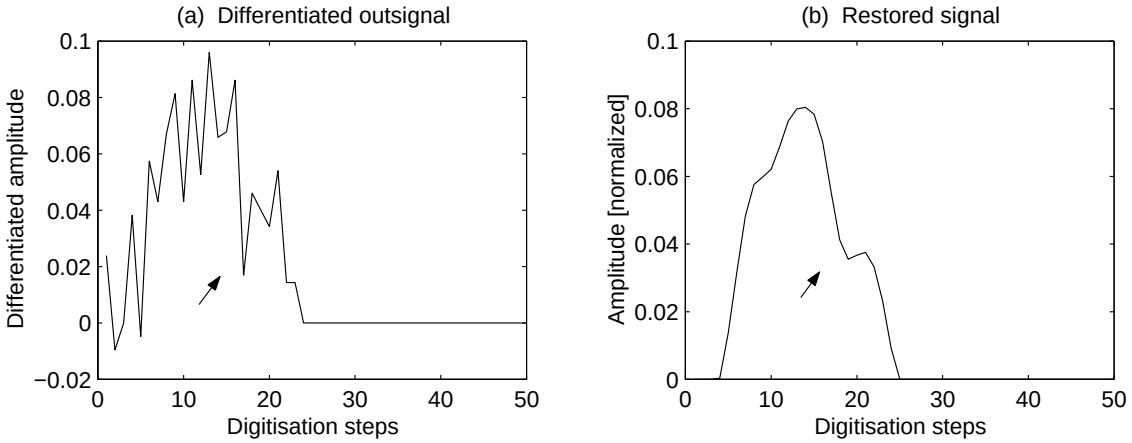


Figure 4.12: Differentiation with disturbing spike.

Figure 4.10 shows the effect of the large spikes produced by the sampling electronics in the rising edge of the signal (there rests a small bend). Figure 4.12 shows that this results in a deep valley inside the differentiated signal (part (a)), marked with an arrow. Part (b) of the figure shows the deconvolution obtained with the most recent program version. It is clear that the effect of the spike is to produce a local minimum in the restored signal.

This example shows how important it is to avoid the large noise spikes. In the case of our test signal we did not add another filter into the macro *decon.m* to smooth the disturbing parts because it would destroy also some important piece of information. Assuming that the electronic problem will somehow be solved for future measurements, the disturbing sample (number 17) was replaced with the middle value of the adjacent points, editing the file by hand. Figure 4.13 shows that in fact, in this case the reconstructed signal has a smooth behaviour.

Concluding this part about noise problems we return to the most recent program version, which differentiates both signal and response. To avoid the noise in the tail

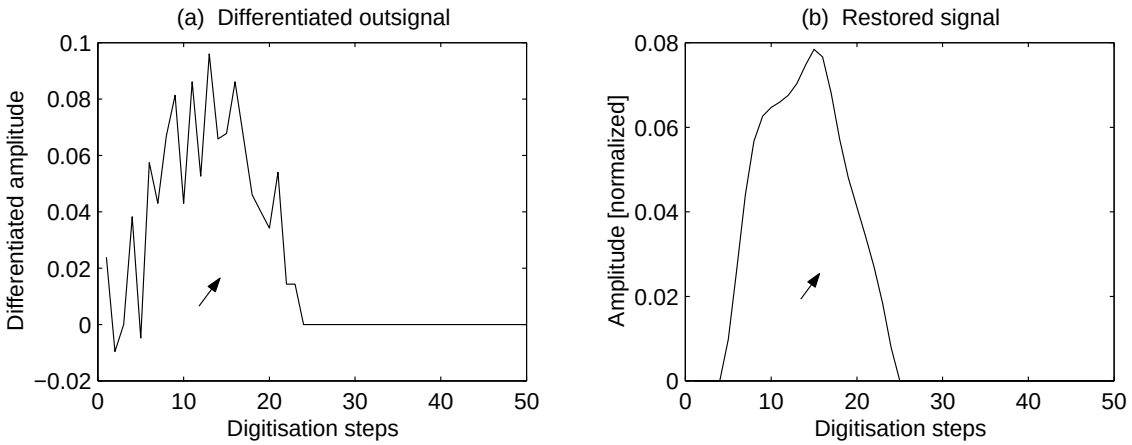


Figure 4.13: Differentiation with the spike removed.

(which would further exist in differentiated form), the signal is defined to be finished when the first differentiated value is zero or negative because this means that the maximum value was reached. Because the preamplifier which produces the sampled signals is an integrating one, there cannot be a part of the signal after the maximum (if the noise does not disturb the signal too much...). The rest of the data field is assumed to be noise and filled up with zeros becoming part of the two outside regions. This is an improvement because the length of the signal to be restored becomes shorter (by about 50 % in the case of the test signal) and the regions of noise cannot anymore effect the signal shape.

The impulse response function is constructed using the equations 4.1 and 4.2 with a proper setting of the constants like described there and a further explanation is not necessary.

In the part describing the theory about deconvolution we mentioned that, if the period without signal is too short in the used data set, there can be an overlap of neighbouring periodicals of the reconstructed signal. Beside this, the constraint of finite extent needs also a certain time interval which can be assumed as signal-free. For these reasons in front and after the end of the signal and response 50 zeros are added, resulting in vectors of 150 elements. This is in fact the minimum amount to avoid overlap and it allows the reconstruction of up to 50 complex Fourier coefficients.

The next step to be done is to Fourier transform both signal and response. This is apparently an easy task but there are two pitfalls in *MATLAB* that complicated a lot this work. The first is that even if one defines the vector length explicitly, *MATLAB* changes it without warning to correspond to how many numbers have

been given. For example, if the largest Fourier coefficient of the signal is 20 and the largest one of the response is 40, this results in two vectors of different length making the deconvolution division impossible. This kind of problem is solved by actively setting the values in the upper elements to enforce the proper vector length. The second pitfall is that *MATLAB* does not normalise the Fourier transform (what alone does not matter) but normalises the inverse Fourier transform dividing the result by the length of the vector. This combination means that after a Fourier transform and an inverse Fourier transform the amplitude of the signal is changed. Calculating with faulty amplitudes, of course, confuses the algorithm implementing the constraint of finite extent. This error is easily solved dividing explicitly also the Fourier transforms by the vector length. Another point to mention is that *MATLAB* creates variables without warning. If one uses a variable as row vector and afterwards the same variable is used as column vector, the program creates a square matrix. This is difficult to uncover because mostly the results are correct. But it is a waste of computing power (to calculate with a whole matrix being interested only in the first column), and applied either to vectors or to matrices few operations have different results.

The execution of the *decon.m* macro continues with a check to avoid zeros in the Fourier transform of the response function setting all values smaller than a limit (here ± 0.00001) to that limit. To avoid miscalculation, the corresponding Fourier coefficients in the transform of the signal are set to zero.

In the following step all frequencies above a certain value are discarded setting to zero all Fourier coefficients above the cut-off frequency. For a sampling rate of 100 MHz, a vector length of 150 and a filter for 30 MHz the cut-off correspond to the 47th coefficient (meaning that in the interval of 150 samples there is place for 47 complete sine waves). The value for the filter to be used in the *decon.m* macro is chosen with the user-adjustable constant "Filter". For the *Struck* system it does not make any sense setting it to a value higher than 47, because this system contains a 30 MHz low pass filter. The noisy test data used here were only reconstructible reducing further the filter to "Filter = 23", which is a low pass for 15 MHz.

The next job to be done is simply a division (element by element) of the Fourier coefficients belonging to the signal by those belonging to the response; afterwards the inverse Fourier transform can be performed. Because according to the mathematic rules for complex numbers the coefficients belonging to the sine terms are conjugated if divided, they have to be conjugated another time before the element-wise vector division is executed.

In the next step the algorithm for the constraint of finite extent is included in

order to recalculate some of the Fourier coefficients previously set to zero by the filter. The number of complex Fourier coefficients to be reconstructed is set by the user-adjustable constant "Applybad". With the test data set it was possible to recalculate up to 28 complex coefficients (which means 56 variables), improving the frequency limit to $32\,MHz$. Of course the constraint algorithm cannot recalculate a signal previously filtered away, but it works well in the sense of reducing the effects of peak broadening and ringing around the peak, which are undesired characteristics of each function with a truncated spectrum. Minimising the side effects outside the signal time space by choosing proper Fourier coefficients at last produces also a waveform nearer to reality inside the signal. Leaving more than the mentioned 28 coefficients in the inverse-filtered result would contribute unacceptably large noise error to the restoration, producing a meaningless result which finally becomes only oscillation.

Samuel J. Howard [11] in his work restored as an example a single Gaussian peak with 5 % of statistical noise. With his iterative program (not published) he had to truncate the spectrum after the sixth Fourier coefficient and was able to restore up to 16 coefficients (32 variables) using the constraint of finite extent. The comparison of his results with ours shows that our program does not work too badly. But also we see again how difficult the deconvolution of noisy signals is, considering the fact that Howard used a signal with 5 % noise (not much more than our 2 %), but a simple Gaussian which is not deformed under the Fourier transform.

Our implementation consists essentially of two loops and two matrix divisions. At first the matrix elements are filled up to represent the linear equation system describing the constraint. This equation system can be found as formula 3.16 in chapter 3. Before dividing the matrix a QR factorisation is needed because otherwise *MATLAB* has problems to solve the division causing often a matrix filled with zeros as "solution". After this factorisation the matrix division solves the equation system giving a vector which contains a set of correction values for the calculated points of the sampled signal. These values are afterwards Fourier transformed in the following way: a rectangular matrix with twice as many equations (rows) as the number of complex Fourier coefficients to be restored is constructed; the number of columns is roughly 100 corresponding to the samples in the extended regions on both sides of the signal. The matrix represents therefore an underdetermined linear equation system. A feature of *MATLAB*, undocumented in [5] but very useful here is that out of the infinite possible solutions it chooses the one with the smallest values at the beginning of the vector and the largest ones at the end. Because only as many Fourier coefficients as equations are needed, this results in a good restoration.

The use of an underdetermined system of equations is extremely important for our purpose because with square matrices one has no freedom to put the solutions with large values to the highest Fourier coefficients.

Adding the desired number of Fourier coefficients to the filtered and divided Fourier transform, one obtains the searched input signal (after inverse transformation). The side lobes and artifacts of the solution are removed cutting (setting to zero) all data after the first intersections with zero on both sides of the maximum. Finally the result is normalised to an area of one and the output is produced like described in the beginning of this section.

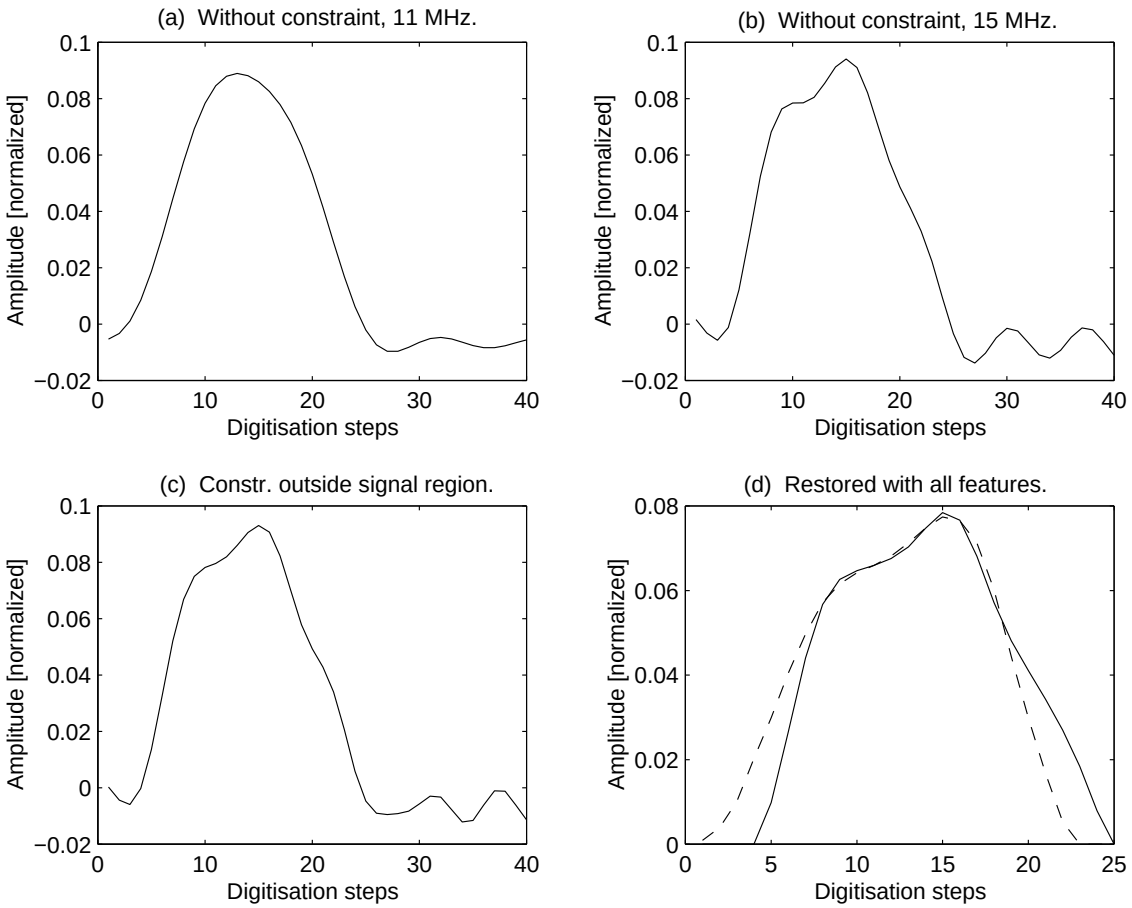


Figure 4.14: Some deconvolution results with different filter settings.

To conclude the section with some graphic examples, figure 4.14 shows four different deconvolution results. Part (a) was calculated without using the constraint of finite extent and a filter of 17, which is 11 MHz. One can see that under these circumstances the deconvolution is blind for the second maximum in the original signal form which is shown as dashed line in part (d) of the figure. Setting the

filter up to $15\,MHz$ (which is the value 23 and part (b) of the figure) makes the second maximum visible but destroys the outfit of the signal. Part (c) shows the reconstructed signal using the constraint for the regions outside of the 50 sampled values. Finally part (d) is the deconvolution result of the recent version of *decon.m* for our test data set. The dashed line in the last part (d) shows the original signal shape, also normalised to an area of one for an easy comparison. One can see that the deconvoluted pulse shape becomes very similar to the original, only the beginning and end is distinctly deformed.

4.9 Simulated pulse shapes under deconvolution

The figures 4.15, 4.16 and 4.17 show in the form of currents the same signals that figure 2.8 in chapter 2 represented as integrated charges (solid lines). One can see that the signals are not smooth, due to the fact that they have been produced by means of a finite elements calculation. Also the deconvolution results are shown for each pulse shape (dashed lines). Both simulated pulse shapes and deconvolution results are normalised to an area of one. In order to compare the experimental curves with the simulated ones in a consistent way, the user adjustable parameters are set here like in the real experiments described in the next chapter.

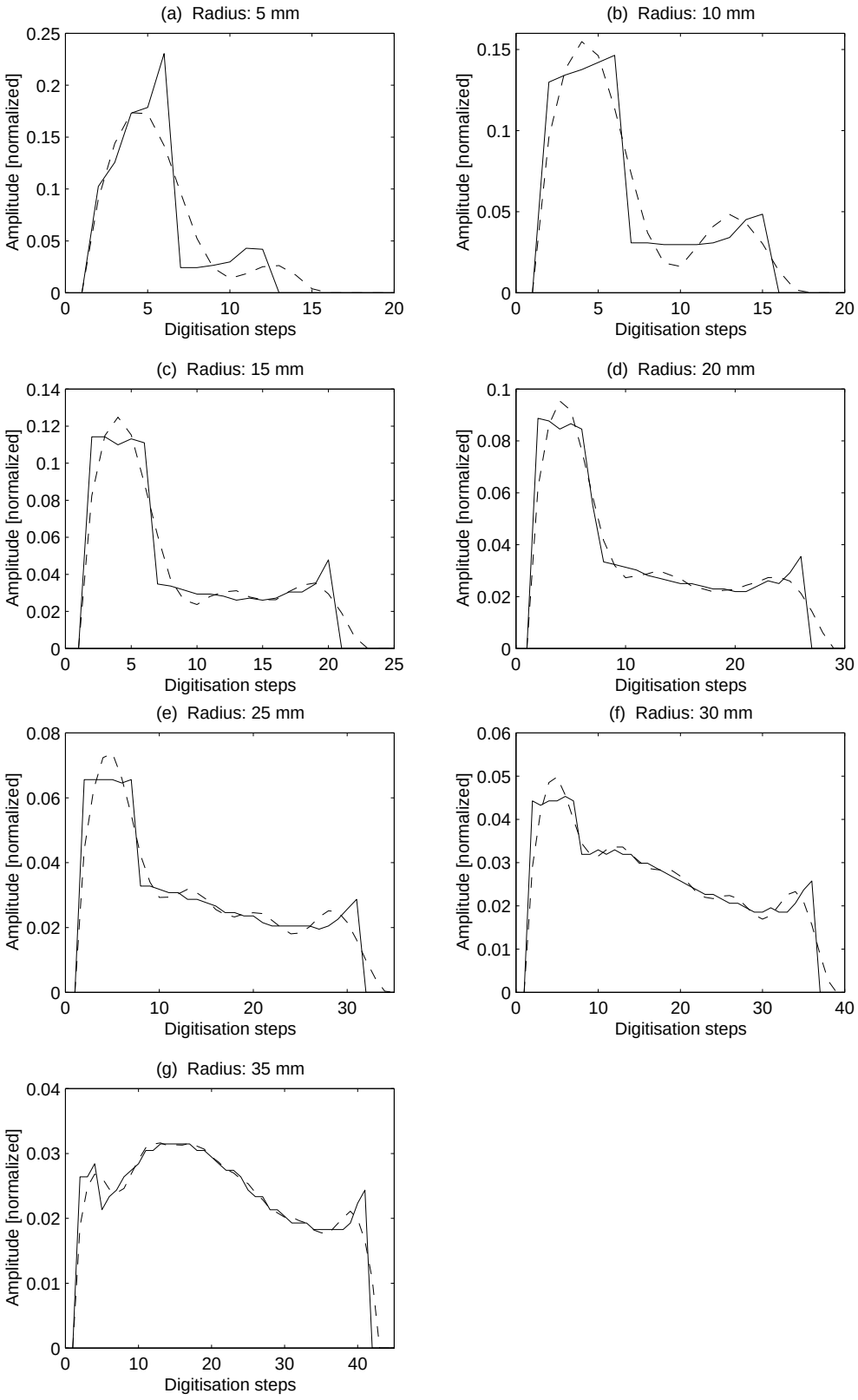


Figure 4.15: Simulated pulse shapes for the front detector region.

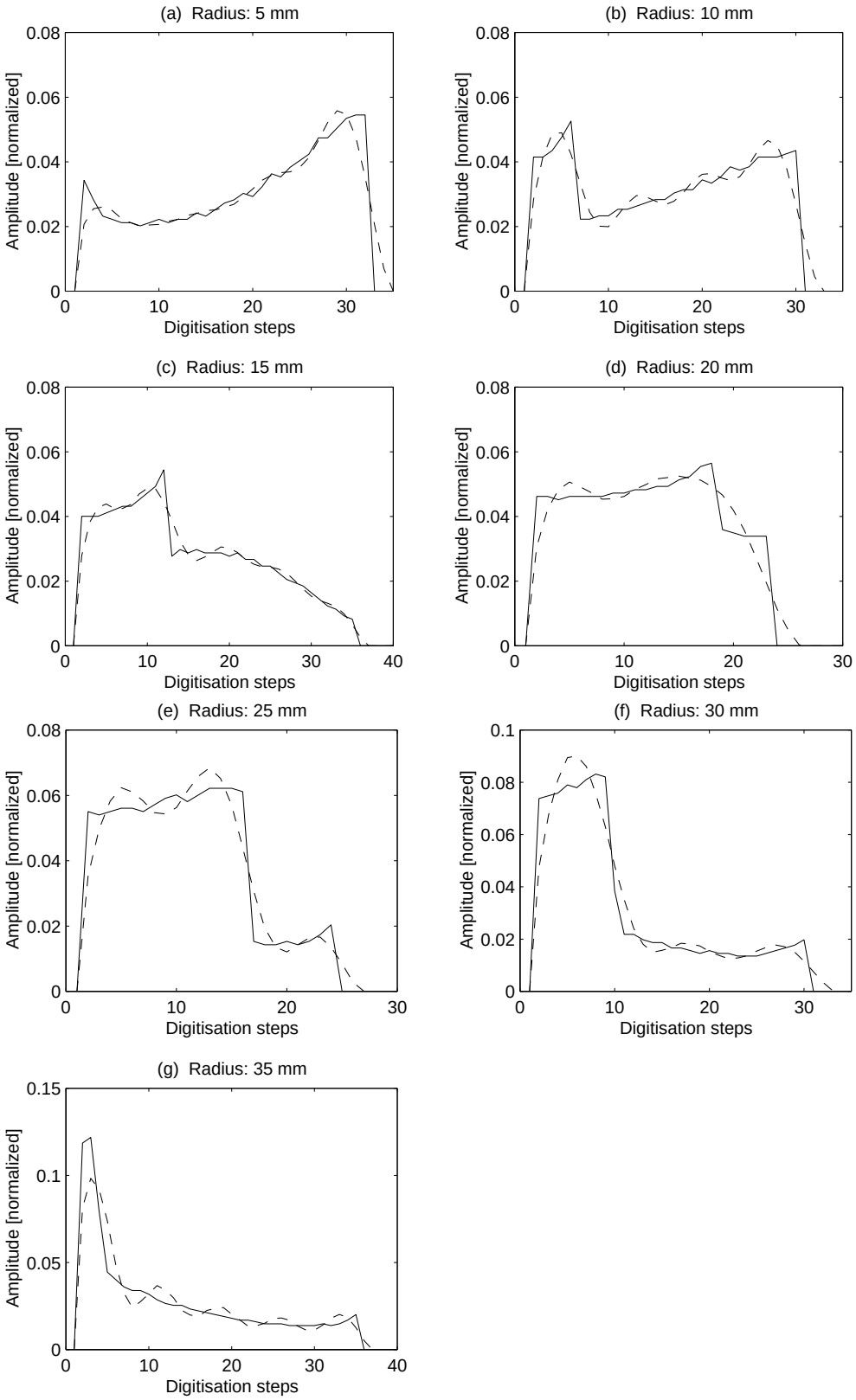


Figure 4.16: Simulated pulse shapes for the middle detector region.

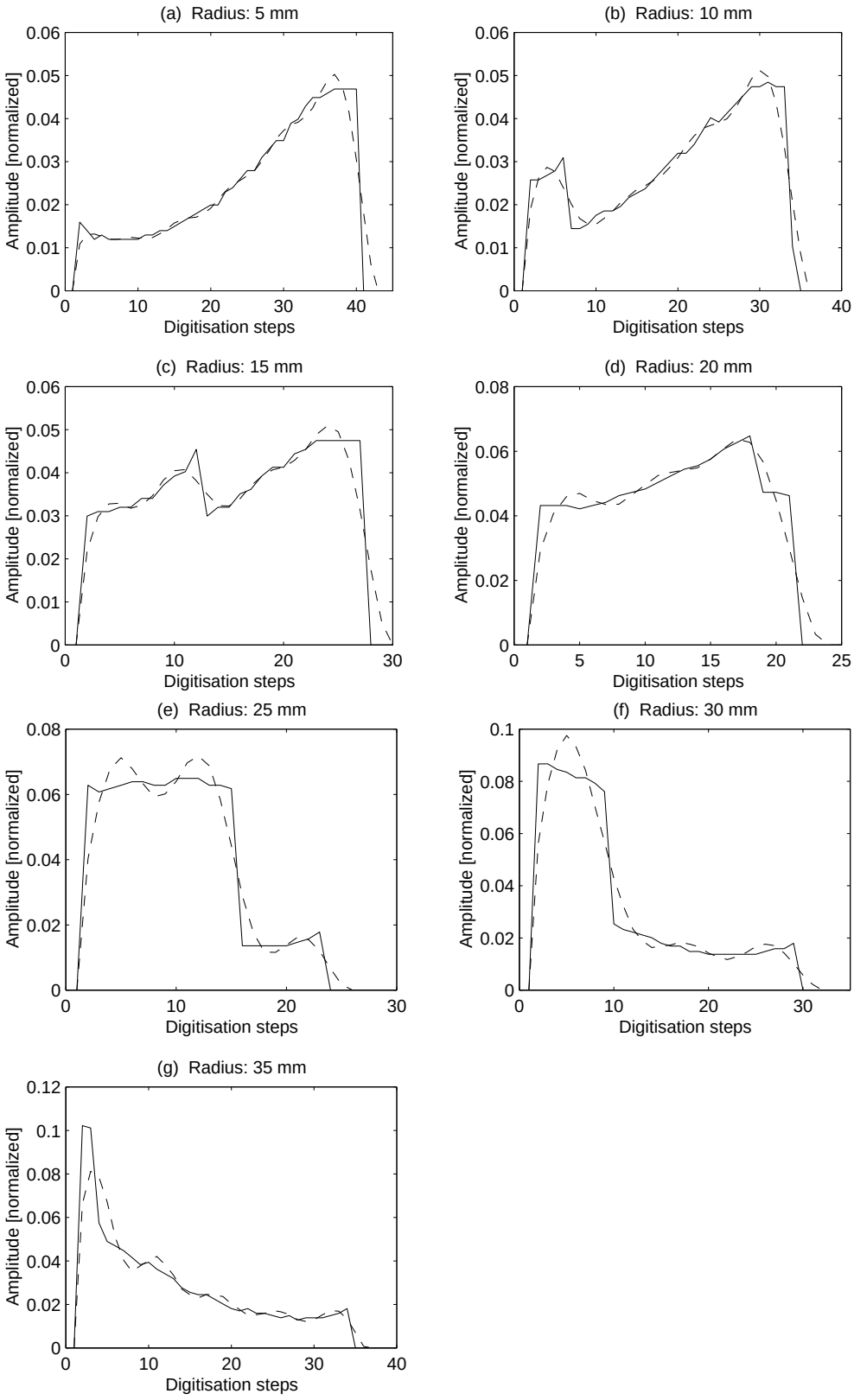


Figure 4.17: Simulated pulse shapes for the back detector region.

Chapter 5

Experimental results

5.1 Searching for a plausible impulse response

The preamplifier of the *ORTEC* detector used for the experiment reported here has been developed in our institute in Padova. Because of the big amount of time needed to find the information (models) to build up a *Spice* simulation, we could not simulate the impulse response for this version of preamplifier in time. Also we have no pulse generator in our labs to produce a pulse short enough to represent something near to a δ-spike.

The problem to solve before any data analysis is possible, is to determine the user-adjustable values T, δ and τ, which describe the pulse shape of the impulse response in our *decon.m MATLAB* macro. The only information known about the behaviour of the used preamplifier was the rise time, measured by its developer. To find out a little bit more about the pulse shape we used the data measured with the collimated source (like described below). We searched by hand, out of the 50 hits for radius zero (where the fastest responses have to be), the nine fastest rising edges. The middle value of these nine signals shows a smooth curve where one can see the oscillation of the preamplifier. In this way it was possible to ascertain the time constant T. There is no chance to find out the overshoot (or undershoot respectively) for a δ-spike, only it is known that it should become larger if the rising edge becomes steeper. For this reason the constants δ and τ are chosen in a manner to produce the rising edge known by the developer and an overshoot which is somewhat larger as the overshoot of the measured data. Initially we hoped to find out something about the overshoot by deconvolving the signals with different chosen overshoot amounts, but we found no significant difference in the deconvolution results. The dominating aspect of the impulse response seems to be just the steepness of the rising edge. The finally chosen adjustment is:

τ	δ	T
$1.2 \cdot 10^{-8}\, s$	$1.9 \cdot 10^{7}\, 1/s$	$12 \cdot 10^{-8}\, s$

Figure 5.1 shows the middle value of the nine fastest rising output signals (solid line) and the chosen impulse response for all the data analysis in this chapter, with the settings mentioned in the table above (dashed line).

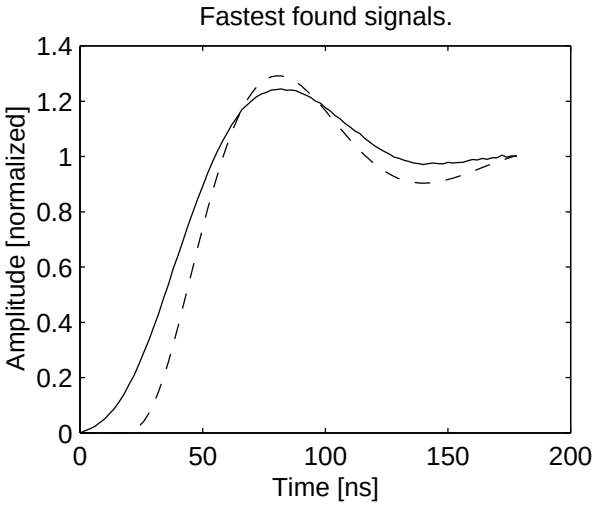


Figure 5.1: The fastest signals found and the chosen impulse response.

5.2 Some example results measured with the *Struck* system

Originally we wanted to perform all the measurements presented in this chapter with the *Struck* system described in chapter 4 because the future measurements with the segmented prototype detector will use this system. Unfortunately, while adjusting the experimental setup the *DL 302* module was damaged. As the company *Struck* was closed a few years ago, the only possibility to replace this module is to take it from another experiment (in France) and at the present time it is impossible to proceed using this system.

So far, the only existing data, which are measured using a real source of gamma radiation and also the *Struck* system, are 400 events registered during the adjustments. The source was placed in an accidental position of the floor, roughly $1\,m$ away from the detector with a not reconstructible angle and without any collimator.

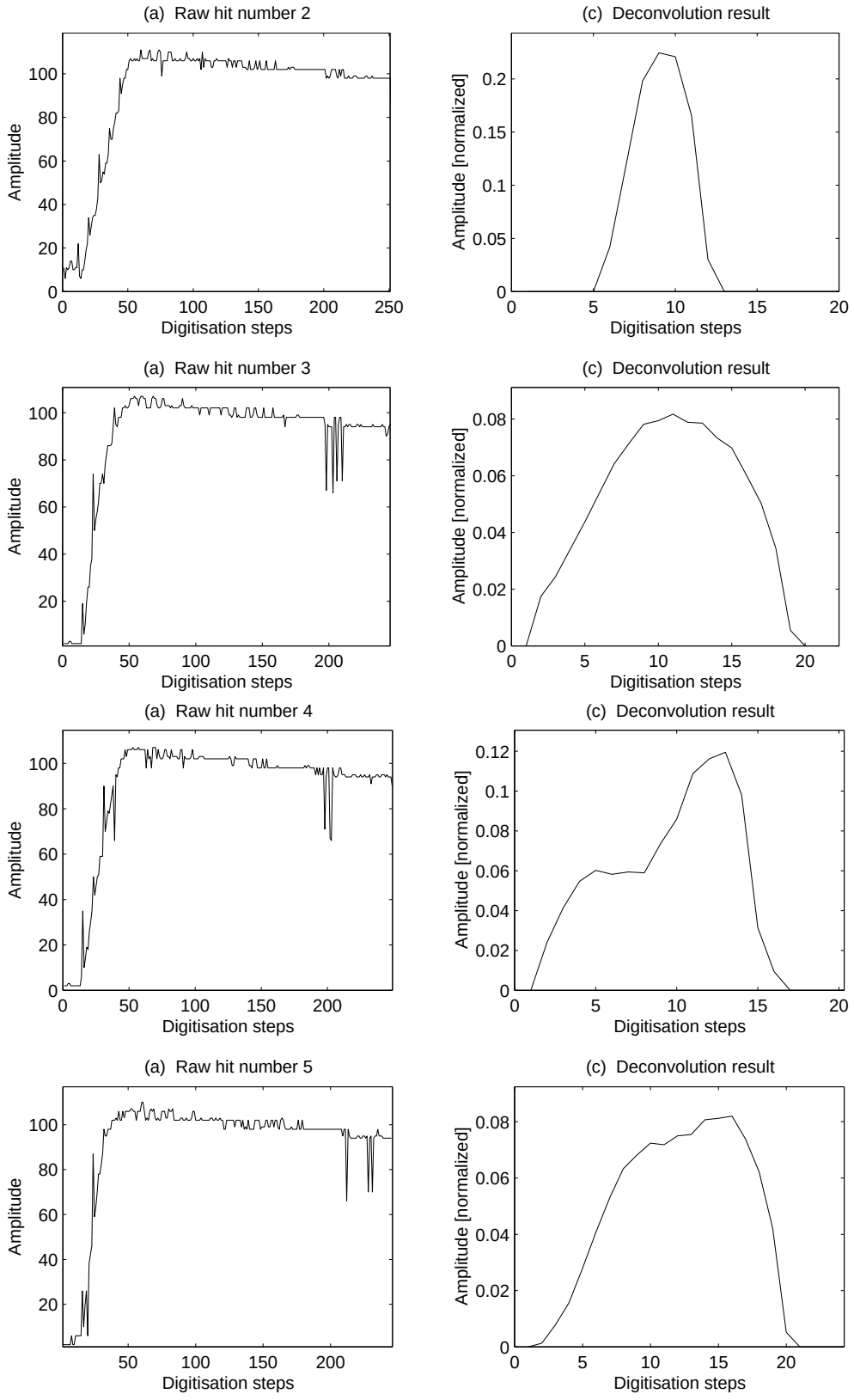


Figure 5.2: Some deconvolution results measured with the *Struck* system.

Because these "measurements" are useless for reconstructing the local positions of the gamma hits within the crystal, only five examples are added here. These examples (figure 4.9 and figure 5.2) show that deconvolution is, in principle, possible also with the *Struck* system.

5.3 Measurement with a collimated source

The first measurement that produced useful data was performed using a gamma-ray source of ^{133}Ba and a Pb collimator with a thickness of $30\,mm$ and a hole diameter of $1\,mm$. The collimated beam entered the Ge crystal from the front side of the detector at several different radial distances. The linear gate circuit selected the spectral line of $356\,keV$.

Figure 5.3 shows the result of a *GEANT* Monte Carlo simulation for the used energy and geometry. Because of their rather high energy the gamma-rays are often absorbed in more than one interaction. While for an energy of $356\,keV$ pair production is not possible one can see in figure 2.2 that the cross section for Compton scattering is ten times larger than for photoelectric absorption. Compton scattering, as explained in Chapter 2, absorbs only a part of the gamma-ray energy and produces a secondary gamma-ray going to a different direction. The result is a distribution of hits in the whole crystal, not only at the chosen radius and in the front region. In figure 5.3 are added lines for the axial cuts used in the simulations mentioned in this text and shown in the figures 2.8, 4.15, 4.16 and 4.17. Comparing the simulations for the "front" region with those of the "middle" region (with a distance of only $15\,mm$), one can distinguish very different pulse shapes. The big problem to achieve meaningful measurements is that there exists, indeed, a region with a higher probability of gamma hits in the crystal but quite a lot of hits far away from the collimated axis also exist.

The idea of the following experiment is to compare the middle value of several measurements for the same radial distance, which represent a statistical distribution of gamma-hit positions inside the Ge crystal, with the simulated data for the closed front region of the crystal in the hope to see some correlation between them.

As mentioned above one module of the *Struck* system at present is damaged. To finish the thesis work in time we decided to use a digital oscilloscope instead of the *Struck* system for further measurements. The advantages of the digital oscilloscope are that it produces less electronic noise, it does not produce the large pulses disturbing the signal in the *Struck* system and finally its sampling rate can be chosen much faster. The disadvantage is that it cannot work up the 26 signals

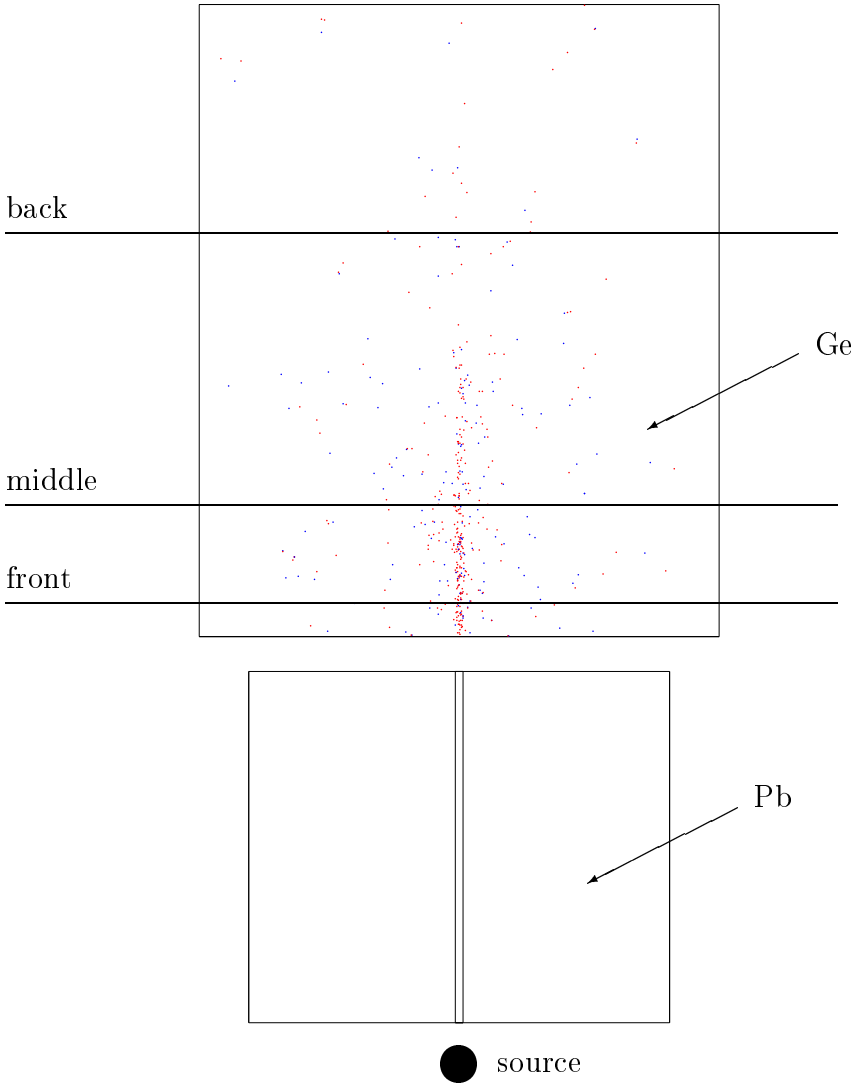


Figure 5.3: Monte Carlo simulation for 356 keV.

of the segmented prototype detector (and buying 26 digital oscilloscopes is clearly too expensive) but, of course, this is not a problem for the measurements reported here.

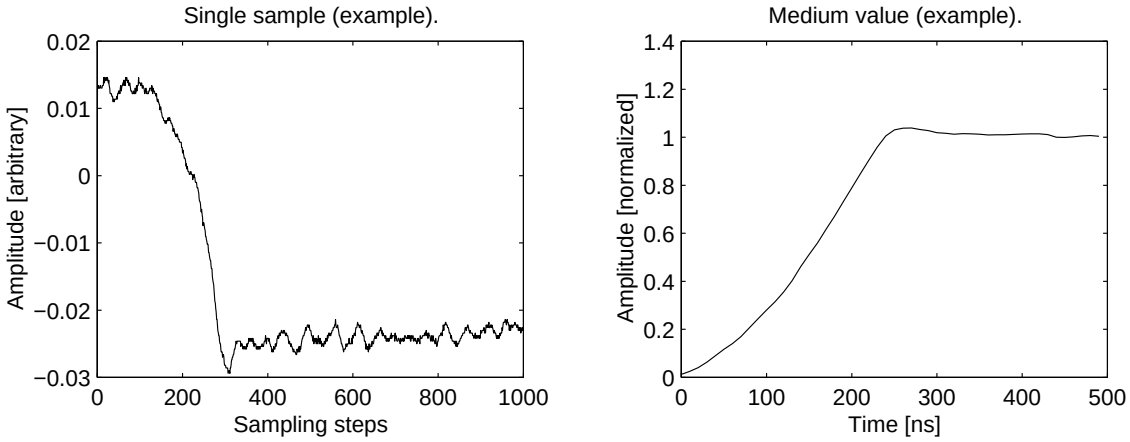


Figure 5.4: An example signal and the output of the modified *polish.m* macro for a $35\,mm$ radius.

For the analysis of the data produced with the digital oscilloscope a modified version of the *MATLAB* macro *polish.m* was used. After inverting the polarity of the signal, this version moves the rising edges of the 50 measured gamma-hits for one radius to the same place in the time scale (by shifting them) and forms the middle value of all hits. Afterwards, it normalises and shifts the signal in the same manner as the original version, but without linearising the (already linear) data. Figure 5.4 shows an example of the 50 registered hits for a radius of $35\,mm$ and the output of the modified *polish.m* for the same radius in the crystal. As one can see, the statistical noise is reduced very much by forming the middle value. This is, of course, expected for any real Gaussian noise which does not contain a systematic error.

With these very smooth signals the deconvolution worked well with each allowed user-adjustable setting in the *decon.m* macro. However, setting the filter to the highest allowed value is not very realistic because for signals recorded by the noisy *Struck* system the filter of course cannot be switched off. For this reason the filter was set to $15\,MHz$, which is the value used for the deconvolution of the simulated test data used in the previous chapter. Unless other measurements with the *Struck* system prove the contrary, we accept the $15\,MHz$ as the highest possible setting for the filter. In contrast to the filter, the number of Fourier coefficients, reconstructed by the constraint of finite extent, is set to the highest allowed value. The recon-

struction algorithm reduces only artificial side lobes and artifacts caused by the deconvolution and does not uncover signal components which have been previously filtered away. So setting it to the highest efficiency is allowed here, although this would be impossible with noisy signals because of the stability problems mentioned in chapter 4.

The solid lines in figure 5.5 show the results achieved with the method described above. The dashed lines show the simulations for the "front" region with a distance of $5\,mm$ from the front side, and the dotted lines finally represent the simulations for the "middle" region with a distance of $20\,mm$. This is, in the simulation, $5\,mm$ behind the end of the closed front region of the crystal at the beginning of the coaxial part.

For the radius of $5\,mm$ and $10\,mm$ (part (a) and (b) of the figure) one can see some correlation between the measured data and the simulated ones for the axial distance of $5\,mm$. For the $15\,mm$ radius (part (c)) most hits seem to be found somewhere between $5\,mm$ and $20\,mm$ distance. For the radius of $20\,mm$ and $25\,mm$ (part (d) and (e) of the figure) one can see more correlation between the measured data and the simulated ones for the axial distance of $20\,mm$, but with a slight hump probably belonging to some radius nearer to the front surface. For the front edge of the crystal (part (f) of the figure for $30\,mm$ and part (g) for $35\,mm$) no similarity between the measured and simulated data appears. This is not surprising because in the outer edge of the detector crystal the electric field is very weak and therefore the charges move very slowly. The result is that relatively small changes in the simulation produce very different pulse shapes. In this region the electric field depends very much on the impurity of the crystal, as one can see for example in figure 2.7. (Also near to the inner contact the field differs very much, but in the closed front region of the crystal the volume portion near to the inner contact is much smaller than near to the outer contact.) For the simulations used here, the impurity of the Ge crystal is only guessed to be 10^{10} atoms per cm^3. This is a realistic value but there is no possibility to find out the real impurity of our used crystal without opening the detector and performing a difficult crystal analysis, which is too dangerous for the detector and also too expensive for our purposes. For the future pulse shape analysis and gamma ray tracking with the segmented prototype detector the results shown here are not sufficient, the real impurity has to be found out!

Looking back at figure 2.2 another time, one can see that for γ-ray energies below roughly $150\,keV$ the photoelectric absorption begins to dominate over the Compton scattering. Figure 5.6 shows the same *GEANT* Monte Carlo simulation

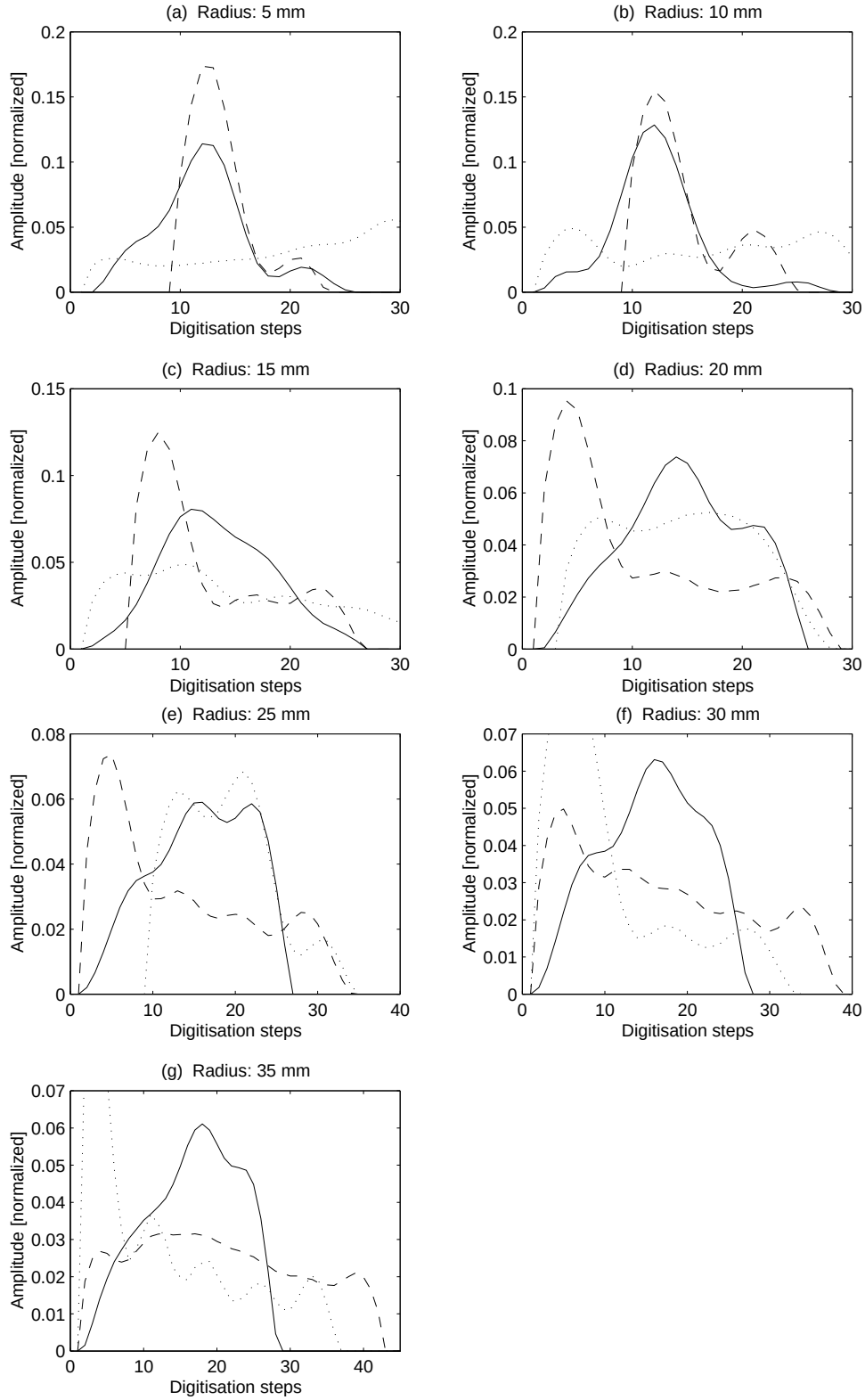


Figure 5.5: Comparison of measured (solid) and simulated (dashed and dotted) pulse shapes.

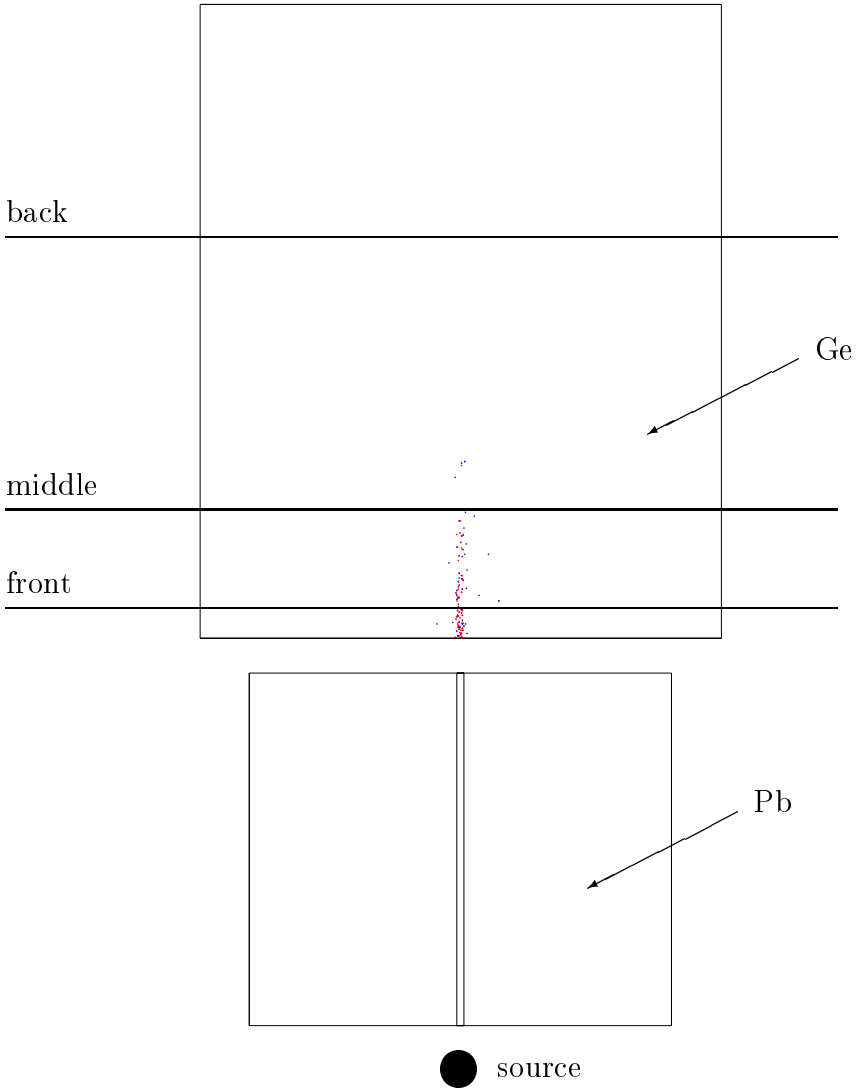


Figure 5.6: Monte Carlo simulation for $120\,keV$.

as described above, but this time for a γ-ray energy of 120 keV, which means roughly a double cross section for the photoelectric absorption as for Compton scattering. The simulation result for this lower energy shows distinctly that now almost all gamma hits are registered in a very close region in the front of the crystal. It is worth to repeat the same measurement another time with a gamma-ray energy of 120 keV or lower. Of course, the lower the energy chosen the bigger are the problems with the noise because the amplitude of noise is always the same while the amplitude of the signal decreases. For an energy of 120 keV one has to expect roughly three times more noise as for 356 keV. However, as one can see in figure 5.4, this should not be a problem using the digital oscilloscope.

5.4 Concluding remarks

As shown in the last chapters, the most urgent problem to solve is the elimination of the spikes with high amplitudes produced by the *Struck* system. Also the electronic noise should be reduced to one digit instead of the actual roughly four digits. It is desirable that the noise would be determined by the preamplifier and not by the digitisation system. Reduction of the noise from four to one digit would mean a four-time increase of the sensitivity of the system. The actual reachable sensitivity for single signals under deconvolution using the *Struck* system is only in the region above 300 keV, probably (but not tested) it could work also with 200 keV if one chooses a low-pass filtering of about 10 MHz. However, the quality of so strongly filtered signals would be questionable. As explained, the gamma energies below about 150 keV are very interesting for the first tries of gamma-ray tracking because for those energies the Compton effect does not play an important role. A possible solution to reduce the noise problems, if the noise of the *Struck* system cannot be reduced, could be an additional adjustable amplifier after the preamplifier and in front of the *Struck* system. In this way it would be possible to adjust the signal amplitude always to roughly 200 ADC counts (which is roughly 160 mV or 1.6 MeV), even if the original signal amplitude would be smaller.

The deconvolution has shown to be very sensitive to noise. Noise reduction could allow to set the software low-pass filter to 30 MHz instead of the actual 15 MHz. This would be a great improvement because 30 MHz is the cut-off frequency of the *Struck* hardware, so setting the software filter to this value means making use of the whole available bandwidth.

In Chapter 2 it was explained that the impurity of the Ge-crystal plays an important role and also the results of the measurements show this. For future

experiments with the segmented detector it is very important to find out its impurity value as exactly as possible. In contrast, the knowledge of the exact pulse shape of the impulse response of the preamplifier has proven to be not very important once its rise time is known.

Finally, because the problems described above should be solvable, we can conclude with the statement that the reconstruction of the signals in the detector crystals is possible using the method of deconvolution. We expect reasonable results also with the segmented prototype detector (when eventually it will be available) because the *Ge*-crystal and the preamplifier used here are very similar to those of the prototype. The reconstruction of the position(s) of the interaction sites of the gamma-ray is another difficult task. In this work only the very simple case of a single interaction in the front region of the crystal is taken into consideration, enforced by using radiation of low energy. This choice is valid to test out the methods developed here, but real measurements in a future segmented gamma-array need to work with multiple hits. Because of this reason the program *sim2mat* is prepared to work also with multiple-hit events.

Appendix A

Program source code of *simread*

The following source code of the program *simread* is a complete, running version but not the program used in the real experiments. This is a minimal version to be used on the local terminal of the VMEBUS computer or using a remote login like *telnet*. The original program has included a large amount of code to be finally able to communicate directly with the *LabVIEW* program using the TCP/IP protocol. This will allow to control the device from any computer connected with the LAN using a *LabVIEW* program. But the added code is not useful to explain the experimental setup, so it is left away to save paper.

```
// simread.cpp - - reads out the scanner and writes the raw data into a
//                 file on a remote computer. ATTENTION! This program has
//                 to be run directly on the VxWorks system MVME2300 !

#include <vxWorks.h>
#include <stdio.h>
#include <string.h>
#include <taskLib.h>

// *** SAM defines ***
#define DL300BASE  0x20000000
#define SAMSLOT    26  // Slot number of "DL302" module, starting with 0 !
#define SAMBASE    (DL300BASE+SAMSLOT*8192) // sam slot * 2^13 !
#define FUNCSAM    (SAMBASE+8*4)            // Scanner function register!
#define ADDRSTOP   (SAMBASE+10*4)           // Scanner overflow register!
#define STARTSAM   (SAMBASE+12*4)           // Programmed Start!
#define CLEARSAM   (SAMBASE+14*4)           // Clear Scanner!

// *** SIM defines ***
#define SIMBASE     0x20100000
#define SIMCSRBASE  (SIMBASE+0x00040000)
#define CSRSIM      SIMCSRBASE         // Sim Control/Status register rw!
#define THRSIM      (SIMCSRBASE+0x4)   // Hit threshold register, 16 wr!
```

```
#define RNGSIM      (SIMCSRBASE+0x8)   // Scan Range Register, 10 wr!
#define OFSSIM      (SIMCSRBASE+0xC)   // Scan Offset Register, 10 wr!
#define STRWIRSIM   (SIMCSRBASE+0x10)  // Start Wire Register, 8 wr!
#define ENDWIRSIM   (SIMCSRBASE+0x14)  // End Wire Register, 8 wr!

// *** DIO XVME 244 ***
#define DIO_BASE      ((char  *) 0xfbffe000)
#define DIO_CSR       ((char  *) (DIO_BASE + 0x81))
#define DIO_PORT0     ((char  *) (DIO_BASE + 0x88))
#define DIO_PORT1     ((char  *) (DIO_BASE + 0x89))  // Unused at moment!
#define DIO_PORT_DIR  ((char  *) (DIO_BASE + 0x87))

// *** set variables to grant 'unsigned long *' ***
#define funcsam   ((unsigned long *)FUNCSAM)
#define addrstop  ((unsigned long *)ADDRSTOP)
#define startsam  ((unsigned long *)STARTSAM)
#define clearsam  ((unsigned long *)CLEARSAM)
#define csrsim    ((unsigned long *)CSRSIM)
#define thrsim    ((unsigned long *)THRSIM)
#define rngsim    ((unsigned long *)RNGSIM)
#define ofssim    ((unsigned long *)OFSSIM)
#define strwirsim ((unsigned long *)STRWIRSIM)
#define endwirsim ((unsigned long *)ENDWIRSIM)
#define simbase   ((unsigned long *)SIMBASE)

int event_len;
SEM_ID mutex;
char data_buffer[32768];

char Outfile[255];  // Filename
int FstWire = 0;    // The number of the first sampling wire in the first
                    // "DL312" module. There are 4 wires in one module!
int LstWire;        // Ditto, for the last wire used in the last module!
int HScThr;         // Threshold: defines end of hit event!
int LScThr;         // Threshold: defines start of hit event!
int ScWin;          // Maximum is 1023 because of memory limit in SAM!
int Presp;          // Presamples. Possible are 3,4,5 (by hardware)!

// *** function prototypes ***
void setup();
int read_event(unsigned short *);
void dump_raw(unsigned short *,int,char *);
void oneshot();
void dl357_dump(unsigned short *);


int main()  // Controls the input and output!
            // (Instead of 'cin' and 'cout' are used 'scanf' and
```

```
            // 'puts' because of problems with the library <stream.h>.)

{
    unsigned short buffy[0x7FFF];
    unsigned short *pntrbuffy = buffy;
    int err;
    char input[255];

    puts(" ");
    puts("simread 1.0, scan data reader running ...");
    puts(" ");

    setup();  // setup the hardware...

    puts("Please choose the path and filename for the scanned data.");
    scanf("%s",Outfile,input);
    puts(" ");
    do
    {
        puts("Please choose the last used sampling wire. 1 is the"
             " first wire in the first");
        puts("DL312 module. There are 4 wires in one module. The"
             " last possible wire is 96!");
        scanf("%d",&LstWire,input); LstWire=LstWire-1;
        puts(" ");
    } while ((LstWire<0)||(LstWire>95));
    do
    {
        puts("Please choose the maximum length of a sample. 50 is the"
             " minimum (because less");
        puts("does not make sense) and 1023 is the maximum amount of"
             " memory.");
        scanf("%d",&ScWin,input);
        puts(" ");
    } while ((ScWin<50)||(ScWin>1023));
    do
    {
        puts("Please choose the threshold defining the start of hit"
             " event. The maximum is 254");
        puts("(but does not make sense). The minimum is 1 to take all!");
        scanf("%d",&LScThr,input);
        puts(" ");
    } while ((LScThr<1)||(LScThr>254));
    do
    {
        puts("Please choose the threshold defining the end of hit"
             " event. The maximum is 254");
        puts("(but does not make sense). The minimum is 1 to take all!");
```

```
    scanf("%d",&HScThr,input);
    puts(" ");
} while ((HScThr<1)||(HScThr>254));
do
{
    puts("Please choose the number of pre-samples. Possible are the"
         " values 2,3,4 and 5.");
    scanf("%d",&Presp,input);
    puts(" ");
} while ((Presp<2)||(Presp>5));
do
{
    puts("Please choose one of the following options by typing"
         " only one of the characters");
    puts("and 'enter':");
    puts("s = simulate a hit by producing a pulse with the"
         " hardware");
    puts("r = read out the hit memory and write the data into the"
         " chosen file");
    puts("    (After a hit!)");
    puts("q = quit the program");
    scanf("%s",input,input);
    if (strcmp(input,"s")==0)
    {
        oneshot();
        puts("**** Simulated hit launched. ****\n");
    }
    if (strcmp(input,"r")==0)
    {
        err = read_event(pntrbuffy);
        if (err== -1)
        {
            puts("Hardware error: Time out! (aborted...)\n");
            return -1;
        }
        if (err== -2)
        {
            puts("Hardware error: No buffer! (aborted...)\n");
            return -1;
        }
        dump_raw(pntrbuffy,event_len,Outfile);
        puts("**** Hit data written into file. ****\n");
    }
    if (strcmp(input,"q")==0)
    {
        puts("Program finished by user command.\n");
        return 0;
    }
```

```
    } while (1==1);  // Loop exit is inside the loop! ("return")
}


void setup()  // Init everything!
{
    // *** Map vsb pmc card to i/o space @ 0x2000000 ***
    pciDevConfig (0,0x10,0,0x00100000,0x20000000,0x3);

    // *** setup digital I/O ***
    *DIO_CSR = 0xff;                // Reset!
    *DIO_CSR = 0x00;
    taskDelay(sysClkRateGet()/20);  // Wait a moment!
    *DIO_PORT_DIR = 0x01;           // Set the port direction: out!
    *DIO_PORT0 = 0x0;               // Clear the out port 0 !

    bfill(data_buffer,32768,0);  // Fill buffer with 0 !

    if ((mutex = semBCreate(SEM_Q_PRIORITY,SEM_FULL)) == NULL)
       printErrno(errnoGet());  // Create Semaphore and error handling!

    // *** Init SIM ***
    *thrsim = HScThr+256*LScThr;           // Set thresholds!
    *rngsim = ScWin*(1+LstWire -FstWire);  // Scan range set to total!
    *ofssim = ScWin*(1+LstWire -FstWire);  // Scan offset set to total!
    *strwirsim = FstWire;                  // Start wire number!
    *endwirsim = LstWire+1;                // End wire number!
    *csrsim = 0x8000;  // Sim Control/Status register is set to:
                       // 5 presamples, 0 postsamples, general clear!
    if (Presp == 4) *csrsim = *csrsim + 0x100;  // 4 presamples, 1 post!
    if (Presp == 3) *csrsim = *csrsim + 0x200;  // 3 presamples, 2 post!
    if (Presp == 2) *csrsim = *csrsim + 0x300;  // 2 presamples, 3 post!

    // *** Init SAM ***
    *addrstop = ScWin;  // Stop address sample (overflow)!
    *clearsam = 0;      // Clear SAM!
    *funcsam  = 0x0b;   // Function Register is set to: 100MHz, sampling
                        // mode, after starting it runs continuously and
                        // cyclical until the Hit Flag is set to stop it
                        // Attention! The bandwidth of the analogous
                        // electronics is limited to 30MHz!
    *startsam = 0;      // SAM software start!
}


int read_event(unsigned short *leb_add)  // Main event acq. routine!
{
    int loop;
```

```
    /* wait the hit buffer ready flag */
    loop = 0;
    while (((*csrsim & 4) != 4) && (loop++ < 9999));  // Wait a moment!
    if (loop == 10000) return -1;  // Error test, time out!

    event_len = 0;
    if (leb_add == 0) return -2;  // Error test, no buffer?

    semTake(mutex,WAIT_FOREVER);  // Take a semaphore(system routine)!

    dl357_dump(leb_add);
    *csrsim = *csrsim | 0x4000;  // Reset hit buffer ready flag!
    *clearsam = 0;     // Clear SAM!
    *funcsam  = 0x0b;  // Common stop & Clock Sync!
    *startsam = 0;     // SAM software start!

    semGive(mutex);  // Give up the semaphore (system routine)!
    return event_len;
}


void dump_raw(unsigned short *buffer,int wordcount,char *name)
// Dump the buffer to a remote file in raw format!
{
    FILE* f;
    unsigned short *ofs = buffer;
    int i;

    f = fopen(name,"w");

    for (i=0; i<wordcount ; i++) fwrite(ofs++,sizeof(short),1,f);
    fclose(f);
}


void oneshot()  // Send a single trigger out to pulser!
{
    static int j=0;

    // *** send trigger to pulser ***
    *DIO_PORT0 = 1;
    *DIO_CSR = 0x02;
    for (j=0; j<100;j++) {};  // Wait a very short moment!
                              // (The real signal input for the ADC is
                              // created by the connected hardware.)
    *DIO_PORT0 = 0;
    *DIO_CSR = 0x0;
```

```
}


void dl357_dump(unsigned short *buffer)  // Take data from DL357 scanner!
{
    unsigned long dl357_len,dump_len;
    int i;
    unsigned long *ptr = (unsigned long *) buffer;

    dl357_len = *simbase & 0xffff;
    dl357_len++;  // Includes total wordcount!

    //********************// This patch is only necessary because of the
    dl357_len &= 0x7fff;  // hardware problems with the most significant
    //********************// bit of the scanner card!

    if((dl357_len % 2) == 0)  // Check for longword boundary alignment!
        dump_len = dl357_len /2;
    else
        dump_len = dl357_len/2 +1;

    // *** dump dl357 buffer to user space as is ***
    for (i=0; i < dump_len; i++)  *(ptr+i) = *(simbase+i);
    event_len += dl357_len;
}
```

Appendix B

Program source code of *sim2mat*

```
// sim2mat.cpp - - swap data of an input SIM file and create an
//                 output file in MATLAB format

#include <fstream.h>
#include <iostream.h>
#include <string.h>

char *NumToString(unsigned long);
int StringToNum(char *);
int sort();

char Infile[255] = "test.in";
char Outfile[255] = "test.m";
char Simfile[255];

// *** Presets for the command line options ***
int Maxwire = 1;
int Presamples = 5;
int MinHitlength = 3;
int MinHithigth = 1;
int Harderrset = 0;
int Simset = 0;
int Packet = 0;

int main(int argc, char *argv[])  // Management of command line options
                                  // and errors!
{
  int i,j,Error,flag=0;
  char test;

  cout << "\nsim2mat 3.0, Scan data interpreter running...\n";
  for (i=1;i<argc;i++)  // Read command line options!
  {
```

```
  test = argv[i][0];
  if ((test!='M') && (test!='P') && (test!='L') && (test!='H')
     && (test!='E') && (test!='I') && (test!='O') && (test!='S')
     && (test!='A')) flag=1;
 if (argv[i][1]!='=') flag=1;
  if (!flag)
  {
    for (j=0;j<(strlen(argv[i])-1);j++) argv[i][j]=argv[i][j+2];
    if (test=='I')
    {
      for (j=0;j<=strlen(argv[i]);j++) Infile[j]=argv[i][j];
    }
    if (test=='O')
    {
      for (j=0;j<=strlen(argv[i]);j++) Outfile[j]=argv[i][j];
    }
    if (test=='S')
    {
      for (j=0;j<=strlen(argv[i]);j++) Simfile[j]=argv[i][j];
      Simset = 1;
    }
    if (test=='M') Maxwire=StringToNum(argv[i]);
    if (test=='P') Presamples=StringToNum(argv[i]);
    if (test=='L') MinHitlength=StringToNum(argv[i]);
    if (test=='H') MinHithigth=StringToNum(argv[i]);
    if (test=='E') Harderrset=StringToNum(argv[i]);
    if (test=='A') Packet=StringToNum(argv[i]);
  }
}
if ((Packet>0) && (Simset))
   cout << "Error: The S and A options are not allowed together!\n\n";
cout << "The options are set to:\n";
cout << "Input data file (SIM DL363) = " << Infile << "\n";
cout << "Output data file (MATLAB) = " << Outfile << "\n";
cout << "Maxwire = " << Maxwire << "  , Presemples = " << Presamples
     << "\n";
cout << "minimal Length of hit = " << MinHitlength;
cout << "  , minimal Height of hit = " << MinHithigth << "\n";
cout << "hardware Error flag = " << Harderrset << "\n";
if (Packet>0) cout << "SIM pAcket length = " << Packet << "\n";
if (Simset) cout << "Control data files (LabVIEW) = " << Simfile
                 << ".**\n";
cout << "\n";
if ((Maxwire<1)||(Maxwire>96) || (Presamples<2) || (Presamples>5)
   || (MinHitlength<3) || (MinHitlength>1000) || (Harderrset>1)
   || (MinHithigth<1) || (MinHithigth>254) || (flag==1))
{
  cout << "HELP - INFORMATION\n";
```

```
cout << "------------------\n";
cout << "Possible options are: M=1 (preset) to M=96 for the last"
     << " used\n";
cout << "scan wire, counting starts with 1. P=2 to P=5 (preset)"
     << "adjusts\n";
cout << "the start of the hit and should be the same like set in the"
     << " SIM!\n";
cout << "L=3 (preset) to L=1000 suppresses 'hits' shorter than set"
     << " here\n";
cout << "what suppresses some noise. H=1 (preset) to H=254"
     << " suppresses\n";
cout << "'hits' lower than set here to suppress noise, if it is set"
     << " higher\n";
cout << "as the Threshold in the SIM.\n";
cout << "The SIM Hardware in our lab sets the most significant bit to"
     << " one\n";
cout << "after some time. This error is detected and resolved by"
     << " this\n";
cout << "program. But if you know that this hardware error may exist,"
     << " it\n";
cout << "is safer to set E=1 (instead of preset E=0). Because, if"
     << " the\n";
cout << "error exists only sometimes, the program could fail to"
     << " detect it\n";
cout << "(because it does not exist in that moment) and make"
     << " mistakes\n";
cout << "if it begins to exist a bit later. The only disadvantages"
     << " of\n";
cout << "resolving the hardware error are: 1) You can use only 32kB"
     << " of\n";
cout << "the 64kB SIM buffer for one event. 2) Eventcounts > 32767"
     << " are\n";
cout << "a little bit ambiguous because there are always 4"
     << " possibilities\n";
cout << "(Eventcount OR 10...0 10...0 binary).\n";
cout << "The filenames are set with I= for the input SIM DL363 data"
     << " file\n";
cout << "and O= for the output MATLAB data file. You can use any"
     << " valid\n";
cout << "path but only less than 255 characters (preset I=test.in"
     << " O=test.m).\n";
cout << "If you add the parameter S= with a valid pathname (for"
     << " example\n";
cout << "S=test), this program generates for each scan wire a file"
     << " like\n";
cout << "'test.12' where the number after the dot is the wire number."
     << " These\n";
cout << "files contain the whole scanning result in a format readable"
```

```
         << " by the\n";
    cout << "LABVIEW system. To be compatible with all computers where"
         << " LABVIEW\n";
    cout << "probably is running, please do not use more than 8"
         << " characters and\n";
    cout << "no dots or special keys in the filename.\n";
    cout << "If multiple SIM data pAckets shall be interpreted in one"
         << " run, they\n";
    cout << "have to have all the same length. Set the pAcket length with"
         << " A= and\n";
    cout << "the correct packet length (in words, 2 byte each). The whole"
         << " file\n";
    cout << "has to be shorter than 1 Mbyte because of a limited"
         << " buffer.\n";
    cout << "Attention: The 'A' and the 'S' option cannot be used"
         << " together!\n";
    cout << "\nExample: sim2mat I=~counts/data.1 O=../hello M=5 P=4 L=50"
         << " H=20 E=1\n";
    cout << "         (it does not matter what flag you set first...)"
         << "\n\n";
    return -2;
  }
  Error = sort();  // Call the sorting routine!
  if (Error== 0) cout << "O.K.\n\n";
  if (Error== 1) cout << "O.K. - but hardware error detected!"
                      << " (resolved...)\n\n";
  if (Error== -1) cout << "Error: Missing marker! (mostly a wrong"
                       << " filename...)\n\n";
  if (Error== -2) cout << "Error: Impossible Wire!\n\n";
  if (Error== -3) cout << "Error: Unexpected end of data file!\n\n";
  if (Error== -4) cout << "Error: Total Wordcount bigger as possible by"
                       << " hardware!\n\n";
  if (Error== -5) cout << "Error: Hit seems to be bigger as possible by"
                       << " hardware!\n\n";
  if (Error>0) {return 0;} else {return -1;}
}

int sort()  // The main sorting and manipulating routine...
{
  ifstream fin(Infile);
  ofstream fout(Outfile);
  unsigned short Wordcount;
  unsigned long Eventcount;
  unsigned char Wirenumber;
  unsigned short Relativestart;
  unsigned short i,j,count;
  unsigned short hitnum=1,hitlen,hithig;
  unsigned short *pntr;
```

```
char *buffer;
unsigned short loopcount=0;   // (needed for 'A' option)
int Return=0;                 // (needed for break out)
char Harderr = 0;            // Flag if Hardware error is detected!
char String[258];           // To construct the LabVIEW output filenames!
unsigned char bufout[1024];

// *** Runtime definition of 64k or 1M buffer ***
// *** dependent on the 'A' option ***
// one word added for sorting algorithm ...
if (Packet==0) buffer = new char[0x10001];
else buffer = new char[0x100001];
pntr = (unsigned short *)buffer;
// *** Runtime definition because needed memory ranges ***
// *** from 1024 to 98304 bytes ***
unsigned char *simpntr = new unsigned char[1024 *(Maxwire)];

// *** Read the whole data file ***
if (Packet==0) fin.read((char *)pntr,0xFFFF);
   // Small version 64k buffer and
else fin.read((char *)pntr,0xFFFFF);  // big version 1M buffer!
fin.close();
// *** fill the LabVIEW array, if needed ***
if (Simset)
{
  for (i=0; i < 1024*Maxwire; i++) simpntr[i] = 0;
  // Set the whole array to 0!
}

// *** loop for multiple-event data fields (option 'A') ***
// All events are appended to a big one, with the event count number
// of the first event in the field.
do
{
  if (Packet) pntr = (unsigned short *)buffer+loopcount*Packet;

  // *** read out the input header information ***
  Eventcount = *pntr;
  pntr++;
  Wordcount = *pntr;
  pntr++;
  if (*pntr != 0xFFFF) {Return=-1; goto breakout;}
     // Error: Missing marker word!
  pntr++;
  Eventcount = Eventcount+65536*(*pntr);
  pntr++;
  pntr++;
  Wirenumber = (unsigned char) (((*pntr) & 0xFF00)/256);   // High byte!
```

```
pntr--;
if ((Wirenumber > ((Maxwire-1) | 1)) || (Harderrset))
   // Recognition of hardware error!
{
  Wordcount = Wordcount & 0x7FFF; // Maximal total length now: 32767 !
  Eventcount = Eventcount & 0x7FFF7FFF; // Ambiguous now!
  Wirenumber = Wirenumber & 0x7F;
  Harderr = 1;
}
// *** create the output header information ***
if (!Packet)  // Write Eventcount only for the first event!
{
  fout << "Eventcount = ";
  fout << NumToString(Eventcount);
  fout << ";\n";
}
// *** sort the rest of data ***
if (Wordcount & 1) Wordcount++;
if (Wordcount > 0x8000) {Return=-4; goto breakout;}
for (count=2;count<=(Wordcount/2);count++)
    // loop for the rest of data
{
  i = *pntr;
  pntr++;
  j = *pntr;
  pntr--;
  *pntr = j;
  pntr++;
  *pntr = i;
  pntr++;
}

// *** read the events and write in MatLab format ***
pntr = (unsigned short *)buffer+loopcount*Packet+4;
        // To Packet start+4 !
j = 4;  // to count the hit data words (the first 4 words are ready)!
do // *** Outer loop between the hits! ***
{
  j = j+3;
  hitlen = 0; hithig = 0;
  Wirenumber = ((*pntr)&0xFF00)/256;                 // High byte!
  pntr++;
  Relativestart = (unsigned short) ((*pntr)&0x1FF);  // Low 9 bits!
  Relativestart = Relativestart-Presamples-2;        // Correction...
  pntr++;
  if (Harderr) Wirenumber = Wirenumber & 0x7F;
  if (Wirenumber > ((Maxwire-1) |1)) {Return=-2; goto breakout;}
                                          // Error: Impossible Wire!
```

```
do  // *** Inner loop inside a hit! ***
{
  i = *pntr;
  if (Harderr) i = i & 0x7FFF;
  if (hitlen < 1024)
  {
    bufout[hitlen] = i;
    if (i > hithig) hithig = i;
  } else {Return=-5; goto breakout;}
     // Write the hit words into buffer to decide later what to do!
  j++;       // count up the hit words...
  hitlen++;  // count up the hit length...
  pntr++;
} while ((*pntr != 0xFFFF) && (j < Wordcount));
             // End the loop at marker word or end of data!
pntr++;
if ((hitlen>=MinHitlength)&&(hithig>=MinHithigth)
   &&(Wirenumber<Maxwire))
   // Accept hit only if conform with input options!
{
  fout << "Wirenumber(" << NumToString(hitnum) << ") = ";
  fout << NumToString(Wirenumber+1) << ";\n";
  fout << "Hitlength(" << NumToString(hitnum) << ") = ";
  fout << NumToString(hitlen) << ";\n";
  fout << "Hitstart(" << NumToString(hitnum) << ") = ";
  fout << NumToString(Relativestart) << ";\n";
  for (i = 0; i < (hitlen); i++)
  {
    fout << "Hit(" << NumToString(hitnum) << "," << NumToString(i+1)
         << ") = ";
    fout << NumToString(bufout[i]) << ";\n";
  }
  hitnum++;
}

// *** Write in the LabVIEW array if needed ***
if (Simset)
{
  for (i = 0; i < (hitlen); i++)
  {
    if (((i+Relativestart) < 1024) && (Wirenumber < Maxwire))
       simpntr[i+Relativestart+Wirenumber*1024] = bufout[i];
    // Write at the right relative position in a virtual kByte!
    // (Ignore eventual existing bytes out of range, because they
    // are not important. They may exist if the time stamp in the
    // input file ("relative start of hit") is not correctly
    // adjusted.)
  }
```

```
      }
    } while ((j < Wordcount) && (*pntr != 0xFFFF));
    // *** End the loop when all hit words read or at the end of data! ***
    if (*pntr != 0xFFFF) {Return=-3; hitnum--; goto breakout;}
                                    // Error: Unexpected end of data file!

    // *** search next package, break when not found ***
    loopcount++;              // Prepare buffer for next start value!
  } while (1);                // Endless loop finishes only with breakout!

breakout:  // <<<<<<<<<<<<< Break out of the loops to finish! <<<<<<<<<<<<
  if (!loopcount) return Return;   // Error break, no valid data!
  // *** End of the loop for multiple-event data fields (option 'A') ***

  fout << "NumberOfHits = ";
  fout << NumToString(hitnum-1);
  fout << ";\n";
  fout.close();

  if (Simset)  // Write all the data in LabVIEV control files, if needed!
  {
    for (j=0;j<Maxwire;j++)
    {
      strcpy(String,Simfile);  // Construct a filename!
      strcpy(String+strlen(Simfile),".");
      strcpy(String+strlen(Simfile)+1,NumToString(j+1));
      fout.open(String);  // Open the file!
      for (i = 0; i < 1024; i++) fout << NumToString(simpntr[i+j*1024])
                                      << "\n";
      fout.close();  // Close the file, free the stream for the next file!
    }
  }

  if (Harderrset) Harderr=0;  // No warning if flag set...
  if (Harderr==1) {return 1;} else {return 0;}
}

char *NumToString(unsigned long number)  // convert a number into a string
{
  char *pstr = new char[10];
  unsigned short count;
  unsigned long expo=1000000000;

  pstr[10] = '\0';
  for (count=0;count<10;count++)  // loop for all decimals
  {
    pstr[count] = number/expo;
```

```
    number = number-pstr[count]*expo;
    pstr[count] += 0x30;  // 0x30 is offset for ASCII
    expo = expo/10;
  }
  while (pstr[0]=='0')  // Erase the leading zeroes!
  {
    for (count=0;count<10;count++) pstr[count]=pstr[count+1];
  }
  if (pstr[0]=='\0') pstr[0]='0';
  return pstr;
}

int StringToNum(char *string)  // convert a string into a number
{
  int i,j,expo,out;

  expo=1;
  out=0;
  if (strlen(string)>4) return -1;  // Break if number too big!
  for (i=strlen(string)-1;i>=0;i--)
  {
    j = string[i] - 0x30;  // 0x30 is offset for ASCII
    if ((j<0) || (j>9)) return -1;  // Break if no number!
    out = out+expo*j;
    expo = expo*10;
  }
  return out;
}
```

Appendix C

Program source code of the *MATLAB* macros

The list below describes the special *MATLAB* commands used in the shown macros, because the proprietary *MATLAB* program may be unknown by the reader. The description is simplified, sufficient to understand the source code, but most of the commands are very powerful and have a lot of options not mentioned here.

For *MATLAB* all variables are matrices, a single number is a matrix with one raw and one column, a vector has only one raw or only one column and a string is a vector of characters.

clear	clears all used variables.
clf	clears the graphical window.
input(...)	produces formatted input.
disp(...)	produces formatted output.
plot(...)	produces formatted graphical output.
subplot(...)	switches graphical sub-windows.
title(...)	produces title text of the graphical (sub)window.
xlabel(...)	produces text for the x-axis of the graphical (sub)window.
ylabel(...)	produces text for the y-axis of the graphical (sub)window.
! ...	executes the following text as *UNIX* command.
eval(...)	executes the string inside the brackets as command.
fopen(...)	opens a file.
fclose(...)	closes the file.
fprintf(...)	produces formatted output to the file.
fscanf(...)	reads numbers out of the file.
sscanf(...)	reads numbers out of a string.
~=	means “not equal” (like “<>” or “!=”).

i	is the square root of –1. Do not use it as loop counter to avoid confusing *MATLAB*!
j	is the same as above.
.	in front of a mathematical operator defines the operation as elementwise. For example * is a matrix multiplication, but .* is multiplying all elements with the corresponding ones in the second matrix.
'	is the transposition of a vector. If the vector is complex, it is also conjugated. Another meaning of this sign is to mark the start and end of a string.
isempty(x)	returns 1 if the variable x is empty and 0 otherwise.
zeros(a,b)	defines a matrix with a rows and b columns, filled with 0.
x=k:l	defines a quantity of values for x ranging from k to l with steps of 1.
x=k:m:l	is the same as above, but with steps of m.
:	defines the needed quantity of numbers (*MATLAB* decides).
fft(x)	is the not normalised Fourier transform of x.
ifft(x)	is the normalised inverse Fourier transform of x.
conj(x)	conjugates a complex vector.
[Q,R] = qr(A)	gives a QR factorisation of the square matrix A, which is a unitary matrix Q and an upper triangular matrix R such that A = Q R.
x = A \ b	solves the linear equation system A x = b.
polyfit(...)	is a built-in function for polynomial fitting.
polyval(...)	is another one, also for polynomial fitting. Both functions are needed together to produce a fitted curve.
max(x)	is a built-in function to search for the maximum value of x and its position inside the vector.
min(x)	is the same as above, but for the minimum.
real(x)	is a built-in function to take the real part of a complex vector.
imag(x)	is the same as above, but for the imaginary part.

C.1 The macro *analyse*

```
% <analyse.m> is the central user interface to analyse a set of events
% produced by the "SIM DL363".
% ------------------------------------------------------------------
% The C++ program <sim2mat> in version 2.1 or later must exist in the
% current directory! (running version of "sim2mat.cpp")
% A "SIM DL363" file must be reachable, it can be produced for example
% with the "simread.cpp" program running on the "VxWorks" system and
```

```
% transfered to the current system using a file transfer protocol.
% Also the macros <polish.m> and <decon.m> are needed in this directory.
% The output files contain the restored signals and have 50 ASCII values,
% each value in one line. The further processing of the result lies in the
% hands of the user.

clear;

% Set the "sim2mat" options here as wanted.
% Attention! Forbidden is to set here the options "M","I","O" and "S" !!!
Options='L=60 H=15 E=1';

% Set here the last used channel (meaning scan wire)!
Lastwire=1;

% ------------------------------------------------------------------------
% --------    End of the user adjustable part!    ------------------------
% ------------------------------------------------------------------------

% Ask for the input file.
str='\nPlease type name (and path) of the "SIM DL363" file to analyse: ';
Infile=input(str,'s');

% Set the options for "sim2mat".
Options=[Options ' M=' int2str(Lastwire) ' O=event.m I=' Infile];

% Call "sim2mat".
Options=['!sim2mat ' Options];
eval(Options);

% Load the macro produced by "sim2mat".
event;

% *** Outer part only to control the user I/O ***
% First principal loop until user quit (or break).
p=1;
while (p~=0);
   q=0;
   if ((p<0)|(p~=round(p)));
      disp('As input are allowed only positive integers!');
   end;
   if (p>Lastwire);
      disp('Such a large channel was not used in this session!');
   end;
   str='Please choose the channel number to be worked up';
   str=[str ' (or "0" to quit): '];
   pstring=input(str,'s');
   p=sscanf(pstring,'%f',1);
```

```
if (isempty(p)==1);
   p=0.5;  % This is not valid to produce the error dialogue.
end;
if ((p>0)&(p==round(p))&(p<=Lastwire));

   % Secondary loop to choose the hit number.
   while (q<=NumberOfHits);
      qsave=q;  % Save q for later use.
      str='\nPlease choose the hit number to show.\n"Return" chooses';
      str=[str ' the next one of the actual channel: '];
      qstring=input(str,'s');
      q=sscanf(qstring,'%f',1);
      if (isempty(q)==1);
         q=0;
      end;
      if ((q<0)|(q~=round(q)));
         disp('As input are allowed only positive integers!');
      end;
      if ((q>NumberOfHits)&(q==round(q)));
         disp('This event has not so much hits!');
      end;
      if ((q==round(q))&(q<=NumberOfHits)&(q>=0));
         if (q==0)
            % Search the next hit in the chosen channel.
            q=qsave+1;
            while((q<=NumberOfHits)&(Wirenumber(q)~=p));
               q=q+1;
            end;
          end;
          if (q<=NumberOfHits);
            % Set the chosen channel convenient for the chosen hit.
            p=Wirenumber(q);
            str=['Selected is hit ' int2str(q) ' (in channel '];
            str=[str int2str(p) ') ...'];
            % There were some MATLAB problems with newline...
            disp(' ');
            disp(str);

            % *** Central part to call the macros and show results ***
            % Prepare the graphic output and show the raw hit.
            clf;
            subplot(2,2,1);plot(Hit(q,:));
            str=['(a)  Raw hit number ' int2str(q)];
            title(str);
            xlabel('Digitisation steps');
            ylabel('Amplitude');
            subplot(2,2,2);
```

```
         % Write q to the disk for use with "polish.m".
         fid = fopen('hitselected.txt','w');
         fprintf(fid,'%u\n',q);
         fclose(fid);

         %Save all variables used here as global.
         save global p q NumberOfHits Wirenumber Lastwire Hit;

         % Call "polish.m".
         polish;
         % Write title.
         title('(b)  Polished and adjusted');

         % Ask the user if it is a hit...
         str='\nIs this a hit to be worked up ("y") or only';
         str=[str ' noise ("n")? '];
         qstring='a';   % "a" is not "y" or "n"...
         while ((qstring~='y')&(qstring~='n'));
            qstring=input(str,'s');
         end;
         if (qstring=='y');

            % Switch to the next window.
            subplot(2,2,3);

            % Call "decon.m".
           decon;
            % Write title.
            title('(c)  Deconvolution result');

            % Ask the user for a filename to save the results...
            str='\nPlease choose a filename (and path) to save the';
            str=[str ' deconvolution result\nof this hit: '];
            qstring=input(str,'s');
            % and save it...
            qstring=['!cp signal.txt ' qstring];
            eval(qstring);
            disp(' ');
         end;

         % Restore the global variables.
         load global;

      end;
   end;
end;
disp(' ');
disp('No more hits found for this channel!');
```

```
      disp(' ');
   end;
end;

% Delete all temporary files.
!rm event.m;
!rm polished.txt;
!rm signal.txt;
!rm hitselected.txt;
!rm global.mat;
```

C.2 The macro *polish*

```
% <polish.m> Polishes the received signals.
% ----------------------------------------
% <event.m> is the file with all the hits to investigate
%           (created by "sim2mat").
% <hitselected.txt> This input contains the selected hit number to be
%                   polished in this run of the macro.
% <polished.txt> This output is a MATLAB vector of the polished hit with
%                50 elements, for the further use by "decon.m".

% ------------------------------------------------------------------------
% --------    No user adjustable part here!    ---------------------------
% ------------------------------------------------------------------------

clear;

% Load the event-file with all hits.
event;

% Read the variable "Hitselected".
fid=fopen('hitselected.txt','r');
Hitselected=fscanf(fid,'%f',1);
fclose(fid);

% Read the whole hit out of the event-file.
for k=1:Hitlength(Hitselected);
   Signal(k)=Hit(Hitselected,k);
end;

% Fill a vector with less than 60 elements up to 60 ...
% Avoid this by setting the sim2mat option L=60 or greater!
if ((Hitlength(Hitselected)>19)&(Hitlength(Hitselected)<60));
   % Fill the rest of the vector with the medium of last 10 hit values.
   Ptest=0;
   for k=(Hitlength(Hitselected)-9):Hitlength(Hitselected);
```

```
      Ptest=Ptest+Signal(k);
   end;
   Ptest=Ptest/10;
   for k=(Hitlength(Hitselected)+1):60;
       Signal(k)=Ptest;
   end;
   Hitlength(Hitselected)=60;
end;
if (Hitlength(Hitselected)<20);
   % Fill the rest of the vector with zeroes. (The vector is only noise.)
   for k=(Hitlength(Hitselected)+1):60;
       Signal(k)=0;
   end;
   Hitlength(Hitselected)=60;
end;

% Search for the start of hit using a time gate of 20 values.
k=1;
Majority=0;   % (flag)
while ((Majority==0)&(k<Hitlength(Hitselected)-18));
   for m=0:19;
      if (Signal(k+m)>10);
         Majority=Majority+1;
      end;
   end;
   if (Majority<11);
      Majority=0;
   end;
   k=k+1;
end;

% Move the start of hit to the beginning.
for m=k:Hitlength(Hitselected);
   Signal(m+1-k)=Signal(m);
end;
Hitlength(Hitselected)=Hitlength(Hitselected)-k+1;

% Fill a vector with less than 60 elements up to 60 ...
% This is the same routine as above!
if ((Hitlength(Hitselected)>19)&(Hitlength(Hitselected)<60));
   % Fill the rest of the vector with the medium of last 10 hit values.
   Ptest=0;
   for k=(Hitlength(Hitselected)-9):Hitlength(Hitselected);
      Ptest=Ptest+Signal(k);
   end;
   Ptest=Ptest/10;
   for k=(Hitlength(Hitselected)+1):60;
      Signal(k)=Ptest;
```

```
   end;
end;
if (Hitlength(Hitselected) < 20);
   % Fill the rest of the vector with zeroes. (The vector is only noise.)
   for k=(Hitlength(Hitselected)+1):60;
      Signal(k)=0;
   end;
end;

% Construct the vector to calculate with ...
Outsignal=zeros(1,80);
for k=1:60;
   Outsignal(k+10)=Signal(k);
end;
Ptest=0;
for k=61:70;
   Ptest=Ptest+Outsignal(k);
end;
Ptest=Ptest/10;
for k=71:80;
   Outsignal(k)=Ptest;
end;

% Polishing of the signal by eliminating the spikes.
x=1:80;
% Compare with a polynomial function of 10th grade.
P=polyfit(x,Outsignal,10);
Polished=polyval(P,x);

% Replace the original data by the fitted data if more than 15 away
% from fit. (The first 10 values are outside of the region of interest
% now, and in the following nothing is calculated with them.)
for k=11:70;
   if ((Outsignal(k)>(Polished(k)+15))|(Outsignal(k)<(Polished(k)-15)));
      Outsignal(k) = Polished(k);
   end;
end;

% Recalculating the original amplitudes. They were changed by the
% Struck system to increase the energy sensitivity for high amplitudes.
for k=11:70;
   Outsignal(k)=Outsignal(k)/(256-0.75*Outsignal(k));
end;

% Scaling the signal to a tail finish of 1 (in the region of interest).
Ptest=Outsignal(70);
% No scaling the signal was noise.
if Ptest==0;
```

```
   Ptest=1;
end;
for k=11:70;
   Outsignal(k)=Outsignal(k)/Ptest;
end;

% Moving left until start of hit (in the region of interest).
% This is a fine adjustment because the coarse selection of the hit
% (above) is not very exact. Maximum distance for move is 10 here.
m=0;
while ((Outsignal(19)<0.2)&(m<10));
   for k=11:69;
      Outsignal(k)=Outsignal(k+1);
   end;
   m=m+1;
end;

% Eliminating negatives by suppressing all what is lower than 0.
% (in the region of interest...)
for k=11:18;
   if (Outsignal(k)<0);
      Outsignal(k)=0;
   end;
end;

% Output of the result.
plot(Outsignal(11:60));
xlabel('Digitization steps');
ylabel('Amplitude [normalized]');

% Write the result to disk.
fid = fopen('polished.txt','w');
fprintf(fid,'%5.4f\n',Outsignal(11:60));
fclose(fid);
```

C.3 The macro *decon*

```
% <decon.m> Deconvolution of a previously polished function.
% ----------------------------------------------------------
% <polished.txt> is a MATLAB vector of the function with 50 elements.
%                It has to start with the hit, normally it is produced
%                by the macro <polish.m>!
% <signal.txt> contains the restored signal. It has 50 ASCII values, each
%              value in one line. The further processing of the result
%              lies in the hands of the user. (The analyse macro asks the
%              user for a filename to change and prevent overwriting it.)
```

```
clear;

% The constants for the response function have to be set! (Delta,T,Tau)
% For electrons and the "Koeln" amplifier: Delta=0.028,T=77,Tau=12.
% For holes and the same amplifier: Delta=0.027,T=78,Tau=12.
% For other situations: try it...
Delta = 0.028;
T     = 77;
Tau   = 12;

% The constant "Filter" sets a low pass filter. The electronics support up
% to 30 MHz. This results in a setting of Filter=47. It does not make
% sense to set it to a higher value. In reality (noisy signals) it has to
% be set to a lower value. The allowed settings are from 2 to 76 !!!
Filter = 23;

% The constant "Applybad" sets the number of "bad" frequencies to be added
% to the "good" ones using the constraint of finite extent. The "good"
% ones are set by "Filter". The maximum number to be set is Applybad=50.
% Attention: The sum of "Filter" and "Applybad" has to be lower or equal
% than 76 !!! Of course adding too much noise destroys the original
% function. Be careful and view how it looks like with Applybad=0. Then
% try 1,2,3,...
Applybad = 28;

% ------------------------------------------------------------------------
% --------    End of the user adjustable part!    ------------------------
% ------------------------------------------------------------------------

Outsignal=zeros(150,1);
fid=fopen('polished.txt','r');
Outsignal=fscanf(fid,'%f',50);
fclose(fid);

% Differentiate the signal. Take signals only until the first point of
% intersection with zero.
k=1;
while (k<50);
   if (Outsignal(k+1)>Outsignal(k));
      Outsignal(k)=Outsignal(k+1)-Outsignal(k);
      k=k+1;
   elseif (k>8);
      % g is a marker for the length of the part with added zeroes.
      % (to be used later ...)
      g=50-k;
      for m=k:50;
         Outsignal(m)=0;
         k=50;
```

```
      end;
   else;
      Outsignal(k)=Outsignal(k+1)-Outsignal(k);
      k=k+1;
   end;
end;

% Move Outsignal into the middle, starting with 51.
for k=1:50;
   Outsignal(k+50)=Outsignal(k);
   Outsignal(k)=0;
end;

% There are problems with the matrix dimension if not explicitly set.
% (MATLAB ignores the definition above and uses a shorter vector...)
Outsignal(150)=0;

% Construct a response function. (Directly at the right place...)
Respsignal=zeros(150,1);
% Calculate the phase angle for a start point with gradient zero.
Phimat=0:0.001:6.828;
y=abs(Delta.*sin(Phimat)-2.*pi./T.*cos(Phimat)+1./Tau);
[k,Phipos]=min(y');
Phi=Phimat(Phipos);
% Create the Respsignal vector.
for t=1:49;
   Respsignal(t+101)=(1+exp(-Delta*t*10)*sin(2*pi*t*10/T+Phi-pi/2));
   Respsignal(t+101)=Respsignal(t+101)*(1-exp(-t*10./Tau));
end;

% Differentiate the response function.
for k=101:149;
   Respsignal(k)=Respsignal(k+1)-Respsignal(k);
end;
Respsignal(150)=0;

% Calculate the right end of the signal.
[k,Respmax]=max(Respsignal(51:100)');
g=g-Respmax;
if (g<0);
   g=0;
end;

% The Fourier transpose.
Outfourier=zeros(150,1);
Respfourier=zeros(150,1);
% Attention: Under MATLAB the inverse Fourier transform is
% normalized but the Fourier transform not !!!
```

```
Outfourier=fft(Outsignal)./150;
Respfourier=fft(Respsignal)./150;

% Avoid zeros in the response components because later follows a division.
% The corresponding signal components are set to zero to avoid
% miscalculation.
for k=1:150;
   RT=real(Respfourier(k));
   IT=imag(Respfourier(k));
   if ((RT<0.00001)&(RT>=0));
      RT=0.00001;
      Outfourier(k)=0;
   elseif ((RT>-0.00001)&(RT<0));
      RT=-0.00001;
      Outfourier(k)=0;
   end;
   if ((IT<0.00001)&(IT>=0));
      IT=0.00001;
      Outfourier(k)=0;
   elseif ((IT>-0.00001)&(IT<0));
      IT=-0.00001;
      Outfourier(k)=0;
   end;
   Respfourier(k)=RT+i*IT;
end;

% Low pass filter for lower of the "Bad" coefficient.
for k=Filter:152-Filter;
   Outfourier(k)=0;
end;

% The deconvolution division.
% The MATLAB division conjugates, so another conj is necessary here.
Infourier=Outfourier./conj(Respfourier);

% The inverse transformation.
Infilt=ifft(Infourier);

% Find the result for the "good" frequencies.
for k=1:150;
   Infre=real(Infilt);
   Infim=imag(Infilt);
end;

% Constraint of finite extent.
% ***************************

% Use the constraint only if wanted.
```

```
if (Applybad>0);

% Search proper values v(t) to fulfil the constraint.
% Build up the matrix.
A=zeros(100+g,100+g);
b=zeros(100+g,1);
for k=51-g:150;
   for m=0:49+g/2;
      A(2*m+1,k-50+g)=cos(2*pi/150*(m+Filter)*k)*150;
      A(2*m+2,k-50+g)=sin(2*pi/150*(m+Filter)*k)*150;
      b(2*m+1)=b(2*m+1)-Infre(k)*cos(2*pi/150*(m+Filter)*k);
      b(2*m+2)=b(2*m+2)-Infim(k)*sin(2*pi/150*(m+Filter)*k);
   end;
end;
% Solve the linear equation system.
% Because of stability problems the matrix is not solved directly
% but using a QR factorisation before the "left matrix division".
[Q,R]=qr(A);
v=Q'*b\R;

% Recalculate proper coefficients for the "bad" frequencies.
% Build up the matrix.
A=zeros(100+g,2*Applybad);
for k=51-g:150;
   for m=0:Applybad-1;
      A(k-50+g,2*m+1)=cos(2*pi/150*(m+Filter)*k);
      A(k-50+g,2*m+2)=sin(2*pi/150*(m+Filter)*k);
   end;
end;
% Solve the linear equation system.
b=v'\A;

% Add as much coefficients as wanted by the setting of "Applybad".
for k=0:Applybad-1;
   Infourier(k+Filter+1)=b(2*k+1)*150+i*b(2*k+2)*150;
   Infourier(151-k-Filter)=b(2*k+1)*150+i*b(2*k+2)*150;
end;

% End of the principal loop (use the constraint or not).
end;

% End of the constraint algorithm.
% ********************************

% Calculate the filtered input signal.
Infilt=zeros(150,1);
Infilt=real(ifft(Infourier));
```

```
% Destroy the artifacts.
[k,Pos]=max(Infilt(1:50));
k=Pos;
while ((Infilt(k)>0)&(k<50));
   k=k+1;
end;
while (k<51);
   Infilt(k)=0;
   k=k+1;
end;
k=Pos;
while ((Infilt(k)>0)&(k>1));
   k=k-1;
end;
while (k>0);
   Infilt(k)=0;
   k=k-1;
end;

% Scale the amplitude to area one.
Insignal=zeros(50,1);
Middle=0;
for k=1:50;
   Middle=Middle+Infilt(k);
end;
for k=1:50;
   Insignal(k)=Infilt(k)/Middle;
end;

% Output of the result.
plot(Insignal);
xlabel('Digitization steps');
ylabel('Amplitude [normalized]');

% Write the result to disk.
fid = fopen('signal.txt','w');
fprintf(fid,'%5.7f\n',Insignal);
fclose(fid);
```

List of Figures

Bibliography

[1] Helmut Kopka. *LaTeX – Band 1, Einführung*. Addison-Wesley (Deutschland) GmbH, Bonn, Reading(MA), 2. ed. 1996

[2] Stephen Prata. *C++, Einführung in die objektorientierte Programmierung*. te-wi Verlag, München, 6. ed. 1992

[3] All reachable manuals for the used products of the following companies: WindRiver (VxWorks), Dr. B. Struck (Hardware DL...), Motorola (Hardware MVME...), National Instruments (LabVIEW).

[4] A. Vladimirescu. *The Spice book*. John Wiley & Sons Inc., New York, 1994

[5] Eva Pärt-Enander, Anders Sjoberg, Bo Melin, Pernilla Isaksson. *The MATLAB handbook*. Addison Wesley Longman Ltd., Edinburgh Gate, Harlow, Essex, 1996

[6] Mario Vascon. *Dispositivi elettronici e sistemi analogici*. Cooperativa Libraria Editrice Universitaria di Padova, Padova, 1997

[7] Glenn F. Knoll. *Radiation detection and measurement*. John Wiley & Sons Inc., New York, Chichester, Brisbane, Toronto, Singapore, first edition 1979 and second edition 1989

[8] G. Bertolini, A. Coche. *Semiconductor detectors*. North – Holland Publ. Co., Amsterdam, 1968

[9] B. Philhour, S. E. Boggs, J. H. Primbsch, A. S. Slassi-Sennou, R. P. Lin, P. T. Feffer, S. McBride, F. S. Goulding, N. W. Madden, R. H. Pehl. *Simulations of pulse shape discrimination (PSD) techniques for background reduction in germanium detectors*. Published in *Nuclear Instruments and Methods in Physics Research*, section A, volume 403. Elsevier Science B.V., Amsterdam, 1998

[10] Alan V. Oppenheim, Alan S. Willsky, Ian T. Young. *Signals and systems*. Prentice-Hall International Inc., London, 1983

[11] Peter A. Jansson, Samuel J. Howard. *Deconvolution: with applications in spectroscopy*. Academic Press Inc., Orlando, London, 1984

[12] Thorsten Kröll, Marco Bellato, Dino Bazzacco. *Pulse Shape Analysis for MARS*. LNL Annual Report 1997 (page 171), Legnaro, 1998

[13] TMR User Meeting. *Gamma-Ray Tracking Detectors*. Department of Physics – University of Padova, Padova, Oct. 1998

Thanks

To conclude I wish to thank all the people who helped me a great deal in many different ways, also those who are not mentioned below.

Prof. Dr. M. Müller-Veggian took care of the thesis work and was responsible for my decision to go to Italy with the ERASMUS exchange programme. Dr. D. Bazzacco corrected my drafts of this text and was always interested in my progress with the work and also the personal manner of my stay. Dr. M. Bellato also helped me with my experiments and with learning the programming of "his" equipment I used. Prof. Dr. S. Lunardi was the Italian responsible for the exchange programme and helped me to find this project. Also there were many friendly and helpful people in the labs. In Jülich Prof. Dr. Hoyler substituted Dr. Bazzacco for the final examination.

In a private manner I want to thank my father Detlev and my deceased mother Inge, who supported my studies the whole time and my brother Heiko, who is very close to me. Finally there are all my friends and God, who helped me in difficult situations.