SoftwareX 7 (2018) 122–128

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

DLTPulseGenerator: A library for the simulation of lifetime spectra based on detector-output pulses

Danny Petschke *, Torsten E.M. Staab

University Wuerzburg, Department of Chemistry, LCTM Roentgenring 11, D-97070 Wuerzburg, Germany

ARTICLE INFO

Article history:
Received 20 October 2017
Received in revised form 9 April 2018
Accepted 10 April 2018

Keywords:
Lifetime spectroscopy
Signal processing
Pulse simulation

ABSTRACT

The quantitative analysis of lifetime spectra relevant in both life and materials sciences presents one of the *ill-posed* inverse problems and, hence, leads to most stringent requirements on the hardware specifications and the analysis algorithms. Here we present *DLTPulseGenerator*, a library written in native *C++ 11*, which provides a simulation of lifetime spectra according to the measurement setup. The simulation is based on pairs of non-TTL detector output-pulses. Those pulses require the Constant Fraction Principle (CFD) for the determination of the exact timing signal and, thus, the calculation of the time difference i.e. the lifetime. To verify the functionality, simulation results were compared to experimentally obtained data using Positron Annihilation Lifetime Spectroscopy (PALS) on pure tin.

Code metadata

Current code version	v1
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-17-00077
Legal Code License	BSD-3-clause
Code versioning system used	GitHub
Software code languages, tools, and services used	C/C++ and Python
Compilation requirements, operating environments & dependencies	**OS**: Microsoft Windows **Compilation requirements (for DLTPulseGenerator.h/.cpp only):** should work with any C++ compiler (has to provide C++11 standard) – recommended: MS-VSCompiler (at least version 2013) **Dependencies for example C++ project - AppDLTPulseGenerator:** Microsoft Visual Studio 2015 **Dependencies for C++ wrapper in Python - pyDLTPulseGenerator.py:** ctypes-library **Dependencies for example project in Python - pyDLTPulseGeneratorApp.py:** matplotlib, NumPy
If available Link to developer documentation/manual	A ***Readme.md*** *file* can be found on **GitHub**: https://github.com/dpscience/DLTPulseGenerator/blob/master/README.md
Support email for questions	danny.petschke@uni-wuerzburg.de

1. Introduction and significance

Lifetime spectroscopy has become an established method in life science, physics and materials science over the last decades. A first step was taken in the early *1960's* by several groups measuring the lifetime distributions of antielectrons (positrons) in materials using photomultiplier–scintillator combinations to detect the gamma rays, which are emitted when they are annihilated with an electron [1–3]. This method, known as Positron Annihilation Lifetime Spectroscopy (PALS), is used for microstructure investigations in a broad range of material classes from metals [4–9] and semiconductors [10,11] to polymers [12,13] and porous glasses [14,15]. The characteristic or specific lifetimes are highly sensitive to the kind and size (from several angstroms to nanometers) of

* Corresponding author.
E-mail addresses: danny.petschke@uni-wuerzburg.de (D. Petschke), torsten.staab@uni-wuerzburg.de (T.E.M. Staab).

https://doi.org/10.1016/j.softx.2018.04.002

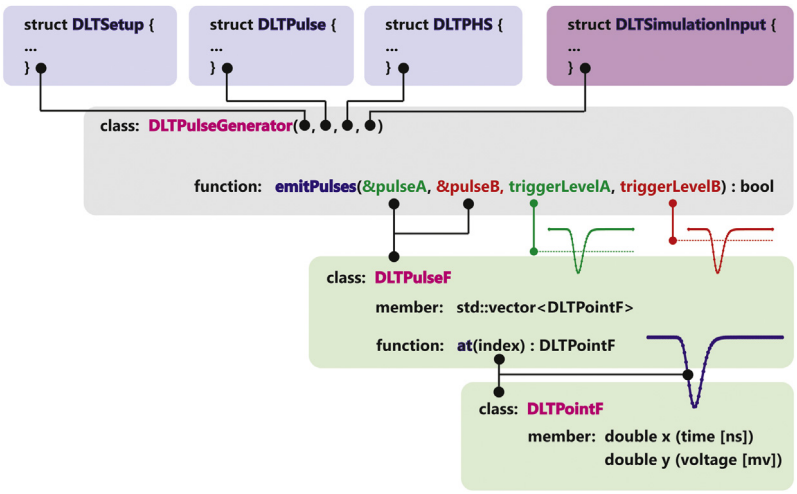

Fig. 1. Schematic illustration of the library architecture. *DLTPulseGenerator* expects four structures for initialization. *DLTPulseF* holds a vector of the data points (time vs. voltage) in double precision represented by the class *DLTPointF*.

material defects (e.g. impurities in semiconductors, vacancies in metals or pores in glasses) and range from several picoseconds to microseconds. After the technical realization of single photon sensitive detectors (photodiodes, APD — avalanche photodiodes), methods such as Fluorescence Lifetime Spectroscopy (FLS), Fluorescence Lifetime Imaging Microscopy (FLIM) or Fluorescence Lifetime Correlation Spectroscopy (FLCS) became feasible and are used nowadays to investigate (life cell) protein interactions [16] or diffusion dynamics [17].

Those lifetime spectra are mathematically described by a sum of exponential lifetime distributions f_i convoluted with an Instrument Response Function (IRF) g. According to the number of components N, the resulting function f is given as

$$f(t) = \sum_{i=0}^{N-1} f_i(t) = \sum_{i=0}^{N-1} \frac{I_i}{\tau_i} \exp\left\{-\frac{t}{\tau_i}\right\}, \tag{1}$$

where τ_i as the ith component *specific lifetime* and I_i its corresponding *intensity*. Mostly, the IRF is analytically approximated by a Gaussian distribution function[1]

$$g(t|\mu,\ \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-0.5\left(\frac{t-\mu}{\sigma}\right)^2\right\}, \tag{2}$$

where σ is the standard deviation and μ is the mean.

A quantitative analysis of the lifetime spectra and, thus, the extraction of the relevant information, i.e. the *specific lifetimes* and its *intensities*, needs to solve the *ill-posed* inverse problem, which means that the uniqueness or even the existence of the solution is not always assured [18]. Various approaches are used: the *Maximum-Likelihood Method*, the most commonly used *Least-Square Fitting* (software: e.g. Positronfit [19–21], PALSFit [22,23], LT [24,25], FluoFit [26] or SymPhoTime 64 [27]) or the *Bayesian* approach using the quantified *Maximum Entropy Method* (MEM or MaxEnt) (software: e.g. MELT [28,29]). Especially for spectra consisting of multiple lifetime components and/or having shorter *specific lifetimes* than the instrumental resolution (means: $\tau_i < \sigma$), an IRF with minimum deviations from the model function Eq. (2) is required to solve this *ill*-conditioned problem, i.e. this is decisive for an exact data treatment. Therefore, the setup and the parameters relevant to determine the correct timing signal from the received pulses, using the Constant Fraction Principle (CFD), needs to be fully optimized to guarantee reproducible and comparable results.

DLTPulseGenerator library generates pairs of detector pulses (non-TTL signals) with exponentially distributed (Eq. (1)) time differences, i.e. the lifetime, by taking the pulse shape and the uncertainties (Eq. (2)) of the most relevant hardware components into account (Fig. 2). This allows the user to study the influences of both the measurement setup and configuration on the lifetime spectrum without being connected to the hardware.

2. Software description

DLTPulseGenerator is written in native *C++ 11* (ISO/IEC 14882: 2011). It provides the optional compilation as static or linked library to make it easy accessible to other programming languages which are preferentially used in research and engineering, e.g. *Matlab* (using mex-library) or *Python* (using ctypes-library,[2]).

The ***class DLTPulseGenerator*** expects four structures (C/C++ syntax: *struct*) for initialization (Fig. 1):

i. **DLTSetup**
ii. **DLTPulse**
iii. **DLTPHS**
iv. **DLTSimulationInput**

These contain the specifications of the setup and information on pulse shape, pulse height distribution and the lifetime distributions. A pulse is represented by the ***class DLTPulseF*** and consists of a vector (*std::vector*) which holds the data points (***class DLTPointF***).

Calling the function *DLTPulseGenerator::emitPulses* expects:

I. a pointer to an object of the *class DLTPulseF*, which is then manipulated and filled with data points (time [ns] vs. voltage [mV]) in double precision and
II. the trigger-level.

In the following chapters, a detailed overview of the physical and mathematical background, which the simulation is based on, is given while relating to the mentioned structures (C/C++ syntax: *struct*).

[1] Different distribution functions such as Lorentz/Cauchy or Voigt as well as superpositions and skewing of distribution functions are not considered in this work but could be easily implemented.

[2] ***pyDLTPulseGenerator***: a class in *Python* showing the functionality and use of *ctypes-library* in combination with *DLTPulseGenerator* (compiled as linked library), is provided by the author.