Bachelor Thesis

Emulation of a Geiger-Mueller-Tube
on a XILINX FPGA board

by
Bernd Pohlmann

Bernd Pohlmann

Title of the paper

Emulation of a Geiger-Mueller-Tube on a XILINX FPGA board

Keywords

Abstract

Bernd Pohlmann

Titel der Bachelorarbeit

Emulation eines Geiger-Müller Zählrohrs auf einem XILINX FPGA Board

Stichworte

Kurzzusammenfassung

Acknowledgments

Pareigis, Korf, Meisel
JP

Contents

List of Figures

1 Introduction

The history of emulation goes back to 1957. The term was introduced by IBM[1] for the IBM 709 computer being able to behave like the IBM 704 so that the older model applications could be run on the new system. This was done by **imitation**.
So an emulator emulation, simulation, calibration

1.1 Motivation

What made me chose this project, radioactivity, dosimetry,..danger...

1.2 Goals

What is to be achieved withing this project...the focus lies on hardware software co-design, fully functional GM-Tube emulator.....

1.3 Structure

first bla, then blub

[1] http://www.ibm.com

2 Analysis

"Nothing in life is to be feared, it is only to be understood.
Now is the time to understand more, so that we may fear less." *Marie Curie*

2.1 Requirements

The goal of this bachelor thesis is a fully functional Geiger-Mueller-Tube[1] emulator. Therefore a system consisting of hard- and software has to be created which on one hand simulates events caused by radioactive decay and secondly is able to measure those events. The emulator has to function like a real-life Geiger-Mueller-Tube. A small GUI[2] driven by via standard I/O[3] devices should provide an easy possibility to configure the behavior of the radiation generator and the GM-Tube. The programmable parameters include:

- seed for the radiation generator
- mean time of occurring events: $1\mu s \leq$ mean time $\leq 10s$
- detector dead-time: $5\mu s \leq$ detDT $\leq 150\mu s$
- and system dead time: sysDT = detDT + ($3\mu s \leq$ recoveryTime $\leq 50\mu s$)
- System dead time will grow wider when a new particle hits the detector between detDT and sysDT. Shortest pulse widening(pulseW) starts at 1/3rd of the detDT when next particle hits the tube right after detDT and will grow linearly up to full detDT.

Also the system has to be designed according to following conventions:

- Simplicity
  The system should be designed as simple as possible and as complex as necessary.
- Expandability
  The design of the system should be in a way that it allows further addition of functions in a reasonable amount of time and effort.
- Portability
  As far as possible the system should be designed in a way as hard- and software independent as possible. I.e. the used FPGA[4] should be exchangeable.
- Real-time
  The GM-Tube emulator is to be created as a soft real-time system.

[1] in further documentation referred to as GM-Tube
[2] Generic User Interface
[3] Input/Output
[4] Fast Programmable Gate Array

2.1.1 Predefined test cases

For proving the system to work accordingly to its equivalent measurement device several test scenarios are defined.

- Speed: Is the radiation generator able to create the events on demand?
  With a minimal mean event time of 1μs it is mandatory that the generator produces at least one events per μs.
- Scale: Does the scale of generator output fulfill the required interval spectrum?
  The applied hardware provides a bit width of 32bits. The full range is to be used and the events need to be in a random exponential distribution representing an actual radiation source.
- Performance: Does the system show the correct number of output pulses?
  With a predefined array of input for the occurring events it needs to be shown that the system gives the correct number of output pulses at the correct times. Event loss is not tolerated.

2.2 Radioactivity

Radioactivity is the common phrase for radioactive decay which describes the ambition of unstable atomic nucleus to transform into a stable state by emitting ionizing[5] particles *and* radiation. An unstable nucleus is characterized by an uneven ratio of protons to neutrons and it remains radioactive until the ratio is balanced. The decay happens spontaneous and works exothermic[6]. The stable state can be reached by one simple decay or within a sequence of reorganizations. During one decay multiple types of radiation are emitted.[2]

2.2.1 History

1896 Antoine Henri Becquerel discovered that it is possible to blacken photographic plates with uranium salt. He showed that this new radiation (just one year after Wilhelm Conrad Roentgen described the phenomenon of x-rays) is able, to pierce through opaque materials and ionize air.
Two years later Marie Curie and her husband found three other naturally occurring elements, thorium and the two much stronger radioactive elements, radium and polonium. Ernest Rutherford, 1899, made several penetration tests and separated two types of radiation. Further tests within magnetic fields showed that the particles have different polarity and that there was a third type of radiation which is not influenced my magnetism. He named the radiation types Alpha-, Beta- and Gamma radiation.
In 1933 Iréne and Frédéric Joliot-Curie made the first successful attempt to create artificial radioactive elements.[1]

2.2.2 Radiation types

During the process of radioactive decay six particle types are emitted. For simplicity I will only refer to the three most common types α, β and γ in further documentation.

[5] with an energy $\geq$ 5eV
[6] after initial triggering the process runs without any further need for external energy input

- α radiation: This decay produces one plus helium-4 nucleus which consists of two protons ans two neutrons. the contained charge is positive due to the surplus of protons. This type of radiation has the highest energy, but in consequence of the mass the lowest penetration depth in materials. Another consequence of the relatively large mass is the ambition to interact with other atoms and lose their energy, so their forward motion is effectively stopped within a few centimeters of air.[4]

$$^{238}_{92}\mathrm{U} \rightarrow ^{234}_{90}\mathrm{Th} + ^{4}_{22}\mathrm{He}^{2+} \textbf{ or simplified } ^{238}\mathrm{U} \rightarrow ^{234}\mathrm{Th} + \alpha$$

- β radiation: There are two types of β decays.
  β^- decay, where an electron is emitted:

$$^{137}_{55}\mathrm{Cs} \rightarrow ^{137}_{56}\mathrm{Ba} + \mathrm{e}^- + \bar{\upsilon}_e$$

  β^+ decay, where a positron[7] is emitted:

$$^{22}_{11}\mathrm{Na} \rightarrow ^{22}_{10}\mathrm{Ne} + \mathrm{e}^+ + \mathrm{v}_e$$

  β^- emission is the most common kind of radioactive decay.

- γ radiation describes a highly energetic electromagnetic radiation. The typical frequency is about 10^{19} Hz with a wavelength of 10^{-12} meters and therefore energies above 100keV.

2.2.3 Radioactivity as a natural phenomenon

Radioactivity is omnipresent. The earth is exposed to cosmic radiation, natural radiation from elements like Potassium emit radiation. The natural element Uranium is used as an energy source in nuclear plants and even a positioning device like Radar, which is a high frequency application, generates radiation.
Usually the radioactive decay runs under predictable circumstances and without endangering life. It follows certain probability theorems like the Poisson distribution. There are also methods to control the emission of radioactivity which are used to operate a nuclear power plant in a safe way.
The great danger of radioactivity lies in the uncontrolled chain reaction a so called critical mass[8] of radioactive material is capable of. This leads to an abrupt discharge of radioactivity and huge amount of kinetic energy is set free. In the moment of a so called nuclear fission a huge amount of alpha, beta and gamma rays are emitted, which poses a serious threat to the environment.

2.2.4 Radiation detection and measurement[3]

First of all: Ionizing radiation emitted by radioactive decay is a danger to all organic life. Therefore methods are needed to measure the amount of radiation in an environment to take actions to prevent organisms from harm, a fact which the scientists who discovered radioactivity in the first place were not aware of.

[7] positive charged electron
[8] the smallest amount of fissile material needed for a sustained nuclear chain reaction

2.2.5 Danger of radioactivity

The radiation dose a living being can be exposed to without being injured, varies between species and even for humans the tolerated dose depends on individual genes. The human cell structure is based on H_2O and ionizing radiation is able to modify or even destroy this structure. The hazardousness of the three radiation types depends on how they interfere with the body. Alpha particles as mentioned only have a range of a few centimeters and even skin can prevent it from penetrating. But if it gets directly into the organism through inhaling radioactive dust even a few milligrams can be fatal. Beta and Gamma rays have a much higher penetration potential but are not that dangerous in small doses.

(a) Ionizing radiation hazard symbol

(b) The radiation symbol

Figure 2.1: International warning signs

Most common direct results of a radiation overdose contain diseases like leukemia and many more types of cancer. Also the genes can be modified. If that is the case a reproduction cycle may develop certain diseases and/or deformations.

2.2.6 Units

The rate of decay of a radioactive isotope is given by the fundamental law of radioactive decay

$$\frac{dN}{dt}|_{decay} = -\lambda N$$

where N is the number of radioactive nuclei and λ is defined as th decay constant. The Unit is *curie* Ci, defined as $3.7\times\ 10^{10}$ disintegrations per second, which equates to the activity of exact one gram of pure ^{226}Ra. Today the used unit is Becquerel

$$1Bq = 2.703 \times 10^{-11}Ci$$

The second directly measurable value is the emitted energy in eV. It is defined as the kinetic energy gained by an electron by its acceleration through a potential difference of 1 volt. The SI unit is the *joule*. More convenient when dealing with radiation energies is the sub multiple femtojoule (fJ).

$$1fJ = 6.241 \times 10^{3}eV$$

2.2.7 Measurement

To measure radiation of any kind a particle must undergo an interaction with an electromagnetic field. The energy of an idealized *single* particle is considered instantaneous due to the time of appearance being so short. So in most detectors a certain amount of interactions is needed to trigger an event which is followed by an electric discharge. This is described by the formula

$$\int_0^{t_c} i(t)dt = Q$$

where t_c represents the charge collection time, *i(t)dt* the charge over time and Q the total amount of charge generated in one specific interaction.
In reality many quanta of radiation interact over a period of time(Figure 2.2). At this point it is important to mention that the arrival of a radiation quanta as well as the time intervals between successive current pulses are randomly distributed.

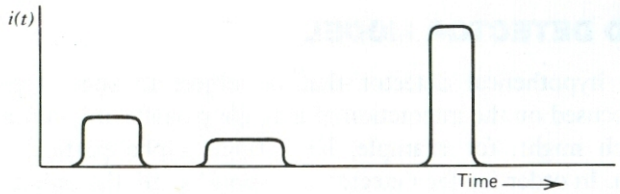

Figure 2.2: typical radiation distribution[3]

Operation Modes

Depending on the purpose of the detector there are three different operation modes: *Current*, *Mean Square Voltage(MSV)* and *Pulse* Mode. Due to this thesis mainly deals with the GM-Tube which acts in current or more often in pulse mode I will not describe MSV in detail. MSV is used in a mixed radiation environments where all kinds of radioactivity are present.
In current mode it is assumed that the response time of the detector is constant. So the recorded signal from a series of events will be a time-dependent current given by

$$I(t) = 1/T \int_{t-T}^{t} i(t')dt'$$

The longer the integration time T the less the statistical fluctuations of the signal will be. But on the other hand the response to rapid changes in the rate or nature of the radiation interactions will slow down. As a result of these conclusions the average current is given by following equation

$$I_0 = rQ = r\frac{E}{w}q$$

where

r = event rate
Q = Eq/w = charge produced for each event
E = average energy deposited per event
w = average energy required to produce a unit charge pair(e.g., electron-ion pair)
q = 1.6 $\times 10^{-19}$C

Current mode is used in many detectors when the event rates are high.
Pulse mode on the other hand is standard operation in the field of radiation spectroscopy[9] where every radiation quanta is counted and the amount of energy of a single quanta is analyzed. Most applications for event counting operate in pulse mode because due to the internal design the amplitude of the signal pulse is directly proportional to the corresponding charge generated within the detector. The resulting simple expression

$$V_{max} = \frac{Q}{C}$$

leads to the conclusion that every pulse is a direct result of a single event within the detector.

Dead Time

Almost all detector systems need a minimum amount of time between two pulses in order to fetch them as two separate events. This gap is needed by the process and/or the electronic within the measurement device. It is usually called *dead time* because the detector is not able to recognize events during this period.
Though there is no possibility of counting events during dead time the system may respond to it anyhow. If a detector is paralyzable it will show longer output pulses when events occur during dead time, which leads to one permanent pulse when the rate of events is equal or higher than the dead time. Non paralyzable systems will simply *ignore* non countable events.(Figure) No matter how the system responds to high event rates it

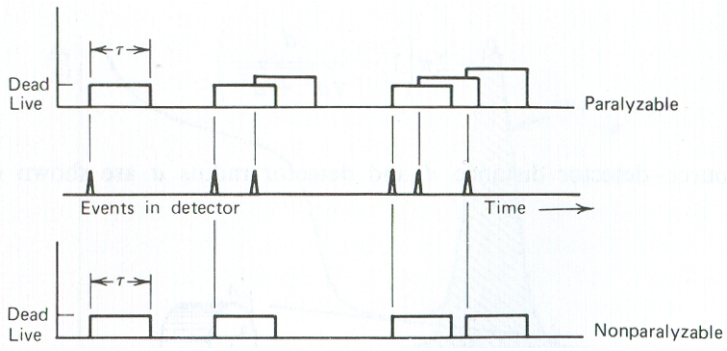

Figure 2.3: illustration of the two dead time behaviors[3]

is necessary to correct the output of the system. Therefore it is mandatory to know the dead time τ. A part of it may be the response time of an electronic circuit which is simple to calculate but most often it is not known and dependent on the operating conditions.
In this case τ will be calculated by the *two-source method*. The method uses the fact, that the counting loss is non linear. So observing the counting rate from two sources individually and in combination results in a counting discrepancy which allows it to calculate the dead time.

[9] A technique used in physical and analytical chemistry for the identification of substances through the spectrum emitted from or absorbed by them

2.3 Geiger-Mueller-Tube

The GM-Tube or GM-Counter is one of the oldest radiation detector types in existence. It was developed by Geiger and Mueller in 1928 and due to the simplicity, low cost and ease of operation it is still the most common tool of radiation measurement.

Figure 2.4: schematic of a GM-tube

2.3.1 Operating Mode(*simplified)*

The GM-Tube is a gas-filled detector based on ionization within an electric field. The primary modes of interaction involve ionization and excitation of gas molecules along the track of a radioactive particle within the gas chamber. Simplified one can imagine the process as a chain reaction where a radioactive particle passes the gas chamber and on its way transfers energy to the gas molecule(s) so that they become excited[10] and on the other hand create ion pairs by freeing electrons which lead to further *excitement* and ionization. Because of the polarity of the electrons(negative) and ions(positive) which move to the electrodes a current emerges. So a single particle can be source of a series of avalanches within the tube.
The energy set free by one avalanche is influenced by the electric field within the gas chamber; the higher the field energy the higher the multiplication initiated by a single event. When a certain level of ionization is reached the discharge is initiated. This usually takes about one microsecond. The discharge stops when the concentration of positive ions reaches a critical state in which they change the polarity of the electric field. This critical concentration is almost constant and depends on the strength of the electric field. As a conclusion the pulse amplitude is fix which makes any interpretation of a coherence between the energy of the radioactive quantum that created the pulse and the amplitude impossible. So the GM-Tube can only be an event *counting* device. This stands in direct contrast to the fact that the GM-Tube is operated in pulse mode.
The fill gas of the chamber is usually a noble gas[11] like helium or argon. Although it is possible to detect Alpha and Beta radiation with it the GM-Tube is most often used to measure Gamma rays. Its application amongst other things lies in radiation dosimetry[12].

[10]reaching an energy level higher than ground level
[11]a non molecular occurring gas with almost no chemical reactivity
[12]Method for measuring a radiation dose over time

2.3.2 Schematic

The schematic assembly of a GM-Tube is shown in Figure 2.5.

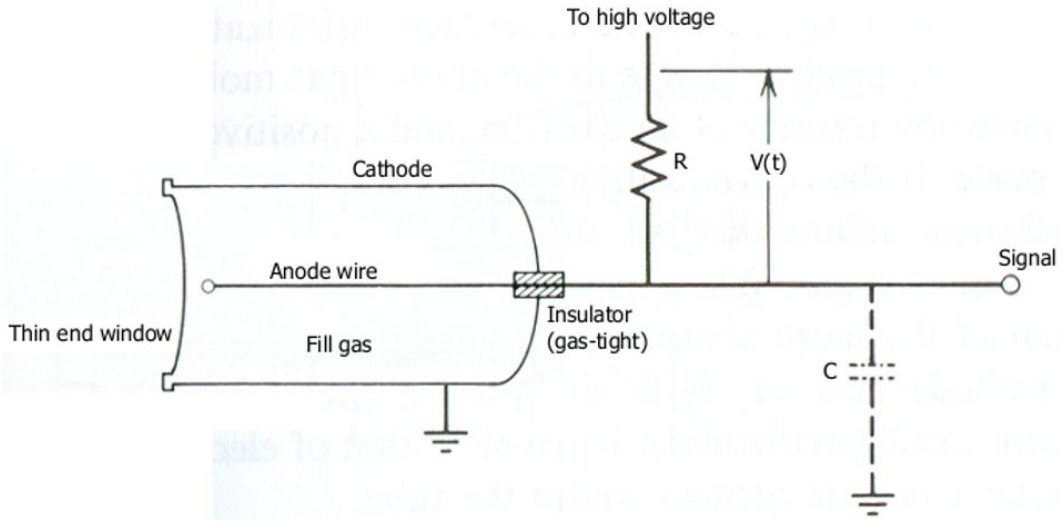

Figure 2.5: a standard GM-Counter

2.3.3 Dead Time Behavior

For creating an emulator the main focus lies on the dead time behavior of the detector, because the behavior is responsible for the system flow design.
As mentioned in subsection 2.3.1 the tube needs a time to recover after a successful discharge. The dead time consists of two constants. First, the time the tube needs to be able to produce a new charge at all, which is given by the *idleness* of the ions, has to pass. Within this time no further particle or gamma ray will be recognized. After a certain amount of ions returned from the cathode the tube is ready to produce a new discharge that will lead to an output pulse. This pulse will have a lower amplitude than a full discharge but will lead to an extension of the pulse that is still high from the last discharge. Only when the electric field is fully recovered a *new* pulse will be generated. The regular dead time of a GM-Tube lies between 50-100μs, the *resolving* time however depends on what amplitude the counting circuit interprets as a full discharge(Figure 2.6). This leads to the conclusion that the GM-Tube is paralyzable.

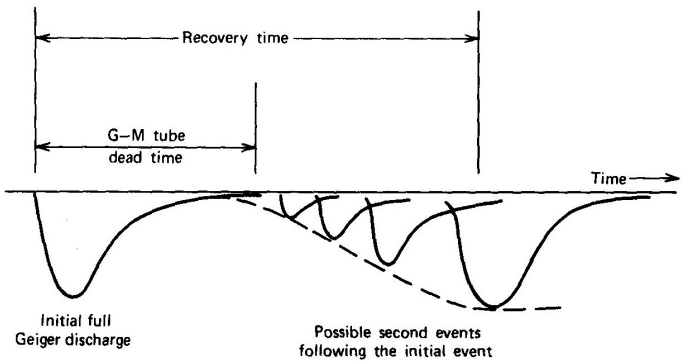

Figure 2.6: dead time behavior of a GM-Tube

2.3.4 Counting Efficiency

Because a single ion pair formed within the tube can trigger a full Geiger discharge, the counting efficiency for particles passing through the tube window is 100%. However because of the window's thickness alpha particles may be absorbed. On the other hand beta rays can be measured if the window thickness is not a significant fraction of the electron range.
It is also possible to measure gamma radiation. Therefore the interaction has to take place close enough to the inner surface of the tube so that the secondarily created electron reaches the fill gas and can so produce ions. This circumstance can be drastically improved by choosing the right tube material. If for example Bismuth is used in a combination with a fill gas like Xenon or Krypton the efficiency for low-energy gamma rays gets close to 100%.

2.4 Embedded Development

Due to the fact that the specialization topic of my studies is embedded systems and this project manifests the possibilities and difficulties of embedded development in this section I will describe some ideas behind the concept.

2.4.1 Embedded Systems

An embedded system is a combination of hard- and software designed to perform up to a few dedicated functions. The components are build *into* a technical context, like e.g. a health monitor, a mobile phone or ABS[13] system in a car, in most cases invisible to the end user. On the other hand a common personal computer due to its general purpose, is not called *embedded*.
In embedded systems the hardware resources are very limited and customized to answer the specified purpose. Front end user interaction may be handled by a custom I/O device (heart-rate monitor, volt-meter etc.) or via a serial port connected to standard hardware. Often the embedded system performs as a part of a distributed system that works without any user interaction like most driving assistant systems in a car.

2.4.2 Embedded Programming

As a response to the dedicated functions almost all embedded systems have a custom made operating system they run on, in many cases it is just a super loop[14]. The kernel, if present, only implements the very basic functions like memory management or interrupt handling.
So it is obvious that the programming language for such a system must operate very close to the hardware. Examples for languages that fulfill this requirement are Assembler, C/C++ and Pascal. The program instructions written for embedded systems are referred to as firmware, and are stored in read-only memory or Flash memory chips.

[13] anti-lock braking system
[14] one big endless loop construct that handles the tasks of a software

2.4.3 Hardware Software Codesign

In every embedded system the trade-off between hard- and software is substantially for designing a solution for a specific task. There are two well known major approaches how to build a system from scratch. First method used is top-down, starting at high abstraction layers and digging into the deeps of the single execution of a command. It basically starts at defining what hardware would be appropriate and implementing a software for that hardware. The exact opposite of this is bottom-up design where the software defines what hardware to use. In earlier embedded systems design the system-control was designed before the data paths were defined. But all these methods lead to unreliable code and long development times.
Codesign[15] takes a different approach, where the system is seen as a synthesis of hardware and software, not as two parts who are somehow put together. Therefore the development of hard- and soft ware has to run in parallel. While looking at what needs to be done one decides in whether a function is simple enough to to it in hardware which is always faster than software or by implementing it with software were more sophisticated mechanisms are available. So the interfaces between hard and software can be defined and development runs without fewer iterations and more reliable code.[6]

2.4.4 Real-time Systems

Definition:

> *"A type of system in which the correctness of the system not only depends on the logical results of a computation but also on the time in which the results are produced."*

Usually, in a RT[16] system at least a part of the tasks has a certain urgency to them. The system has to react to events and/or generate such in a predefined time. Therefore the designer must have control of the scheduling within the system, to make sure it the tasks are deterministic.
There are two different kinds of RT tasks:

- **Hard** RT where the task must meet its deadline. Otherwise it will cause unacceptable damage or a fatal error to the system or its operation environment.
- **Soft** RT where meeting the deadline is desirable but not mandatory. Even if the deadline is not met the system can keep operating[7]

This analysis shows that the GM-Tube emulator as well as the real Gm-Tube is a *soft* RT applications although malfunction poses an indirect threat to the user.

[15] cooperative design of hard- and software
[16] Real-Time

2.5 Algorithms

In this project certain algorithms are needed to create the required events. These algorithms have to provide random numbers which are reproducible.

2.5.1 Pseudo Random Number Generators

A uniform pseudo random number generator(PRNG) provides the user with a series of numbers starting with a given seed and running through its whole spectrum of numbers by appearing to output those numbers in a random fashion. The numbers are evanly distributed and the series repeats after the generator went through its width, usually given in bits. The algorithms include the most common linear congruential generator(LCG) and linear feedback shift registers(LFSR) as well as the complexer algorithms like Mersenne twister.
The LCG is often implemented in the so called rand() function as a package within the programming language C. The operation is very simple and described by

$$X_{n+1} = (aX_n + c)\ mod\ m$$

, where X_0 is the seed, a the multiplier, c the increment and m the modulus. The LFSR on the other hand is explained within its name. A seed value is shifted several times in different directions with defined offsets. The Mersenne twister algorithm invented by Makoto Matsumoto and Takuji Nishimura basically twists the values from a LFSR in a complex process.
The LCG is the weakest algorithm of the three mentioned because it fails at several tests for randomness, whereas the LFSR with well chosen shift steps is the fastest and the *most random* of the simple generators.[8]

2.5.2 Exponential Distribution

Unlike the output of the PRNGs radioactive events are exponentially distributed. The radiation quanta follows the Poisson distribution

$$P(X = k) = \frac{\lambda^k}{k!}e^{-\lambda} \qquad (k = 0, 1, 2, ...; \lambda > 0)$$

and the occurrence of the events the regular exponential distribution function

$$F(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 - e^{-\lambda x} & \text{if } x \geq 0 \end{cases}$$

The fastest known method to generate exponentially distributed random numbers from a uniform distribution is the Ziggurat algorithm introduced by George Marsaglia.[10]

(a) The Poisson distribution

(b) The $e^{-\lambda x}$ distribution

Figure 2.7: International warning signs

2.5.3 The Ziggurat Algorithm

blabla

2.6 Basic conditions

At Turku University of Applied Sciences one of the specialization topics for students is *Embedded Software*. The department has its own laboratory and workshop. The mayor system for advanced embedded development is the XILINX®Virtex-II Pro Development Board. It consists of a XC2VP30 FPGA[17], 256MB DDR SDRAM and various I/O ports.

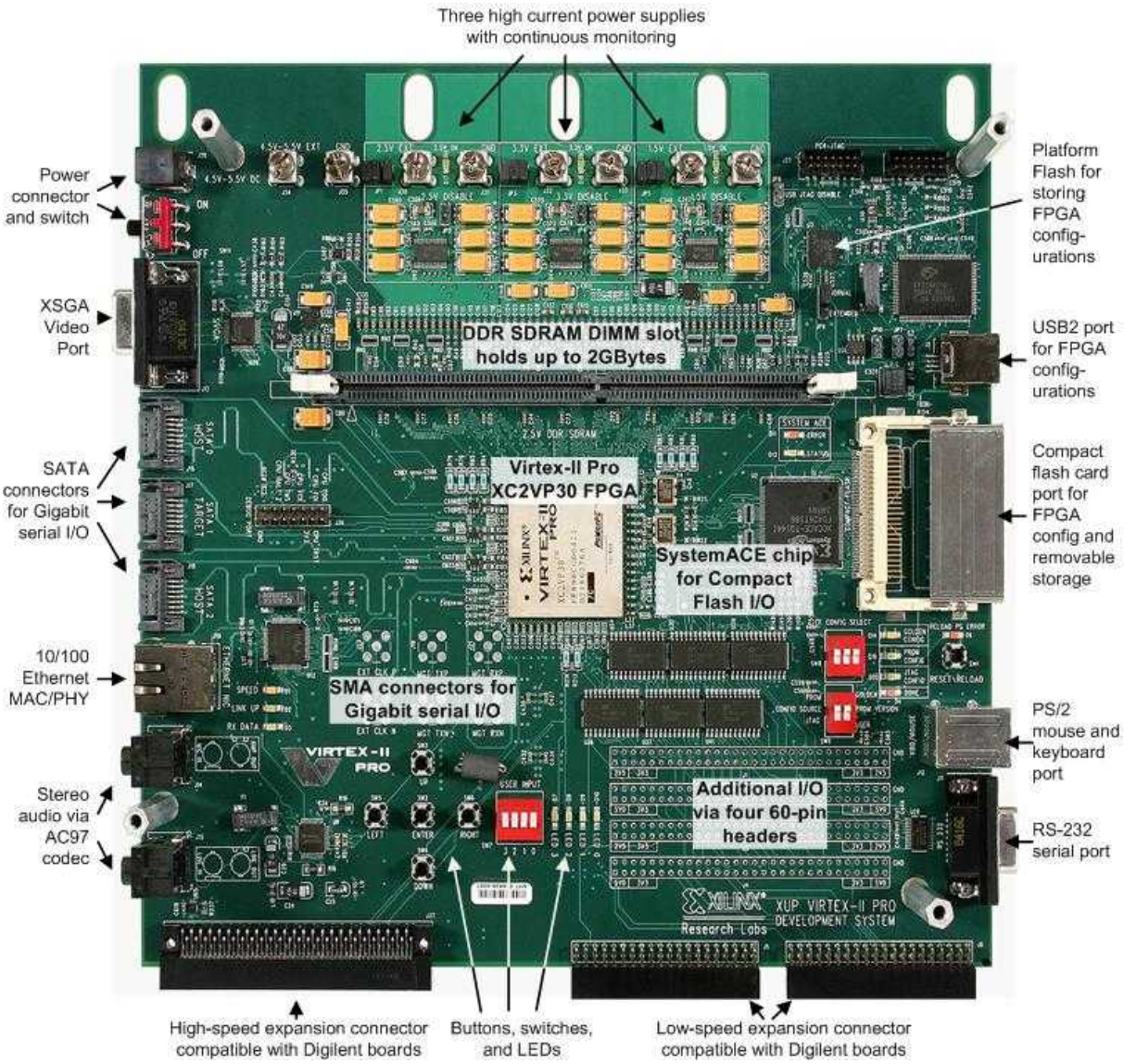

Figure 2.8: The XUP Virtex-II Pro Development board

This board offers the opportunity to gain a wide range of knowledge and experience. Amongst others it is used in an Embedded Linux course as well as in thesis projects.
The computer for developing the hard- and software for this thesis is a Fujitsu-Siemens E8010D with a 1.8GHz processor and 2GB of RAM.
For measurement the lab offers a wide range of oscilloscopes.

[17] fast programmable gate array

3 Design

> "Design is not just what it looks like and feels like. Design is how it works."
> *Steve Jobs*

This chapter describes components needed and *how* they are working together. Detailed implementation is described in chapter 4.

3.1 Architecture

The system contains the random number generator(RNG), a buffer for the generated events, the base and emulation hardware and a user interface. The interaction between the components is shown in Figure 3.1. Due to the relative simplicity of the system it is

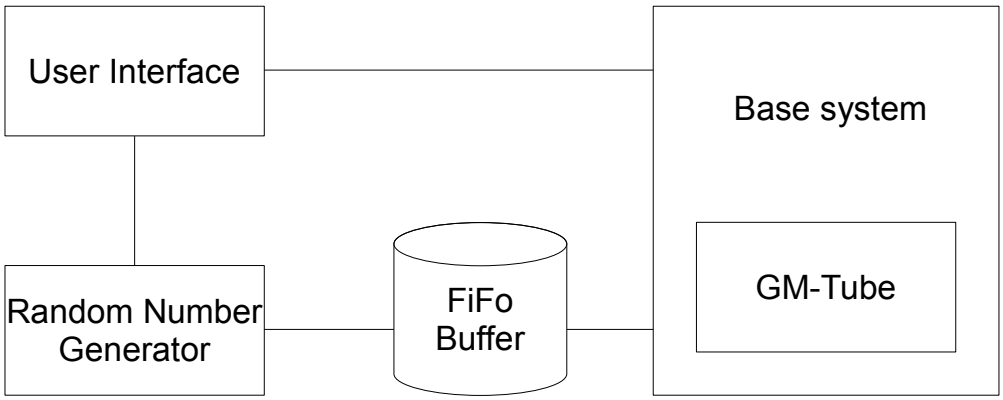

Figure 3.1: The GM-Tube Emulation Architecture

now possible to divide it into two main parts, software and hardware.
As to the fact that the specification demands the GUI to use standard I/O functionality the keyboard is the input device of choice. Its character based inputs as well as the output to the screen will be handled by software. The RNG itself would be faster to design in hardware but the complexity added by the need for an exponential distribution leads to a software module.
As one requirement the base system has to be hardware. Due to the need for speed the GM-Tube will also be implemented as a hardware module.
The FiFo[1] buffer needs special attention as to it is the connector between the RNG and the GM-Tube.
The following sections will take a closer look to the design of the five modules and the actual data and control flow between them.

[1] First-in-First-out

3.2 Hardware design

As shown in figure 2.8 the used hardware offers a huge amount of, but only a few modules are actually needed for the design of the emulator.

- **FPGA**
  the main processing unit
- **SDRAM**
  the random access memory for holding variables and tables needed for generating random numbers in an exponential distribution
- **Compact Flash Slot**
  for the memory card that holds the FPGA configuration the system boots from
- **RS232 Port**
  used as a UART[2] interface to interact with the user interface
- **LEDs**
  to visualize the output pulses and make them measurable at the same time

These Modules are interconnected within the base system.

3.2.1 Base system

The base system forms the hardware platform for the GM-Tube. It holds the architecture for all standard components provided by the FPGA board.
All the single components are interconnected through a system bus driven by a clock signal.

3.2.2 The GM-Tube Emulator

The design focus lies on the GM-Tube Emulator. Therefor figure 3.2 shows it's data in- and outputs. For reasons of better understanding the base system, in which the emulator is implemented, is not displayed.

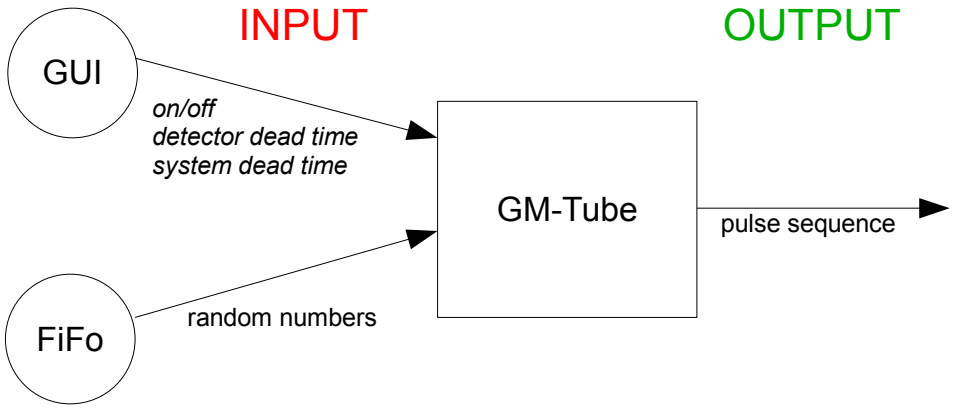

Figure 3.2: Gm-Tube I/O

[2]universal asynchronous receiver/transmitter

3.2.3 The FiFo

Due to the required performance the FiFo is designed as a part of the GM-Tube, which makes it hardware. From software side the RNG will write its output to a hardware address by using the principle of memory mapped I/O. The GM-Tube module will therefor provide a library with a simple in-line command.

3.3 Data Flow

The data flow of the system is shown without mentioning the base system because it just adds another layer to the GM-Tube that does not provide any mandatory information.

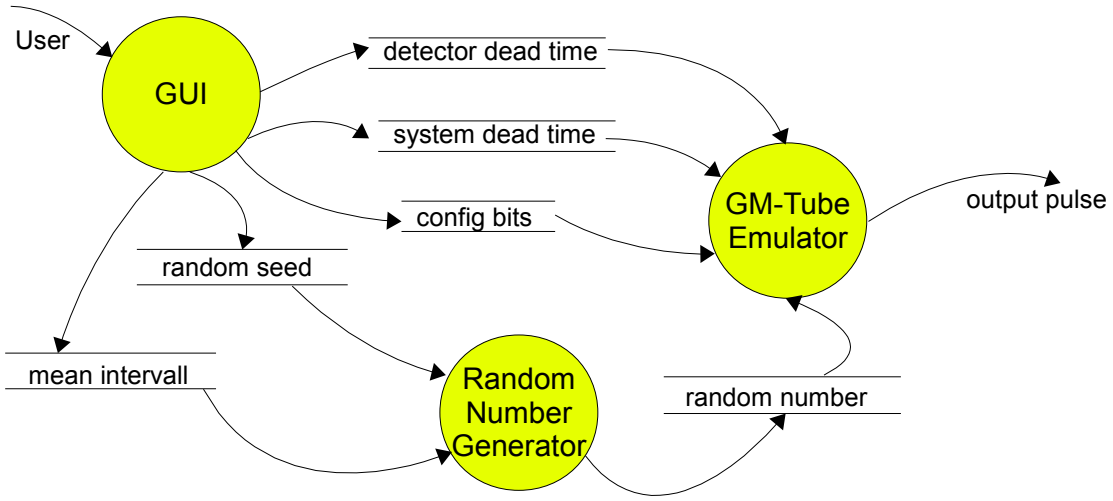

Figure 3.3: DFD of the top level system

3.3.1 Internal Data Flow of the GM-Tube

When designing the tube behavior it seemed convenient to enhance the functionality in a way that not only the regular system output pulse is shown but also the event pulse as well as the detector pulse. So it is possible to directly see and measure the difference between the number of generated radioactive events, the detector behavior and the actual Geiger count. This decision leads to the DFD shown in figure 3.4.

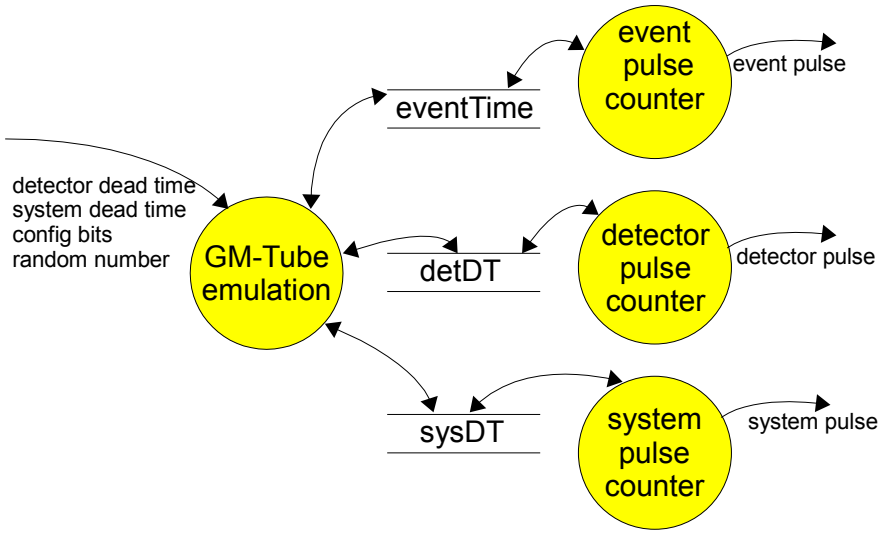

Figure 3.4: DFD of the Emulator

3.3.2 Internal Data Flow of the RNG

The RNG function is split up in three main parts.

- Setting up the tables needed for the Ziggurat algorithm
- Generation of evenly distributed random numbers
- Conversion of of the evenly distributed numbers to exponential distribution[3] and fitting them to the mean intervall

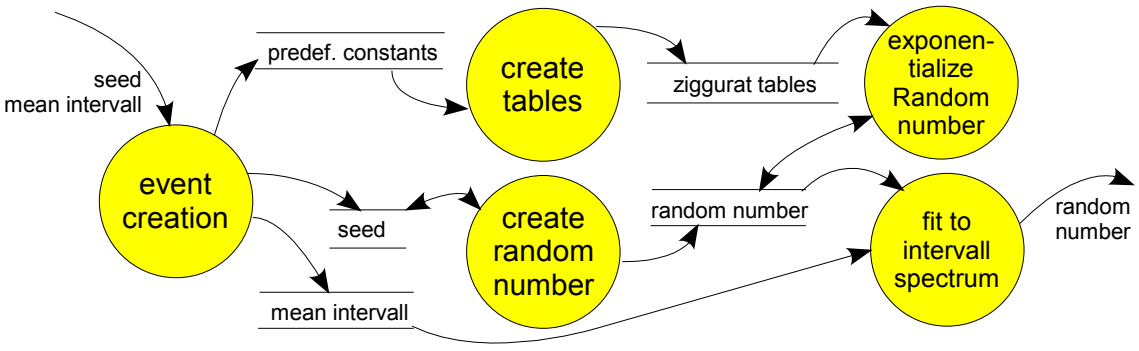

Figure 3.5: DFD of the Random Number Generator

[3]referred to as exponentialize

3.4 Control Flow

For a simpler illustration the control flow only contains the abstraction layer depth *needed* from the respective modules. Therefore the sequential and parallel executions are shown in figure 3.6 NOT READY YET

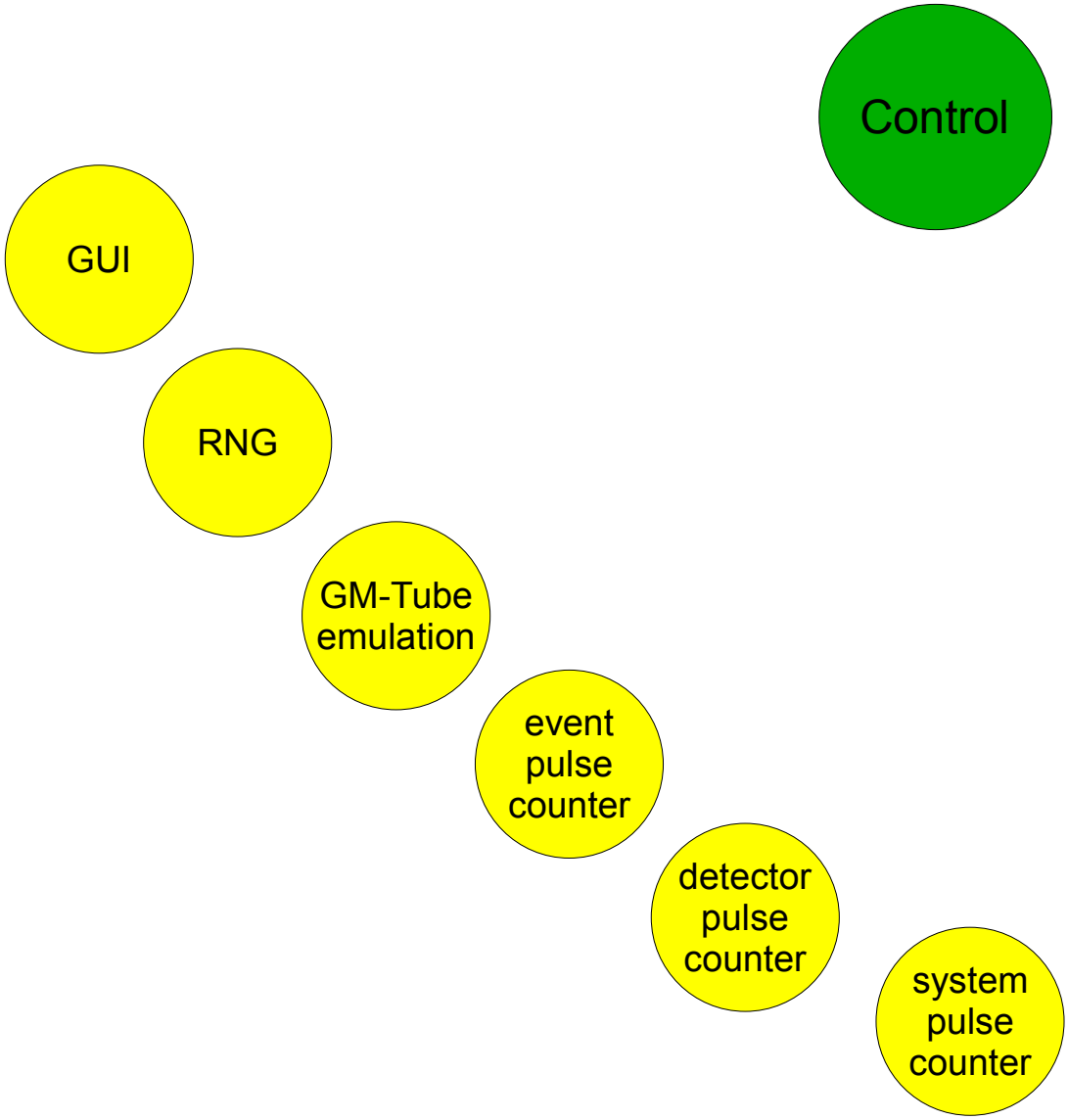

Figure 3.6: Control Flow Diagram of the system

3.4.1 Concurrency

3.5 Software design

The design of the software components is done

3.5.1 Base System Kernel

The main system, on which the GUI and the RNG are running will be a preconfigured kernel provided by XILINX. It includes general I/O features like UART and the LED control, memory management and interrupt control routines. Part of the core are also a possibility to use floating point arithmetic and mathematical functions needed for the RNG.

3.5.2 User Interface

The GUI design is rudimentary. When the system starts the user has to enter the configuration values step by step. If the user does not want to configure the system manually default values are taken for dead times, the seed and the mean interval. After starting the system, it continues operation unless it is interrupted by any character sent.

3.5.3 Random Number Generation

Due to the existence of various implementations of random number generators and mechanisms no new design is needed. The exponential derivate of numbers will be created after George Marsaglias paper "The Ziggurat Method for Generating Random Variables". A detailed description of the algorithm can be found in Appendix A.

4 Implementation and Testing

> "If your experiment needs statistics,
> you ought to have done a better experiment." *Ernest Rutherford*

For implementation the approach of *continuous integration*[1] was chosen. That allowed tests needed in the early phase to see if the performance requirements are met by the chosen algorithms. Without proceeding in short iterations the risk of failure for the project would have been inevitable due to the complex cooperation of hard- and software. The following sections describe what development environment was used and how the codesign ideas were applied for a harmonic interaction between hardware and software.

4.1 Development Environment

The design of the system was created with Prosa[2].
For developing the hardware layout of the base system and integrating the GM-Tube emulator the XILINX Platform Studio and the EDK[3] were used. The GM-Tube Emulator was built and tested separately with the ISE[4].
The programming development was done with Eclipse[5]. Due to the fact that only one developer was involved and the iterations were taken in small steps no version control or bug tracking software was needed.

4.2 Programming Languages

As a result of the complexity and at the same time hardware closeness the chosen programming language for the RNG was native C. The hardware description language used was VHDL[6], because it is the XILINX standard, which does not offer implementation of Verilog[7] modules.

4.3 Building the Hardware

The hardware implementation is divided into the base system and the GM-Tube emulator. Former is a straight forward process mostly default by the EDK and therefore the implementation detail is kept on a reasonable level. On the other hand the GM-Tube as the main focus of this documentation will be described down to the lowest layer.

[1] Integrate and build the system many times a day, every time a task is completed.[12]
[2] http://www.prosa.fi/
[3] http://www.xilinx.com/ise/embedded_design_prod/platform_studio.htm
[4] http://www.xilinx.com/ise/logic_design_prod/foundation.htm
[5] http://www.eclipse.org
[6] (VHSIC (**V**ery High Speed Integrated Circuits) **H**ardware **D**escription **L**anguage)
[7] http://en.wikipedia.org/wiki/Verilog

4.3.1 The Base System

The base system architecture was created with the *Base System Builder*(BSB) of the EDK tool chain. Although it is possible to design the whole system completely from scratch the provided wizard was used because otherwise this part would have consumed too much project time.
The generated system is shown in the following block diagram.

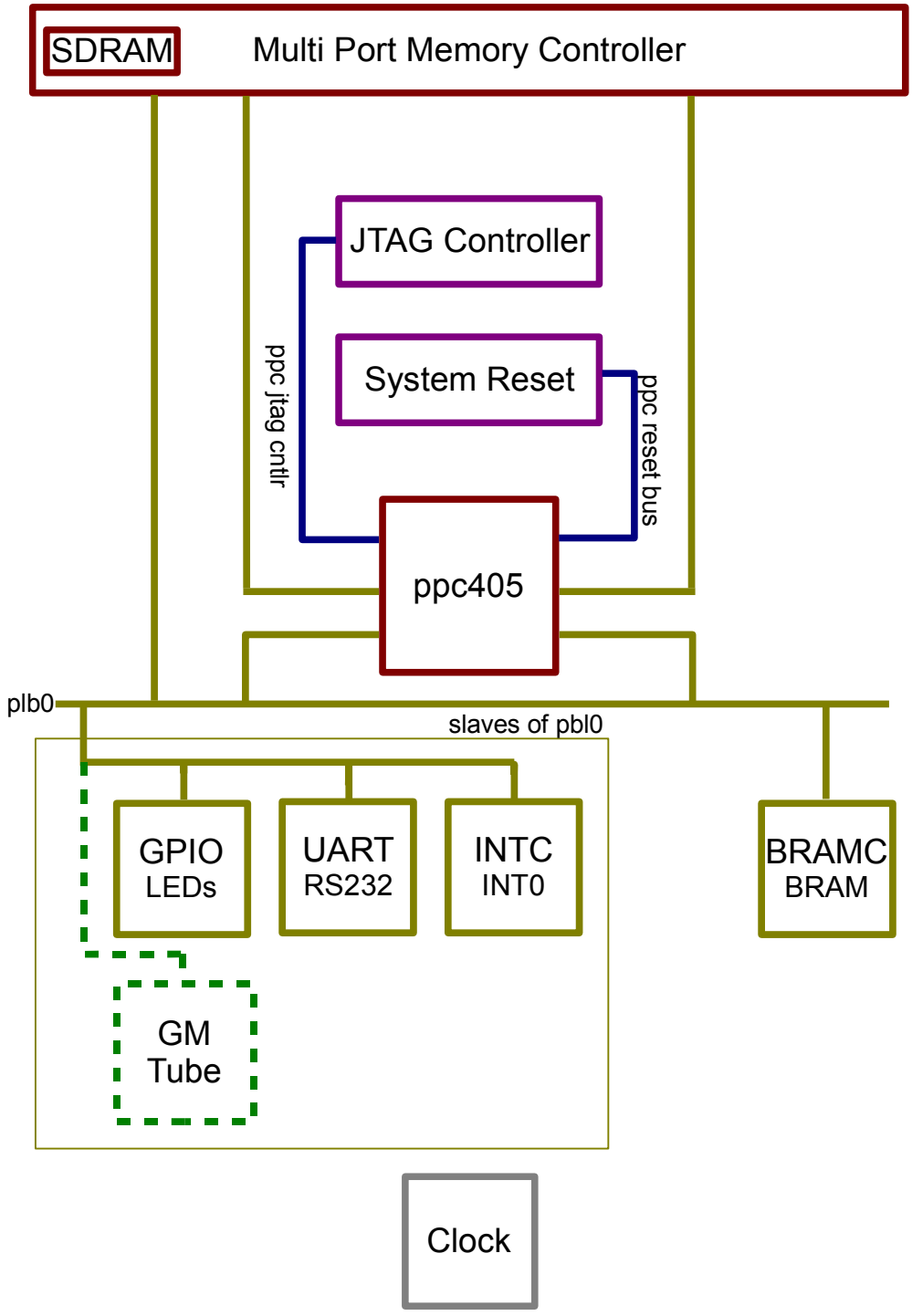

Figure 4.1: Block Diagram of the Base System

First the choice of the processor architecture was made. The FPGA offers two PowerPC cores and up to eight MicroBlaze CPUs, but the project only uses one PowerPC(ppc405) core. The processor clock runs with 300MHz, the maximum for the available hardware. It was also necessary to enable the instruction and data cache units of the ppc405 because performance measurement with a disabled cache showed that the emulator would not be realizable without. Finally the on-board JTAG controller was added as the debug interface for the CPU.
In the next step, the I/O interfaces were configured.

- `RS232_UART`: The serial interface uses UART standard 16550 for the highest available speed. The design includes the use of an interrupt so that the UART could halt the running emulator.
- `4bit LEDs`: The LEDs use the Xilinx Platform Studio(XPS) GPIO[8].
- `INTC`: An XPS interrupt controller is included to handle the interrupts produced by the UART controller.

The system needed at least one memory controller to hold data and instructions. This requirement is handled by an XPS BRAM[9]. Due to the fact that the Ziggurat algorithm needs to store tables of 256 32bit values and works with memory intense logarithm functions the 64k of maximum BRAM were not enough. To be on the safe side the full 256MB SDRAM external memory are included. Both memory controllers are connected with to processor cache for optimal performance.
All devices but the JTAG interface and the system reset controller communicate via the processor local bus(plb0) which is also driven by a 100MHz bus clock.

4.3.2 The GM-Tube Emulator

As shown in figure 4.1 the GM-Tube emulator has to be connected as a slave to the plb0 like all other I/O devices. Therefore a standard peripheral was built with the EDK. The XPS generated a framework for the custom peripheral 'GM-Tube' which allows it to use the clock and reset signals as well as the data connection to the system bus.
The chosen design gave the requirements for the interfaces to be added.

1. The Write FiFo
   - called *write* because from user's view values are *written* to it, although the peripheral reads from it
   - with a size of 1024 × 32bit to buffer the values from the comparably slow RNG
   - including additional signals that hold the information about the buffer status (empty, full)
2. Four 32bit Registers
   - the *enable* register for the possibility to start and stop the GM-Tube emulation
   - respectively one register for holding the detector & one for the system dead time
   - one register for holding the widening time for the output pulse

[8] general purpose I/O
[9] Block RAM

After generating the hull of the GM-Tube the ISE was used to customize the IP[10] and add the actual functionality. This is explained in next three subsections which finally lead to the implementation shown in figure 4.2.

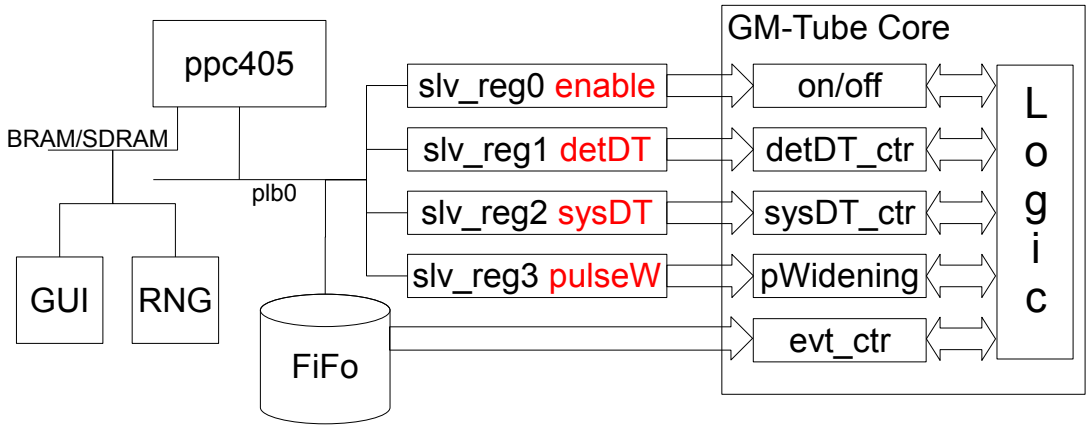

Figure 4.2: The GM-Tube IP in the System

The GM-Tube Framework

The GM-Tube framework is mostly generated by the EDK, but for forwarding the signals from the GM-Tube core additional ports were assigned and connected. Every signal(event, detector & system) will be connected to different LED port. This is why these signals need to have own ports in the framework too. The extra port declarations are

```
-- ADD USER PORTS BELOW THIS LINE ------------------
event_pulse : out std_logic;
detector_pulse: out std_logic;
system_pulse : out std_logic;
-- ADD USER PORTS ABOVE THIS LINE ------------------
```

and these are assigned within the port mapping for the *User Logic*.

```
-- MAP USER PORTS BELOW THIS LINE ------------------
event_pulse => event_pulse,
detector_pulse => detector_pulse,
system_pulse => system_pulse,
-- MAP USER PORTS ABOVE THIS LINE ------------------
```

Altogether those build the interfaces that are connected to the LEDs (see paragraph 4.3.3). For a better overview the ports in all three logics(framework, user & core) have the same names.
Although the ports are usually interconnected through signals, they were left away for reasons of simplicity.

The User Logic

The user logic is the connector for the GM-Tube core instance, the FiFo and the registers. Within this module two sections have to be altered.

[10]integrated peripheral

1. User Logic Ports and Port Map
   The same three ports as mentioned in paragraph 4.3.2 are created and connected to their equivalents.

2. "Architecture IMP[11] of the user_logic"
   All the ports of the GM-Tube core are declared and assigned in the port map.

   - The clk and reset ports are mapped to the system's equivalents.
   - Slave registers zero to three are connected to enable, detDT & sysDT as well as pulseW ports.
   - The data_load, data_required and the data_acknowledge are mapped to the FiFo's ports and signals.

For detailed implementation(VHDL code) of the user logic please refer to appendix B.

The GM-Tube Core

The core module holds the actual GM-Tube emulation. As shown in the design(3.4) the GM-Tube contains different simultaneous operations. Depending on the current state The system can be divided into two main instruction sets, event/ detector/system pulse creation and down counting. The diagram shown in figure 4.3 represents the process flow of one single clock cycle.
The GM-Tube core is implemented as a combination of down counters and comparators. After the system is enabled the first value is read from the FiFo to initialize the event counter which represents the generated radiation event interval. Every clock cycle, the value is decremented by one. This decrement represents the duration of 10ns and marks the system pulse resolution. If the detector dead time is zero when a new event occurs the detector and the system pulse are high. At the same time the dead time counters for both are reset and a new value from the FiFo is requested. In the next clock cycle, the acknowledge bit will be set by the FiFo and the new value will be loaded into the event counter.
In most cycles the event counter is unequal to zero and so the event and detector pulse are down. If the system dead time is up the system pulse will also be switched of. Every regular counting cycle the current widening for the system pulse is calculated.
When revisiting the requirements it is noticeable that the the events may occur with a mean time of 1μs which compared to the system clock is very short. Is that the case the danger of buffer under run is very high so it is necessary that the user keeps track of the FiFo fill level. If the FiFo runs empty unexpected things might happen.

[11] Implementation

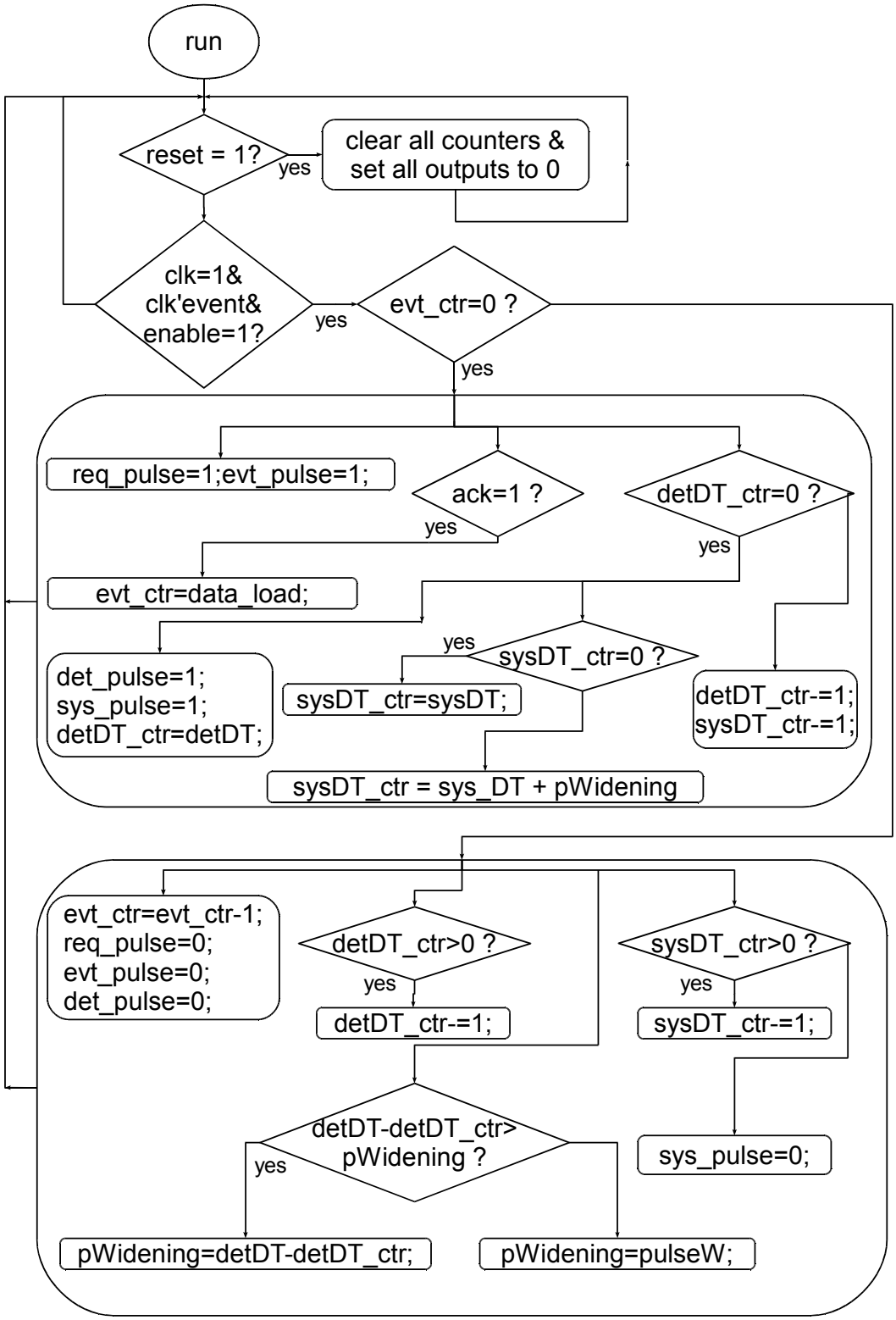

Figure 4.3: Instruction Flow of the GM-Tube Emulator

4.3.3 Synthesizing the Hardware

After the custom IP was implemented and the stand-alone synthesis had worked it was added as a new IP(figure 4.4) to the base system.

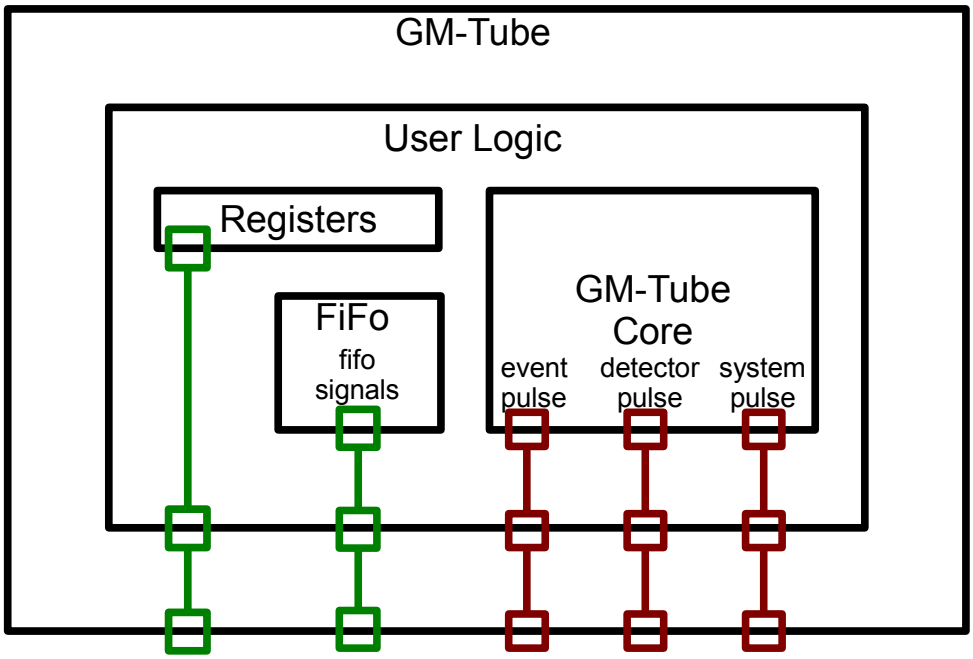

Figure 4.4: The GM-Tube IP

Therefore the configuration of the base system had to be updated/altered at four points.

1. Assembly
   - connection to the plb0 system bus
2. Port Configuration
   - creation of one external port for each pulse
3. Address Configuration
   - assigning the address space with the auto assign function
4. GPIO Pin Mapping(system.ucf)
   - manually assigning the new external ports to the LEDs

Connection for the event pulse

```
Net geiger_counter_event_pulse_pin LOC = AC4;
Net geiger_counter_event_pulse_pin IOSTANDARD = LVTTL;
Net geiger_counter_event_pulse_pin SLEW = SLOW;
Net geiger_counter_event_pulse_pin DRIVE = 12;
```

After this step the hardware was tested.

4.4 Testing the hardware

This section describes the hardware test for the interplay between the FiFo, the registers and the GM-Tube core module within the base system.

4.4.1 Test Arrangement

For proving that the internal mechanism responds correctly to a given set of values a simple test case was created. A minimal test program, uploaded via JTAG, filled the FiFo with a set of

a) equal numbers [100000,100000,100000,...],

b) decreasing numbers [100000,999000,998000,...,0],

c) a combination of a & b,

configured the dead times and pulse widening and enabled the GM-Tube. After checking manually(via UART to console), that the registers are written correctly and the FiFo fill level and elements are as supposed the collection of data was taken with the following hardware arrangement.

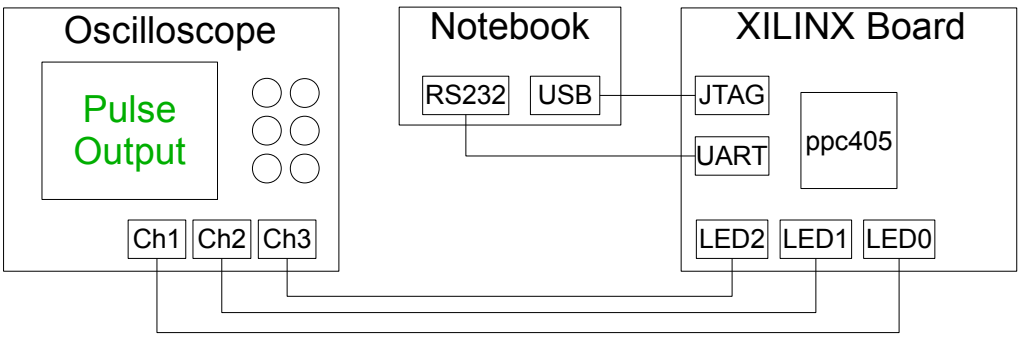

Figure 4.5: Hardware Test Arrangement

4.4.2 Test Results

After more that fifty test series with varying dead time and pulse widening configurations the hardware was declared as "fully functional and without flaws". The pulses were generated like supposed and the asynchronous communication with the FiFo(Timing details in paragraph 4.7) did not lead to any lost values/events.

4.5 Building the software

Compared to the hardware design and implementation process the software could be implemented in a fairly simple manner. The following subsections describe which software packages and libraries were used and how the application was realized.

4.5.1 The Kernel

The XPS basically offers four different kernel possibilities.

- *standalone*: a minimal kernel offering basic functionality without including threads
- *xilkernel*: also a XILINX base kernel, but including more functionality like software timers or threads
- *VxWorks*: a real-time operating system made and sold by Wind River Systems[12]
- *selfmade*: the possibility to create an own kernel from scratch

Due to the fact that the university had no license for the VxWorks libraries and there was not enough time to build an own kernel the two remaining suitable possibilities were *standalone* and *xilkernel*. When the software implementation started this choice led to the xilkernel because the decision about multi threading could not be made yet.

4.5.2 The GUI

The GUI consists of printf and scanf commands offered by the standard C library *stdio.h*. The values are written into the registers with the help of the automatically by the GM-Tube synthesis generated macros[13].

4.5.3 The RNG

The algorithm for the random number generator is copied from the Ziggurat algorithm paper mentioned in the analysis. Additional the values are converted to *unsigned long* and depending on the mean interval multiplied by a power of ten.
The standard C math library provided the required functions.

4.5.4 Measurement

During the implementation of the software it was necessary to take a closer look at the RNG performance. The formulated mean time requirement made it mandatory that the RNG is able to produce at least one random number within 1μs.
Therefore a simple test routine was written(pseudo code).

```
main_routine{
   endless_loop{
      enable LED;
      loop(1 million){generate random number;}
      disable LED;
   }
}
```

[12] http://www.windriver.com/
[13] in-line commands

The time of the LED pulse was measured similar to figure 4.5. The time it takes to run through the loop once are approximately 30ns and the time to enable/disable an LED is about 300ns. Resulting from multiple test runs with different seeds the time to generate one number lies slightly under 1μs which showed that the requirements were met. The compiler optimization level was set to three, except for the loop time measurement, a fact that did not matter because the assembler code review of that section showed no difference for either optimization level.
Different test runs with one thread for the GUI and one for the RNG showed that due to unpredictable timings the GUI had to be implemented with the use of interrupts. Although the used *xilkernel* produces more overhead than the *standalone* version, no performance difference could be measured and the kernel choice made in subsection 4.5.1 was not changed.

4.6 Testing the software

The test arrangement for the software test is similar to figure 4.5. Only this time the kernel plus the GUI and the RNG are on a compact flash(CF) card and the jtag connection is not used.
The software was tested with ten different seed and mean interval values and the generated random numbers fulfilled the requirements. The GUI also worked smoothly and did not disturb the timing of the RNG.
Anyhow two things are to be mentioned:

- The seed value must be different from zero because of the SHR; a shifted zero will always be zero.
- The gap between the measured timings of the RNG and the Ziggurat paper in appendix B are a result of the slow floating point emulation on the FPGA.

4.7 Synthesizing and Testing Hard- and Software

The final software was created by compiling the xilkernel, RNG, GUI and the libraries for the GM-Tube IP. The resulting application was put onto the CF card and the system started.

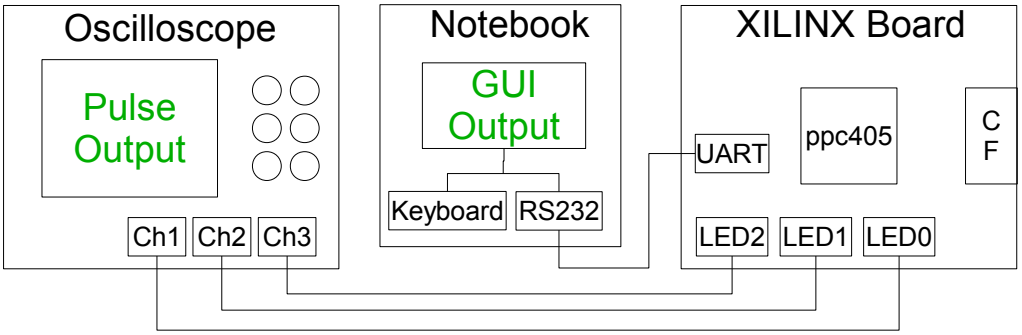

Figure 4.6: System Test Arrangement

After running multiple combinations of the previous tests, the three main questions from the analysis were revisited.

- Speed: Is the radiation generator able to create the events on demand?
  ☑The RNG is able to create at least on event per μs, which is the smallest mean interval time
- Scale: Does the scale of generator output fulfill the required interval spectrum?
  ☑The hardware allows an interval spectrum between 10ns and 10s. The outcome of this is that this spectrum covers the required mean interval times but due to the limited width of 32bit no events larger than 10s are possible.
- Performance: Does the system show the correct number of output pulses?
  ☑The number of event/detector/system pulses is correct.

5 Conclusion

> "However beautiful the strategy, you should occasionally look at the results."
> *Winston Churchill*

To begin with this chapter the requirements are revisited and the test results interpreted, followed by two sections covering some of the problems and solutions during the project and finally future possibilities are shown.

5.1 Results

All results refer to requirements which are met completely.

5.1.1 Accuracy

The accuracy of the system is mostly dependent on the hardware. The largest possible integer value of the system defines the biggest number the RNG can output.
The self-created GM-Tube IP has a maximum error of 20ns caused by the asynchronous FiFo request, which according to the maximum resolution of 10ns is absolutely tolerable.

5.1.2 Performance

With the used hardware it is not possible to create a faster or more reliable system. The soft real-time requirements are fully met.

5.1.3 Usability

The system is completely controllable via the user interface. The fact that with RS232, keyboard and console only standard I/O devices were used mets the requirements. Because the possible event scale and the extra pulse outputs the system offers excellent services for experimenting with different configurations for the GM-Tube behavior.

5.1.4 Stability

The system is stable as long as the FiFo does not run empty, an event that did not occur during the tests and is very unlikely to happen.

5.1.5 Portability

The portability of the soft and hardware is optimal due to the use of standard C for the software. The GM-Tube hardware on the other hand is written in VHDL and uses no specific processor options, a fact that makes even the FPGA exchangeable.

5.2 Occurred Problems and Solutions

During the project especially the hardware creating held many challenges.
The XILINX software surprises with more than a few features the programmer has to deal with and which are not part of the main functionality. As one of many it forces the user to use a certain mechanism for the VHDL process: The reset condition in the main process must not be written with an *elsif* statement. This feature leads to a synthesizing error which sometimes appears when building the hardware within the ISE and sometimes when embedding the hardware in the EDK.
For finding the right solutions the XILINX forum[1] the best contact point.

5.3 Future Possibilities

The use of the Ziggurat algorithm offers the possibility, not only to use the radiation generator for events but because of the decreasing density of the numbers also as a generator for the energy of those events. With this addition and another kind of radiation detector on the hardware side emulation of a spectral analyzer could be realized.
The GM-Tube IP itself could be expanded by a configurable counting mechanism which allows the user to see the counts per time interval. Also a dosimeter operation mode is thinkable.

5.4 Resume

All requirements for the emulator were met. The hardware functionality is expendable and the project builds a good basis for further development. As one of the first hardware software co design projects on the XILINX board it holds good potential for the Turku University of Applied Sciences.
Personally the project was very challenging due to the fact that before I never gotten closer to hardware than with Assembler. It was a very good experience and I am looking forward to deepen my experience in embedded design.

[1] http://forums.xilinx.com/xlnx/

Bibliography

[1] Encyclopædia Britannica Online, *Radioactivity*. April 2009.

[2] Werner Stolz, *Radioaktivität Grundlagen - Messungen - Anwendungen*. Vieweg+Teubner, 5th Edition, 2005.

[3] Glenn Knoll, *Radiation Detection and Measurement*. Wiley & Sons, 2nd Edition, 2007

[4] Irving Kaplan, *Nuclear Physics*. Addison-Wesley, 2nd Edition 1962

[5] John Catsoulis, *Designing Embedded Hardware*, O'Reilly, 2nd Edition, 2005

[6] Bassam Tabbara, *Breathing life into hardware and software codesign*. Article from the magazine *Embedded Systems Programming* Vol.18 No.4, April 2005

[7] William Stallings, *Operating Systems*, Prentice Hall, 5th Edition, 2005

[8] http://en.wikipedia.org/, *Pseudorandom number generator*, 23rd April 2009

[9] Bronstein, Semendjajew, Musiol, Mühlig, *Taschenbuch der Mathematik*, Verlag Harri Deutsch, 5th Edition, 2001

[10] George Marsaglia and Wai Wan Tsang,
The Ziggurat Method for Generating Random Variables, Florida State University and University of Hong Kong, http://www.jstatsoft.org/v05/i08/paper, 23rd April 2009

[11] Edward Yourdon, *Just Enough Structured Analysis*, Prentice Hall, 2nd Edition, 2006

[12] Kent Beck, *Extreme Programming explained*, Addison-Wesley, 1st Edition, 2000

Statement of Independence

Herewith I declare

Appendix A

here the ziggurat code

Appendix B

here the gm-tube core vhdl code