КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра медичної радіофізики

«На правах рукопису»

Робота допущена до захисту в ЕК
рішенням кафедри медичної радіофізики
від __ травня 2026 року, протокол № ____.
Завідувач кафедри кандидат фіз.-мат. наук, доцент
_____ Сергій РАДЧЕНКО

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА
на тему:
«ПРОГРАМНА РЕЄСТРАЦІЯ ОПТИЧНИХ СИГНАЛІВ З ФОТОННОЮ РОЗДІЛЬНОЮ ЗДАТНІСТЮ ЗА ДОПОМОГОЮ ШВИДКІСНОГО АЦП»

Виконав:
студент 2-го курсу магістратури
денної форми здобуття освіти
спеціальності 105 Прикладна фізика та наноматеріали
ОНП Біомедична фізика, інженерія та інформатика
Рижевський Євген Русланович _____

Наукові керівники:
доктор фіз.-мат. наук, старший науковий співробітник
Дмитрук Андрій Миколайович _____
кандидат фіз.-мат. наук, доцент
Радченко Сергій Петрович _____

Рецензент:
доктор філософії
Костецький Антон Олегович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____ Євген РИЖЕВСЬКИЙ

Київ - 2026

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

АЦП — аналого-цифровий перетворювач

ПЗ — програмне забезпечення

ПК — персональний комп’ютер

API — інтерфейс прикладного програмування

ctypes — стандартна бібліотека Python для зв’язку з функціями зовнішніх бінарних бібліотек

CSV — текстовий формат табличних даних з роздільниками-комами

DLL — динамічно підключувана бібліотека операційної системи Windows

FWHM — повна ширина імпульсу на рівні половини амплітуди

GUI — графічний інтерфейс користувача

JSON — текстовий формат подання структурованих даних

MAD — медіана модулів відхилень від медіани, робастна оцінка розкиду

PCCT — фотонно-лічильна комп’ютерна томографія

p.e. — фотоелектрон, одинична подія народження носія заряду у напівпровідниковому детекторі

PMT — фотоелектронний помножувач

RMS — середньоквадратичне значення

SiPM — кремнієвий фотопомножувач (МППС — мікропіксельний підрахунковий пристрій)

USB — універсальна послідовна шина передачі даних

РЕФЕРАТ

Магістерська робота: 91 с., 10 рис., 22 джерела, 2 додатки.

Робота присвячена розробці програмного комплексу для реєстрації, збереження та аналізу оптичних сигналів з фотонною роздільною здатністю на базі високошвидкісного аналого-цифрового перетворювача CAEN DT5761 у зв'язці з кремнієвим фотопомножувачем як джерелом сигналу.

Розроблено модульний програмний комплекс мовою Python з графічним інтерфейсом на основі бібліотеки PyQt6. Архітектура побудована навколо абстракції пристрою-джерела даних, що дозволяє підставляти як реальну плату через прошарок ctypes до системних бібліотек CAEN, так і вбудований симулятор сигналу кремнієвого фотопомножувача без зміни решти коду.

Запропоновано бінарний формат збереження записів .phadc із самоописовим JSON-заголовком, що відкривається стандартними засобами питру одним рядком коду. Реалізовано три режими тригера — програмний, за рівнем сигналу, зовнішній.

Реалізовано алгоритм пошуку імпульсів на основі робастних оцінок базової лінії через медіану і шуму через MAD-статистику з адаптивним порогом за критерієм п'ять сигм. Для кожного знайденого імпульсу обчислюються амплітуда, повна ширина на половині максимуму з лінійною інтерполяцією між відліками, площа та час наростання за рівнями 10–90 відсотків.

Розроблений симулятор сигналу кремнієвого фотопомножувача відтворює пуассонівські моменти приходу подій, біекспоненційну форму імпульсів, квантовані рівні однофотоелектронного відгуку, темновий рахунок, гаусівський шум і мережеву наводку 50 Гц.

Програмний комплекс протестовано на еталонному синусоїдальному сигналі генератора, на власному симуляторі та на реальному обладнанні з підключеним кремнієвим фотопомножувачем. На реальній платі підтверджено коректну роботу режимів програмного тригера і тригера за рівнем сигналу, отримано квантовану структуру амплітудного спектра, характерну для одноелектронного відгуку, виміряно частоту темнового рахунку детектора.

Ключові слова: ФОТОННА РЕЄСТРАЦІЯ, КРЕМНІЄВИЙ ФОТОПОМНОЖУВАЧ, ШВИДКІСНИЙ АЦП, CAEN DT5761, ЦИФРОВА ОБРОБКА ІМПУЛЬСІВ, ОДНОФОТОННА РОЗДІЛЬНА ЗДАТНІСТЬ, ПРОГРАМНИЙ ІНТЕРФЕЙС.

ЗМІСТ

ВСТУП

Сучасні методи лабораторної реєстрації оптичних і рентгенівських сигналів дедалі активніше зміщуються від класичної схеми накопичення сумарної енергії потоку фотонів до підходу, побудованого на рахунку окремих фотонів. У такій схемі реєстрації сигнал від детектора розпадається на послідовність окремих імпульсів, кожен з яких відповідає одній взаємодії з квантом випромінювання, причому амплітуда імпульсу пропорційна енергії взаємодії. Це принципово відмінний від інтегруючого підходу режим, який дозволяє виокремити кожну подію, оцінити її енергію і виключити зі звітної статистики фоновий тепловий шум за енергетичним порогом [1].

Найпомітнішим клінічним проявом цього зміщення стало впровадження фотонно-лічильної комп'ютерної томографії (PCCT). На відміну від класичних КТ-сканерів, де сцинтиляційний блок з фотодіодом перетворює потік рентгенівського випромінювання на сумарний електричний сигнал, у PCCT-системах використовується пряме напівпровідникове детектування на телуриді кадмію, телуриді кадмію-цинку або кремнії. Кожен поглинутий фотон породжує хмару носіїв заряду, а зчитувальний інтегрований канал перетворює її на електричний імпульс із амплітудою, пропорційною енергії. Така архітектура надає вищу просторову роздільну здатність за рахунок дрібнішого пікселя без оптичних розділювачів, дозволяє виконувати спектральну реконструкцію зображення з енергетичних бінів і зменшує дозу опромінення пацієнта завдяки відсіченню теплових шумів. Перші серійні PCCT-системи — NAEOTOM Alpha від Siemens Healthineers, OmniTom Elite від Samsung Healthcare, портативний Extremity Scanner System від MARS Bioimaging, Photonova Spectra від GE HealthCare — уже пройшли клінічну сертифікацію і використовуються у клініках Європи та Північної Америки [1].

Принципи лічильної реєстрації не обмежені рентгенівським діапазоном і клінічними застосуваннями. У видимому, ультрафіолетовому та інфрачервоному світлі ту саму задачу розв'язують лавинні фотодіоди у режимі Гайгера,

фотопомножувачі різних типів, мікроканальні пластини. У всіх цих приладах кінцевий сигнал формується як послідовність коротких імпульсів, кожен з яких відповідає поглинанню одного або кількох фотонів. Найпоширенішим детектором цього класу у лабораторній практиці є кремнієвий фотопомножувач, який позначають як SiPM. Це матриця мікроскопічних лавинних фотодіодів, об'єднаних у спільне коло, у якій кожен поглинутий фотон спричиняє лавину носіїв заряду у відповідному мікропікселі і формує на виході короткий струмовий імпульс наносекундної тривалості [2]. Завдяки тому, що мікропікселі об'єднані у паралельне коло, амплітуда вихідного сигналу при одночасному поглинанні кількох фотонів є кратною амплітуді одиничного відгуку. Це дозволяє безпосередньо рахувати кількість фотонів у вікні реєстрації за висотою сформованого імпульсу. Однак отримання кількісно достовірних результатів з таких детекторів накладає серйозні вимоги на електронний тракт реєстрації. Тривалість одиничного імпульсу від SiPM становить одиниці-десятки наносекунд, що відповідає необхідній смузі пропускання аналогового тракту понад 200 МГц і частоті дискретизації аналого-цифрового перетворювача на рівні гігасемплів за секунду [3]. Імпульси приходять стохастично, з характеристиками потоку Пуассона, і можуть накладатися один на одного при високих швидкостях рахунку. До корисного сигналу домішується темновий рахунок самого детектора, електронні шуми тракту, мережеві наводки на частоті 50 Гц. Працювати з такими сигналами на класичному лабораторному обладнанні, звичайних цифрових осцилографах загального призначення, складно через обмежений об'єм буферної пам'яті, відсутність гнучкого керування серіями записів і повну відсутність вбудованих засобів пошуку імпульсів та статистичної обробки.

Для цих задач сучасною стандартною аппаратурою є спеціалізовані waveform-цифровайзери з частотою дискретизації до гігасемплів за секунду і об'ємом буферної пам'яті на рівні мільйонів відліків. Прикладом такого приладу є цифровайзер CAEN DT5761 з частотою дискретизації 4 GS/s, розрядністю 10 бітів і об'ємом on-board буфера близько восьми мільйонів відліків [4]. Виробник

постачає до цифровайзера комплект програмних бібліотек, через які реалізується інтерфейс керування і зчитування даних, а також готову утиліту WaveDump для базового перегляду форми сигналу і збереження вибірок [5]. Однак можливості цієї утиліти обмежені первинною діагностикою тракту і не охоплюють повний цикл лабораторної роботи з фотонним детектором. WaveDump не зберігає метаданих запису у файлі, не дозволяє гнучко керувати серіями вимірювань з структурованою організацією каталогів, а також, не містить вбудованих засобів пошуку імпульсів і обчислення їх характеристик. Для повноцінного дослідження вимагається або окремий набір скриптів пост-обробки, або зовсім інший програмний інструмент.

Постає задача розробки програмного комплексу, який покривав би повний цикл лабораторної роботи з кремнієвим фотопомножувачем — від налаштування цифровайзера через серійний збір даних до пошуку імпульсів і експорту результатів. Такий комплекс має забезпечити гнучкий контроль над усіма параметрами захоплення, реалізувати програмний пошук імпульсів з робастними алгоритмами оцінки базової лінії та шуму, керувати експериментальними каталогами зі збереженням метаданих та мати можливість автономно працювати без обладнання за рахунок реалістичного симулятора. У даній роботі описано побудову саме такого комплексу для цифровайзера CAEN DT5761 у зв’язці з кремнієвим фотопомножувачем у складі модуля з керованим високовольтним живленням Hamamatsu C11204-01 [6].

З огляду на вищевикладене, метою роботи є розробка спеціалізованого програмного забезпечення для реєстрації оптичних сигналів з фотонною роздільною здатністю за допомогою високошвидкісного аналого-цифрового перетворювача CAEN DT5761, що поєднує гнучке керування параметрами захоплення з вбудованими алгоритмами цифрової обробки імпульсів і можливістю автономної роботи без фізичного підключення до обладнання, завдяки реалістичному симулятору сигналу детектора.

1. ОГЛЯД ЛІТЕРАТУРИ ТА ТЕХНІЧНИХ ПЕРЕДУМОВ

У цьому розділі викладено фізичні засади реєстрації оптичних сигналів з фотонною роздільною здатністю, розглянуто принципи роботи кремнієвих фотопомножувачів як актуального типу детекторів одиничних фотонів. Проаналізовано вимоги до швидкісних аналого-цифрових перетворювачів для коректного захоплення коротких імпульсних сигналів. Окремо розглянуто методи цифрової обробки імпульсів від детекторів і виконано огляд наявних програмних рішень з ідентифікацією їх обмежень для дослідницьких задач, до яких належить і дана робота.

1.1. ФОТОННА РЕЄСТРАЦІЯ: ІНТЕГРУЮЧИЙ І ЛІЧИЛЬНИЙ ПІДХОДИ

Реєстрація потоку електромагнітного випромінювання у вимірювальних приладах здійснюється за двома принципово різними схемами. Перша полягає у накопиченні всіх носіїв заряду, породжених поглинутими у чутливому об’ємі детектора фотонами протягом певного інтервалу часу. Накопичений заряд інтерпретується як інтегральна інтенсивність потоку. Така схема реалізована у переважній більшості класичних детекторів — від газових іонізаційних камер і фотодіодів до сцинтиляційних блоків, що використовуються у класичній рентгенівській комп’ютерній томографії.

Друга схема відрізняється тим, що замість сумарного накопичення заряду електроніка реєстрації виокремлює сигнал від кожного окремого фотона і обробляє його як самостійну подію. Для кожної такої події може визначатися момент її настання, амплітуда сформованого імпульсу та інші характеристики. Підсумкове значення інтенсивності отримується підрахунком кількості подій у заданому інтервалі часу, причому додатково стає доступним розподіл подій за енергією. Це і є фотонна, або лічильна, реєстрація [1].

Принципова перевага лічильного підходу полягає у спектральній селективності. Оскільки амплітуда сформованого імпульсу пропорційна енергії

взаємодії, кожну подію можна віднести до певного енергетичного інтервалу. Завдяки цьому з одного вимірювання отримується не одне число, що відповідає інтегральному рахунку, а повний амплітудний спектр. Інша перевага — можливість енергетичного відсічення електронних шумів, тобто події з амплітудою нижче заданого порогу не реєструються як фотонні, що покращує відношення сигнал/шум у порівнянні з інтегруючою схемою.

Найяскравішим прикладом практичного впровадження лічильного підходу стала фотонно-лічильна комп'ютерна томографія. Замість сцинтиляційного блока з фотодіодом, що виконує інтегрування світла, у PCCT-системах застосовуються прямі напівпровідникові детектори на телуриді кадмію, телуриді кадмію-цинку або кремнії. Високовольтне зміщення збирає електронно-діркові пари, породжені кожним рентгенівським фотоном, безпосередньо на пікселі-аноді, де зчитувальний інтегрований канал формує імпульс із амплітудою, пропорційною енергії фотона [7, 8]. Така архітектура дозволяє отримати вищу просторову роздільну здатність за рахунок дрібнішого пікселя без оптичних розділювачів, виконувати спектральні реконструкції на основі енергетичних бінів та зменшити дозу опромінення пацієнта завдяки відсіченню теплових шумів.

Декілька серійних PCCT-систем уже пройшли клінічну сертифікацію Управління з контролю за продуктами і ліками США і використовуються у клініках Європи та Північної Америки — NAEOTOM Alpha від Siemens Healthineers, OmniTom Elite від Samsung Healthcare, Extremity Scanner System від MARS Bioimaging, Photonova Spectra від GE HealthCare [1]. Технологія залишається коштовною, але активно розвивається, і у середньостроковій перспективі може суттєво потіснити класичні системи з інтегруючими детекторами. Принципи лічильної реєстрації не обмежені рентгенівським діапазоном. У видимому, ультрафіолетовому та інфрачервоному світлі аналогічні задачі розв'язують лавинні фотодіоди у режимі Гайгера, фотопомножувачі різних типів та мікроканальні пластини. У всіх цих приладах

кінцевий сигнал формується як послідовність коротких імпульсів, кожен з яких відповідає поглинанню одного або кількох фотонів. У сучасній лабораторній практиці одним з найпоширеніших детекторів цього класу є кремнієвий фотопомножувач, якому присвячено наступний підрозділ.

1.2. КРЕМНІЄВИЙ ФОТОПОМНОЖУВАЧ ЯК ДЕТЕКТОР ОДИНИЧНИХ ФОТОНІВ

Кремнієвий фотопомножувач у науковій літературі зазвичай позначається SiPM, а у каталозі Hamamatsu Photonics — терміном MPPC. Це твердотільний детектор світла, що поєднує здатність реєструвати окремі фотони з компактними розмірами кристала, низькою напругою живлення та нечутливістю до магнітних полів [2]. Поява SiPM у середині 2000-х років стала альтернативою класичним вакуумним фотопомножувачам і відкрила можливість використання фотонно-лічильних детекторів там, де габарити, енергоспоживання або наявність магнітних полів, робили вакуумні рішення непридатними [9].

Конструктивно SiPM являє собою матрицю мікроскопічних лавинних фотодіодів, об’єднаних у спільне коло. Кожен з таких елементарних діодів, що називається мікропікселем, працює у режимі Гайгера. На pn-перехід прикладається зворотна напруга, яка перевищує напругу пробою. У такому стані поглинутий фотон, породивши електронно-діркову пару в активному об’ємі, ініціює лавинне розмноження носіїв заряду, кероване сильним полем у переході. Лавина обмежується послідовним резистором гасіння, увімкненим у коло кожного мікропікселя [3].

Принциповою особливістю SiPM є те, що окремий мікропіксель функціонує у бінарному режимі — «спрацював або ні», незалежно від кількості фотонів, що його ініціювали. Аналогова інформація про кількість одночасно поглинутих фотонів виникає з паралельного з’єднання всіх мікропікселів матриці. Якщо у певний момент часу спрацювали два пікселі, то сумарний струмовий імпульс на виході матриці буде вдвічі вищим за одиничний, якщо три

— у три рази, і так далі. Тому амплітудний спектр такого детектора має квантовану структуру з дискретними рівнями, що відповідають одночасній реєстрації одного, двох, трьох і більшої кількості фотоелектронів [10].

Форма одиничного імпульсу SiPM визначається переважно параметрами кола гасіння. Передній фронт формується швидко — за одиниці наносекунд, відповідно до часу розвитку лавини. Спадання має експоненційний характер з постійною часу, заданою добутком ємності мікропікселя на опір гасіння, і зазвичай становить від десятків до сотень наносекунд залежно від конкретної моделі [3]. На осцилограмі такий імпульс виглядає як коротке відхилення від базового рівня, причому за умов прийнятої у даній роботі схеми зчитування полярність відхилення є негативною — імпульс спрямований вниз від рівня спокою.

Окрім корисних подій, що відповідають поглинанню фотонів, SiPM генерує власні імпульси через теплові флуктуації у кристалі, так зване «явище темнового рахунку». Електрони, що термічно емітуються у активний об’єм мікропікселя, ініціюють лавини, які не відрізнити за формою від справжніх фотонних подій [10]. Швидкість темнового рахунку для типового SiPM з активною площею порядка одного квадратного міліметра при кімнатній температурі лежить у діапазоні від кількох сотень кілогерц до одиниць мегагерц. Це число задає природний нижній фон для будь-яких вимірювань.

До іншого роду спотворень корисного сигналу належать оптичний кросток і післяімпульси. Оптичний кросток виникає тоді, коли лавина в одному мікропікселі породжує вторинні фотони, що поглинаються у сусідніх мікропікселях і запускають лавини у них. Результатом є завищення амплітуди події відносно справжньої кількості фотонів, що її ініціювали. Післяімпульси — це затримані лавини через захоплення носіїв заряду пастками у кристалі і їх повільне вивільнення після основної події [3, 10].

У роботі застосовується детекторний модуль на основі SiPM, об’єднаний у спільну збірку з джерелом високовольтного живлення Hamamatsu C11204-01. Це програмно-кероване джерело з прецизійною стабілізацією напруги і температурною компенсацією, що критично для відтворюваності коефіцієнта підсилення детектора [6]. Конструктивні особливості даного модуля розглядаються у підрозділі 2.2.

1.3. ШВИДКІСНІ АНАЛОГО-ЦИФРОВІ ПЕРЕТВОРЮВАЧІ ДЛЯ РЕЄСТРАЦІЇ ІМПУЛЬСНИХ СИГНАЛІВ

Стохастична природа потоку фотонів і коротка тривалість одиничного імпульсу SiPM накладають специфічні вимоги на апаратний тракт оцифрування. Передній фронт одиничного імпульсу SiPM формується за одиниці наносекунд, повна ширина на половині максимуму у типових моделях лежить у діапазоні від трьох до десяти наносекунд. За критерієм щонайменше десяти відліків на півширину імпульсу частота дискретизації має становити не нижче одного гігасемпла за секунду, а у типовому випадку — від двох до чотирьох GS/s.

Така частота досягається в АЦП паралельного перетворення. Архітектура цього типу побудована на масиві компараторів, що одночасно порівнюють вхідну напругу з 2^n-1 опорними рівнями. Кожен з компараторів видає бінарний результат порівняння на спільний пріоритетний шифратор, що формує вихідний цифровий код. Завдяки паралельності всіх операцій, час перетворення обмежений лише власною швидкодією компараторів і шифратора, що дозволяє досягати частот вибірки у гігасемплах за секунду. Платою за швидкодію є експоненційне зростання кількості компараторів зі збільшенням розрядності, що практично обмежує Flash-перетворювачі розрядністю на рівні від 8 до 12 бітів.

Для задач фотонного рахунку обмеження розрядності не є критичним. У режимі Гайгера амплітуда одиничного імпульсу від мікропікселя SiPM становить десятки мілівольт після підсилення. У такому діапазоні

десятирозрядна шкала з 1024 рівнями забезпечує достатню роздільну здатність для впевненого розрізнення сусідніх рівнів однофотоелектронного відгуку.

Принципово важливою є наявність на платі АЦП внутрішньої кільцевої пам'яті. У такій схемі вхідний потік відліків безперервно записується у кільцевий буфер, причому при заповненні нові відліки переписують найстаріші. Коли подія викликає спрацювання тригера, поточний стан буфера фіксується, і доступним для зчитування виявляється весь записаний фрагмент сигналу — як до моменту тригера, так і після. У термінах налаштувань плати кількість відліків, збережених до моменту тригера, задається параметром pre-trigger, а після — параметром post-trigger.

Сучасні цифровайзери підтримують три основні режими формування тригера. Програмний тригер ініціюється командою від комп’ютера і використовується для діагностики та для записів, у яких інтерес становить базовий рівень сигналу без прив'язки до подій. Тригер за рівнем сигналу спрацьовує тоді, коли вхідна напруга перетинає заданий поріг у заданому напрямку — це основний режим для реєстрації фотонних подій. Зовнішній тригер реалізується через спеціалізований вхід плати на стандартному рівні сигналу TTL і дозволяє синхронізувати запис із зовнішнім приладом — джерелом світлових імпульсів або іншим детектором у схемі збігів [11]. Усі три режими у даній роботі підтримуються розробленим програмним комплексом і протестовані на реальній платі. Конкретні параметри плати CAEN DT5761, що застосовується у роботі, детально розглядаються у підрозділі 2.1.

1.4. ЦИФРОВІ МЕТОДИ ОБРОБКИ СИГНАЛІВ ВІД ДЕТЕКТОРІВ

Перехід від аналогового тракту обробки імпульсів детекторів до цифрового, що відбувся у радіоелектроніці протягом 1990–2000-х років, докорінно змінив підхід до задачі. У класичній аналоговій схемі обробка виконувалася послідовністю спеціалізованих модулів, тобто формувачем імпульсів, дискримінатором, амплітудно-цифровим перетворювачем та

лічильником подій. Кожен такий модуль реалізовував свою функцію апаратно з фіксованими параметрами. Перехід на оцифрування форми кожної події з подальшою повною програмною обробкою дозволив відмовитися від цієї жорсткості. Уся сукупність характеристик сигналу обчислюється з єдиного цифрового масиву відліків, причому склад і параметри обчислень можуть змінюватися без втручання у апаратну частину [12].

Сучасний цифровий підхід до обробки імпульсних сигналів детекторів, відомий як Digital Pulse Processing, охоплює широкий спектр задач — від простого підрахунку подій до спектрального аналізу, реконструкції форми, фільтрації шуму та виявлення накладень. У рамках задачі однофотонного рахунку набір ключових операцій є відносно компактним, а саме: оцінка рівня спокою сигналу, оцінка рівня шуму, виявлення моментів перетину сигналом порогу детекції, обчислення характеристик кожного знайденого імпульсу, відсіювання шумових спрацювань та їх агрегатна статистична обробка [13].

Особливу увагу заслуговує задача оцінки базової лінії та шуму. У класичних реалізаціях для цього використовувалися середнє арифметичне і середньоквадратичне відхилення відліків у деякому опорному вікні. Однак ці статистики чутливі до викидів. Якщо у опорне вікно випадково потраплять справжні імпульси, оцінки виявляться завищеними. Сучасні алгоритми використовують замість них робастні оцінки: медіану та медіану абсолютних відхилень від медіани, у англомовній літературі відому під скороченням MAD. Для нормально розподілених даних значення MAD пов'язане зі стандартним відхиленням множником, що чисельно дорівнює приблизно 1.4826 [14]. Медіана і MAD є робастними оцінками з пробійною точкою у половину. Тобто, мається на увазі, що навіть якщо близько половини відліків у опорному вікні будуть викидами, обидві оцінки залишаться близькими до значень, які вони мали б при відсутності викидів.

Поріг детекції імпульсів зазвичай задається кратно оціненому шуму. У фізиці частинок усталеним вибором є поріг на рівні п'яти стандартних відхилень над базовою лінією [15]. Цей вибір ґрунтується на тому, що для гаусівського шуму ймовірність випадкового перетину 5σ-порогу одним відліком становить приблизно $2.87 \cdot 10^{-7}$, що для типового запису довжиною у сто тисяч відліків дає очікувану кількість хибних спрацювань менше однієї за один запис.

Знайдені імпульси характеризуються чотирма основними величинами. Амплітуда дорівнює максимуму відхилення сигналу від базової лінії в межах імпульсу. Повна ширина імпульсу на рівні половини амплітуди дає інтегральну характеристику тривалості події. Площа, тобто інтеграл сигналу над базовою лінією у межах імпульсу, у фізичному сенсі пропорційна повному заряду, накопиченому детектором у даній події, і для SiPM є найточнішим індикатором кількості фотоелектронів. Час наростання визначається як проміжок між моментами досягнення сигналом рівнів 10 і 90 відсотків амплітуди — це класична електронна характеристика швидкодії детектора [15].

При практичній реалізації пошуку імпульсів потрібно враховувати два явища, що породжують артефакти. Перше — короткочасне опускання сигналу нижче порогу всередині одного фізичного імпульсу через флуктуації шуму на хвості. Без додаткової обробки алгоритм буде сприймати одну подію як дві окремі. Стандартним рішенням є склеювання імпульсів, розділених малим часовим інтервалом — операція merge gap. Друге — короткочасні шумові викиди, що випадково перевищують поріг лише на один-два відліки. Такі викиди не відповідають реальним фізичним подіям і відсіваються за критерієм мінімальної ширини імпульсу.

1.5. ОГЛЯД НАЯВНИХ ПРОГРАМНИХ РІШЕНЬ

Компанія CAEN, як виробник цифровайзера DT5761, забезпечує користувачів декількома програмними інструментами для роботи з обладнанням. Усі вони побудовані на спільному стеку бібліотек. У основі стеку

лежить системний драйвер USB. Над ним розташована бібліотека CAENVMELib, що надає уніфікований інтерфейс доступу до регістрів приладу. На наступному рівні знаходиться комунікаційна бібліотека CAENComm. І найвищий рівень — це бібліотека CAENDigitizer, що надає прикладному коду набір функцій для керування цифровайзером як приладом [11, 16].

Готовою прикладною програмою, що постачається разом з обладнанням, є утиліта WaveDump. Вона призначена для діагностики плати, перегляду форми сигналу в реальному часі і збереження вибірок у файл [5]. WaveDump є ефективним інструментом первинної перевірки працездатності тракту, проте при детальному ознайомленні з її можливостями можна виявити три базові обмеження, через які вона не може повноцінно замінити спеціалізовану дослідницьку програму.

Перше обмеження — формат зберігання даних. WaveDump записує сигнал у вигляді сирого бінарного або текстового потоку відліків без структурованого заголовка з метаданими. Параметри запису у самому файлі не зберігаються і мусять документуватися окремо. У дослідницькій практиці, де результати накопичуються серіями за різних умов, відсутність вбудованих метаданих ускладнює пізнішу інтерпретацію даних і знижує відтворюваність результатів.

Друге — обмежені засоби керування експериментальними каталогами. WaveDump зберігає всі записи у фіксований файл, що перезаписується при кожному новому запуску, або у файли з простою послідовною нумерацією без можливості гнучкого розділення серій вимірювань. У сценаріях з десятками або сотнями записів за різних умов користувач змушений вручну переміщувати файли та вести зовнішній журнал.

І на кінець, третє і найсуттєвіше — відсутність вбудованих засобів пост-обробки. WaveDump виконує лише захоплення і збереження сирих даних, тоді як обчислення характеристик імпульсів, побудова амплітудних спектрів, статистичний аналіз серій записів покладаються повністю на користувача. У типовому робочому процесі це означає необхідність розробляти окремий набір

скриптів для розпаковки сирих файлів, фільтрації базової лінії, пошуку імпульсів та обчислення параметрів кожної події. Кожна дослідницька група розв’язує цю задачу самостійно, що породжує множину несумісних реалізацій і знижує загальну відтворюваність наукових результатів.

З огляду на ці обмеження, для повноцінної дослідницької роботи, з цифровайзером DT5761 у задачі однофотонного рахунку, доцільно створити спеціалізоване програмне забезпечення, яке працює напряму з бібліотекою CAENDigitizer на хост-комп’ютері, надає гнучкий контроль над усіма параметрами захоплення, реалізує програмну обробку імпульсів, керує експериментальними каталогами і зберігає метадані разом з даними. Саме така програма розробляється у даній роботі.

Окремо варто зазначити вибір середовища розробки. Бібліотека CAENDigitizer постачається у скомпільованій формі для мов сімейства C і C++, що історично робило ці мови природним вибором для побудови надбудов над нею. Однак сучасні наукові обчислення зміщуються до Python через зрілість його екосистеми бібліотек для числової обробки, побудови графічних інтерфейсів і візуалізації даних. Для зв’язку Python-коду з C-бібліотеками існує стандартний модуль ctypes, який дозволяє завантажувати динамічні бібліотеки у простір процесу і викликати їх функції безпосередньо з Python-коду, передаючи параметри через C-сумісні типи даних [17, 18]. Завдяки тому, що всі ресурсомісткі операції виконуються всередині бінарної бібліотеки, продуктивність такого рішення знаходиться на рівні C-реалізації, а складність прикладного коду на порядок нижча.

1.6. ВИСНОВКИ ДО РОЗДІЛУ

У розділі викладено фізичні та технічні засади реєстрації оптичних сигналів з фотонною роздільною здатністю. Показано, що лічильний підхід забезпечує спектральну селективність, енергетичне відсічення шумів і потенційно вищу просторову роздільну здатність, що зумовлює його поширення у сучасній медичній діагностиці та лабораторних дослідженнях.

Розглянуто принцип роботи кремнієвого фотопомножувача як актуального типу детектора одиничних фотонів. Описано квантовану структуру вихідного сигналу, основні характеристики імпульсного відгуку, наносекундну тривалість і експоненційний спад, та властиві для всіх SiPM джерела шуму: темновий рахунок, оптичний кросток і післяімпульси.

Проаналізовано вимоги, а саме, що детекторні сигнали такого типу ставлять до апаратного тракту реєстрації. Обґрунтовано необхідність частоти дискретизації на рівні гігасемплів за секунду, проаналізовано переваги архітектури паралельного перетворення, описано принцип кільцевої пам'яті з можливістю передреєстрації передісторії події, виокремлено три режими формування тригера.

Систематизовано основні операції цифрової обробки імпульсних сигналів. Обґрунтовано переваги робастних оцінок базової лінії через медіану і шуму через MAD-статистику над класичними оцінками. Описано вибір порогу детекції за критерієм п’ять сигм, перелічено чотири основні характеристики імпульсу: амплітуду, повну ширину на половині максимуму та площу і час наростання.

Виконано огляд штатного програмного забезпечення, що постачається виробником цифровайзера, виявлено три суттєві обмеження утиліти WaveDump у задачах дослідницького характеру, а саме: відсутність вбудованих метаданих у форматі зберігання, обмежені засоби керування експериментальними каталогами, відсутність засобів пост-обробки. Обґрунтовано доцільність розробки власного спеціалізованого програмного забезпечення мовою Python з використанням ctypes для зв’язку з бібліотекою CAENDigitizer виробника, що і становить мету даної магістерської роботи.

2. АПАРАТНЕ ЗАБЕЗПЕЧЕННЯ І АРХІТЕКТУРА ПРОГРАМНОГО КОМПЛЕКСУ

У розділі описано апаратну частину експериментальної установки — високошвидкісний цифровайзер та детекторний модуль на основі кремнієвого фотопомножувача. Викладено будову програмного стеку виробника обладнання і підхід до інтеграції цього стеку у Python-середовище. Сформульовано загальну архітектуру розробленого програмного комплексу з обґрунтуванням ключового архітектурного рішення — абстракції пристрою-джерела даних.

2.1. ВИСОКОШВИДКІСНИЙ ЦИФРОВАЙЗЕР CAEN DT5761

Розглянемо засіб оцифрування сигналу від детектора, що застосовується у даній роботі — настільний цифровайзер CAEN DT5761 виробництва італійської компанії CAEN S.p.A. Зовнішній вигляд приладу наведено на Рис. 2.1.

Рис. 2.1. *Зовнішній вигляд цифровайзера CAEN DT5761. Прилад виконаний у настільному корпусі червоного кольору, що є характерною ознакою лінійки CAEN Desktop Digitizers. На передній панелі розташовані оптичний роз'єм LINK, роз'єми синхронізаційних сигналів CLK IN, GPO, TRG IN, GPI, лінійка світлодіодів стану (PWR, USB, TRG, BUSY) і USB-роз'єм для зв'язку з комп'ютером та аналоговий вхід каналу CH0.*

Прилад належить до класу waveform-цифровайзерів, що записують безперервний фрагмент аналогового сигналу заданої тривалості з високою частотою вибірки, на відміну від класичних спектрометричних АЦП, які видають

лише одне число, амплітуду піка, на кожну зареєстровану подію. Така архітектура запису надає повну часову інформацію про форму кожного імпульсу і дозволяє виконувати програмну обробку, склад якої не обмежений можливостями апаратури [4]. Повний перелік технічних характеристик плати наведено у Додатку Б.

Архітектурно перетворювач DT5761 побудований за принципом паралельного перетворення на основі масиву компараторів з пріоритетним шифратором. Завдяки паралельності всіх операцій час одного перетворення обмежений лише власною швидкодією компонентів аналогової частини, що уможливлює досягнуту частоту 4 GS/s. Розрядність 10 бітів задає 1024 дискретних рівнів представлення сигналу. Це число можна вважати помірно низьким на тлі спектрометричних АЦП, де поширені 12 і 14 бітів, однак для задач реєстрації імпульсних сигналів від SiPM такої розрядності достатньо. Амплітуда одиничного однофотоелектронного імпульсу після підсилення становить десятки мілівольт, що в одиницях молодшого розряду АЦП дає кілька десятків кодів, відповідно, цього цілком вистачає для впевненого розрізнення сусідніх рівнів однофотонного, двофотонного і трифотонного відгуку.

Аналоговий вхідний тракт плати побудований на стандартний імпеданс 50 Ом з вхідним діапазоном 1 Vpp. Окремої уваги заслуговує наявність 16-розрядного цифро-аналогового перетворювача, що керує постійним зміщенням вхідного сигналу. Цей DAC дозволяє програмно зміщувати робочу точку АЦП у межах повного динамічного діапазону, що використовується для оптимального розташування сигналу на шкалі квантування. Для SiPM, у якого імпульси спрямовані у негативний бік від рівня спокою, зсув встановлюється близько верхньої межі діапазону, щоб уся амплітуда імпульсу впевнено лежала у межах шкали без обрізання. Конкретне значення цього параметра у розробленому програмному комплексі за замовчуванням становить 32768 одиниць шістнадцятирозрядного простору, що відповідає середині шкали.

Внутрішня пам'ять плати організована за принципом кільцевого буфера. Безперервний потік відліків від АЦП записується у кільцевий буфер каналу так, що при заповненні нові відліки переписують найстаріші. Коли надходить сигнал тригера, поточний стан буфера фіксується, і користувачу для зчитування доступний фрагмент сигналу заданої довжини з налаштовуваним співвідношенням передісторії і післяісторії. Сумарний об’єм буфера у DT5761 становить близько восьми мільйонів відліків, що при частоті 4 GS/s відповідає максимальній тривалості одного запису близько двох мілісекунд.

Передня панель приладу містить роз'єми BNC для входів каналу, роз'єми для зовнішнього тригера TRG-IN, синхронізаційного виходу S-IN і CLK для зовнішнього тактового сигналу, USB-порт для зв’язку з комп’ютером. На задній панелі розташовано роз'єм живлення +12 В. Світлодіодна індикація передньої панелі сигналізує про подачу живлення, наявність USB-зв'язку, активність тригера і стан зайнятості плати. Коректне керування цими індикаторами під час циклу збору даних є додатковим інструментом діагностики правильності роботи програмного комплексу.

2.2. ДЕТЕКТОРНИЙ МОДУЛЬ З КРЕМНІЄВИМ ФОТОПОМНОЖУВАЧЕМ

Джерелом сигналу у експериментальній установці є детекторний модуль, у якому кремнієвий фотопомножувач об’єднаний у спільну збірку з джерелом високовольтного живлення Hamamatsu C11204-01. Зовнішній вигляд модуля наведено на Рис. 2.2.

Рис. 2.2. *Детекторний модуль на основі кремнієвого фотопомножувача. На спільній друкованій платі змонтовані сам SiPM-сенсор Hamamatsu MPPC (зелений квадратний модуль ліворуч на платі, з білим прямокутником по центру), програмно-кероване джерело високовольтного живлення Hamamatsu C11204-01 (сірий блок зверху по центру), розв'язувальні та узгоджувальні елементи аналогового тракту. Внизу плати розташовано SMA-роз'єм для виходу сигналу до цифровайзера.*

Кремнієвий фотопомножувач у цьому модулі — є одним з типових SiPM серії Hamamatsu MPPC з активною площею порядка одного квадратного міліметра, поділеною на матрицю мікропікселів з кроком кілька десятків мікрометрів. Конструкція мікропікселя і принцип формування лавини при поглинанні фотона детально розглянуті у підрозділі 1.2, тому тут зупинимося лише на тих характеристиках, що безпосередньо впливають на побудову апаратного тракту і параметри програмної реєстрації.

Робочою напругою такого детектора є зворотна напруга на pn-переході, що перевищує напругу пробою на 1–5 вольт залежно від конкретної моделі. Загальна напруга зміщення лежить у діапазоні від 50 до 80 вольт. Величина перевищення робочої напруги над пробоєм безпосередньо визначає коефіцієнт підсилення детектора — чим більше перевищення, тим інтенсивніша лавина і більший заряд накопичується у відгуку на одиничний фотон. Однак зі зростанням перевищення

збільшуються і паразитні ефекти — темновий рахунок, оптичний кросток і післяімпульси. Оптимальна робоча точка вибирається як компроміс між цими протилежними тенденціями, причому для кожного конкретного екземпляра детектора вона відрізняється.

Окрім робочої точки, на коефіцієнт підсилення SiPM сильно впливає температура. Підвищення температури активного об’єму на один градус Кельвіна знижує підсилення на 1–2 відсотки за рахунок зміни напруги пробою. У лабораторних умовах без термостабілізації коливання температури кристала протягом тривалого вимірювання можуть досягати кількох градусів через нагрів від оточуючої електроніки і власне детектора, що призводить до повзкого дрейфу амплітуди однофотоелектронного відгуку.

Саме для розв’язання цієї задачі у склад детекторного модуля включене джерело Hamamatsu C11204-01 — програмно-кероване високовольтне джерело живлення з прецизійною стабілізацією вихідної напруги і вбудованою температурною компенсацією [6]. Воно вимірює температуру SiPM через окремий датчик і автоматично коригує вихідну напругу так, щоб перевищення над пробоєм залишалося постійним незалежно від температури кристала. Завдяки такій активній компенсації коефіцієнт підсилення детектора стабілізується на рівні десятих часток відсотка, що дозволяє накопичувати тривалі вимірювання без зміщення положень піків амплітудного спектра.

Електричне коло після SiPM містить розв'язувальний конденсатор, що блокує постійну складову високовольтного зміщення, і узгоджувальний резистор 50 Ом, що формує імпеданс лінії передачі до входу цифровайзера. У результаті сигнал, поданий на вхід DT5761, є послідовністю коротких негативних імпульсів наносекундної тривалості з амплітудою від кількох до десятків мілівольт у залежності від кількості фотонів у події і обраного коефіцієнта підсилення. Полярність сигналу у такій схемі підключення є негативною — кожна подія викликає короткочасне опускання напруги нижче рівня спокою. Цей факт враховується у налаштуваннях програмного комплексу.

2.3. ПРОГРАМНИЙ СТЕК CAEN І ІНТЕГРАЦІЯ З PYTHON

Розглянемо програмний інтерфейс, через який відбувається керування цифровайзером з прикладного коду. Виробник постачає його у вигляді багаторівневої ієрархії динамічних бібліотек, кожна з яких реалізує свій рівень абстракції над апаратурою.

На найнижчому рівні знаходиться драйвер USB-каналу. На операційних системах Windows він має назву CAENUSBdrvB і встановлюється окремим інсталятором з пакета постачання. Драйвер забезпечує розпізнавання плати операційною системою і передачу пакетів даних через шину USB. Безпосередньо з прикладного коду цей рівень не викликається — взаємодія з ним відбувається опосередковано через бібліотеки вищих рівнів. Над драйвером розташована бібліотека CAENVMELib, що реалізує універсальний інтерфейс до регістрів плати незалежно від типу фізичного зв'язку. Хоча її назва містить абревіатуру VME-стандарту шини, бібліотека покриває і USB-підключення, і інші типи каналів, які можуть використовуватися старшими моделями цифровайзерів CAEN [16]. Над нею побудована бібліотека CAENComm, що забезпечує комунікаційний рівень, тобто формування пакетів запитів і відповідей, обробку помилок протоколу. Найвищим рівнем стеку є бібліотека CAENDigitizer, що надає прикладному коду набір функцій, які трактують цифровайзер як прилад зі змістовним інтерфейсом, а не як набір регістрів [11]. Саме на функціях цієї бібліотеки побудовано прошарок зв’язку з апаратурою у розробленому комплексі.

Усі чотири рівні стеку постачаються у вигляді бінарних компонентів, скомпільованих для мов сімейства C і C++. Для викликів з прикладного коду їх потрібно завантажити у простір процесу і викликати функції з дотриманням сигнатур, описаних у заголовкових файлах. У середовищі Python для цієї задачі використовується стандартний модуль ctypes — частина основної бібліотеки мови, що не потребує встановлення додаткових пакетів [17, 18].

Розглянемо альтернативи такому підходу. Очевидним варіантом було б використання готової Python-обв’язки до CAEN, якщо така існує. Виробник постачає експериментальну обв’язку caen-felib, проте на момент розробки комплексу вона перебувала у стадії раннього розвитку і не покривала всіх потрібних функцій. Інший варіант — побудова всієї надбудови на мові C++ з подальшим вбудуванням Python-інтерпретатора для скриптової частини, що потребує значно складнішої системи збирання і не дасть виграшу в продуктивності для нашої задачі. Тому використання ctypes виявляється оптимальним компромісом між простотою реалізації і повнотою доступу до можливостей бібліотеки.

Модуль ctypes виконує функцію прошарку зв’язку між інтерпретатором Python і бінарними інтерфейсами C через прикладний бінарний інтерфейс операційної системи. Він надає набір типів даних, узгоджених з типами мови C — c_uint16, c_uint32, c_int32, POINTER, c_char_p — і функцію CDLL для завантаження динамічної бібліотеки. Після завантаження бібліотеки її експортовані функції стають доступними як атрибути об’єкта бібліотеки і викликаються звичайним Python-синтаксисом з передачею параметрів через C-сумісні типи.

Принципова перевага такої схеми у тому, що всі ресурсомісткі операції виконуються всередині скомпільованої C-бібліотеки. Це і обмін з драйвером USB, і копіювання великих блоків пам'яті, і безпосередньо саме перетворення сирого потоку байтів на структури подій. Python-код виступає лише як рівень керування — він формує послідовність викликів, передає параметри, перевіряє коди повернення, але не виконує жодного циклу по окремих відліках. Завдяки цьому продуктивність вирішення на основі ctypes практично не поступається C-коду, тоді як складність прикладного коду на порядок нижча.

2.4. АРХІТЕКТУРА РОЗРОБЛЕНОГО ПРОГРАМНОГО КОМПЛЕКСУ

Розглянемо загальну архітектуру розробленого програмного комплексу, що має назву Photon ADC Studio. Це настільний застосунок з графічним

інтерфейсом, що покриває повний цикл роботи з фотонним детектором — від налаштування цифровайзера до експорту результатів аналізу.

Код розділено на два пакети. Пакет core містить бекенд застосунку — всі модулі, що не залежать від графічного інтерфейсу. До нього входять абстракція пристрою-джерела даних, дві реалізації цієї абстракції (симулятор сигналу SiPM і драйвер реальної плати), рушій асинхронного збору даних, файловий ввід-вивід для розробленого бінарного формату, алгоритм пошуку імпульсів і менеджер експериментальних каталогів. Пакет gui реалізує інтерактивну частину застосунку на основі бібліотеки PyQt6 з підтримкою побудови графіків через бібліотеку pyqtgraph [19, 20].

Розділення на два пакети не є чисто естетичним рішенням. Оскільки gui залежить від core, але core нічого не знає про gui, бекенд застосунку можна повторно використовувати в інших контекстах. Наприклад, у пакетному скриптовому режимі для автоматизованих експериментів, без перебудови графічної частини. Така структура також спрощує тестування алгоритмічних частин, бо їх можна випробовувати без запуску всього застосунку.

Принциповим архітектурним рішенням, на якому будується вся структура комплексу, є абстракція пристрою-джерела даних. Замість того, щоб GUI і алгоритми обробки безпосередньо звертались до бібліотеки CAEN, вводиться проміжний шар, абстрактний клас ADCDevice, що оголошує мінімальний інтерфейс взаємодії з будь-яким пристроєм, здатним видавати потік відліків. Цей інтерфейс містить методи відкриття пристрою, його закриття, застосування параметрів конфігурації, переведення у режим очікування тригера, програмного формування тригера, зчитування записаної форми сигналу, зчитування неперервного фрагмента даних і отримання інформації про пристрій. Конкретні реалізації цього інтерфейсу, клас SiPMSimulator для симульованих даних і клас CAENDigitizer для реальної плати, успадковуються від ADCDevice і надають кожен свою деталізацію цих методів.

Завдяки такій абстракції вся решта коду застосунку працює виключно з типом ADCDevice і не знає, що насправді стоїть за ним. Для перемикання між симулятором і реальним приладом достатньо замінити один об’єкт у головному вікні застосунку. Ця заміна може відбутись як на старті програми за прапором командного рядка, так і у будь-який момент під час роботи через випадаючий список Backend у заголовку головного вікна — у такому разі весь стан застосунку зберігається, а змінюється лише джерело даних.

Це архітектурне рішення відоме як шаблон проектування «адаптер» у класифікації Гамми та інших авторів-засновників сучасної дисципліни проектування програмного забезпечення [21]. Воно вирішує одночасно дві задачі. Перша — забезпечення можливості розробки і відлагодження застосунку за умов відсутності доступу до реального обладнання, що особливо актуально під час перших етапів роботи, коли плата фізично не доступна автору. І друга — створення основи для масштабування комплексу на інші моделі цифровайзерів CAEN тієї ж лінійки без переписування решти коду, тобто, для підтримки нової моделі достатньо створити нову реалізацію ADCDevice, інкапсулюючи специфіку цієї моделі в межах одного класу.

Окрім абстракції пристрою, у пакеті core виокремлено ще декілька самостійних модулів з власними чітко визначеними областями відповідальності. Модуль acquisition реалізує рушій асинхронного збору даних, що виконує запис у окремому потоці виконання, щоб тривалі операції захоплення не блокували графічний інтерфейс. Модуль file_io містить функції запису і читання файлів розробленого формату .phadc. Модуль signal_processing реалізує алгоритм пошуку імпульсів і обчислення їх характеристик. Модуль workspace керує експериментальними каталогами зі списком нещодавно використаних каталогів, що зберігається між сесіями застосунку. Модуль config містить структури конфігурації запису. Модуль device_factory забезпечує програмне створення об’єкта пристрою заданого типу.

Пакет gui побудований навколо головного вікна MainWindow, у якому розташовані чотири функціональні вкладки: осцилограф реального часу, конфігурація запису, менеджер каталогів і аналізатор записаних файлів. Кожна вкладка реалізована окремим модулем, tab_oscilloscope, tab_acquisition, tab_workspace, tab_analysis, і взаємодіє з ядром через спільний об’єкт пристрою та спільну конфігурацію запису. Модуль styles містить стильові правила інтерфейсу.

Детальна реалізація кожного з перелічених модулів — як на боці ядра, так і на боці GUI — розглядається у третьому розділі.

2.5. ВИСНОВКИ ДО РОЗДІЛУ

У розділі описано апаратну частину експериментальної установки та представлено загальну архітектуру розробленого програмного комплексу.

Розглянуто технічні характеристики цифровайзера CAEN DT5761, а саме: частоту дискретизації 4 GS/s, розрядність 10 бітів, динамічний діапазон 1 Vpp, наявність шістнадцятирозрядного DAC зміщення нуля, кільцеву буферну пам'ять об’ємом близько 8 MSamples, режими формування тригера від внутрішнього сигналу, від рівня вхідної напруги і від зовнішнього TTL-входу. Обґрунтовано адекватність цих характеристик для задач реєстрації одиничних фотонів кремнієвим фотопомножувачем.

Описано конструкцію детекторного модуля з кремнієвим фотопомножувачем Hamamatsu MPPC і інтегрованим у нього блоком стабілізованого живлення Hamamatsu C11204-01. Розглянуто механізм температурної компенсації коефіцієнта підсилення детектора, що принципово важливий для відтворюваності вимірювань.

Викладено будову програмного стеку виробника обладнання — чотирирівневу ієрархію з драйвера USB, бібліотек CAENVMELib, CAENComm і CAENDigitizer. Обґрунтовано підхід до інтеграції цього стеку у Python-

середовище через стандартний модуль ctypes, що забезпечує продуктивність на рівні C-коду при значно нижчій складності прикладного коду.

Сформульовано архітектуру розробленого програмного комплексу на основі шаблону проектування «адаптер». Введено абстрактний клас ADCDevice, реалізаціями якого є симулятор сигналу SiPM і драйвер реальної плати CAEN. Завдяки такій абстракції весь застосунок працює з єдиним інтерфейсом джерела даних, що уможливлює як розробку і відлагодження без обладнання, так і масштабування системи на інші моделі цифровайзерів цієї лінійки без переписування основного коду.

3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ

У попередньому розділі сформульовано загальну архітектуру комплексу та обґрунтовано ключове рішення — побудувати його навколо абстракції пристрою-джерела даних. У цьому розділі розглянемо, як ця абстракція втілена у конкретному коді, як над нею надбудовано драйверний шар для роботи з реальною платою CAEN DT5761 та повноцінний симулятор сигналу кремнієвого фотопомножувача, як описаний бінарний формат збереження записів, як побудована графічна частина застосунку і яким чином виконується пошук імпульсів у записаних файлах. Обсяг конкретних реалізацій великий, тому виклад побудуємо так, щоб для кожного компонента наголосити на ключовій ідеї і пояснити її без занурення у деталі, які виносяться у Додаток А.

3.1. АБСТРАКЦІЯ ПРИСТРОЮ-ДЖЕРЕЛА ДАНИХ

Розглянемо найперше архітектурне рішення, з якого виростає вся структура комплексу — введення проміжного шару між кодом, що працює з даними, і кодом, що знає про конкретний пристрій. Розглянемо альтернативи цьому рішенню, щоб зрозуміти, чому саме абстракція виявляється виправданою.

Найпрямолінійніший підхід — це організація коду без проміжного шару взагалі. Кожен компонент застосунку, графічний інтерфейс, рушій збору, файлові операції, безпосередньо викликає функції бібліотеки CAENDigitizer [11] і працює з масивами семплів, отриманими з реального обладнання. Такий код є мінімалістичним, без жодних надлишкових структур, і виглядає природним для першої версії застосунку. Проте він має критичну ваду — він жорстко зв’язує всю програму з конкретною моделлю обладнання. Будь-яка з трьох поширених модифікацій застосунку, додавання режиму симулятора для розробки без плати, підтримка іншої моделі цифровайзера, перехід на іншу бібліотеку виробника, потребувала б змін у всіх місцях коду, де відбуваються виклики CAEN-функцій. Це означає переписування графічного інтерфейсу, рушія збору, файлових

операцій разом з самою заміною джерела. Така архітектура жорстко непридатна для розвитку.

Інший варіант — використати готову бібліотеку Python-обв’язок до CAEN, якщо така існує. Виробник постачає експериментальну обв’язку caen-felib, проте на момент розробки комплексу вона перебувала у стадії раннього розвитку і не покривала всіх потрібних функцій. Крім того, навіть готова обв’язка не вирішує проблему зв’язування з конкретним обладнанням — вона лише замінює один тип зв’язку на інший.

Введемо абстрактний інтерфейс ADCDevice, що описує мінімальний набір операцій, які можна виконати з будь-яким пристроєм-джерелом цифрових відліків. Така абстракція є реалізацією шаблону проектування «адаптер» у класифікації Гамми та інших авторів-засновників сучасної дисципліни проектування програмного забезпечення [21]. Її задача, звести різнорідні джерела даних, реальну плату CAEN, програмний симулятор, можливі майбутні моделі обладнання — до єдиного інтерфейсу. Весь застосунок працює з цим інтерфейсом, нічого не знаючи про те, що насправді стоїть за ним.

Реалізуємо цей інтерфейс у модулі core/adc_base.py через стандартний механізм абстрактних класів Python з модуля abc. У мові немає окремої синтаксичної конструкції для інтерфейсів, проте через декоратор @abstractmethod досягаємо семантично еквівалентного ефекту, тобто, усі підкласи ADCDevice зобов’язані визначити перелічені у ньому методи, інакше створення об’єкта підкласу видає помилку при імпорті. Це дисциплінує реалізації — неможливо випадково забути визначити один з методів і отримати потім незрозумілу помилку при виклику.

Постає питання, який набір методів включити до інтерфейсу. Тут діє два протилежних принципи. З одного боку, інтерфейс має бути достатнім для всіх режимів роботи застосунку, інакше окремі його сценарії доведеться реалізовувати в обхід абстракції, що ламає її сенс. З іншого боку, інтерфейс не

повинен містити нічого, що було б специфічним для конкретного типу обладнання. Якби в інтерфейс просочилися деталі CAEN-API, прямий доступ до регістрів плати, типи з бібліотеки виробника, специфічні для CAEN режими роботи, симулятор не зміг би їх реалізувати без невластивих йому викривлень, а саме поняття абстракції втратило б цінність.

Зведемо до інтерфейсу дев'ять методів. Перші три, open, close та is_open, керують життєвим циклом сесії зв’язку з пристроєм. Метод configure приймає об’єкт конфігурації AcquisitionConfig з повним набором параметрів запису: довжиною, зміщенням нуля, передісторією, полярністю, режимом тригера та його рівнем. Методи arm і software_trigger керують процесом захоплення, а саме, перший переводить пристрій у стан очікування тригера, другий ініціює запис програмно. Методи read_waveform і read_live_chunk повертають дані. Перший — одну зареєстровану форму сигналу, інший — неперервний фрагмент сигналу без прив'язки до тригера. Допоміжний метод info повертає словник з ідентифікаційною інформацією про пристрій. Повний листинг абстрактного класу разом з описом усіх його методів (*див. Додаток А, фрагмент А.1*).

Крім самого інтерфейсу, у тому ж модулі визначаємо клас-структуру AcquiredWaveform, що повертається методом read_waveform. Структура несе масив відліків у вигляді numpy-масиву типу uint16, частоту дискретизації, параметри запису, що супроводжували захоплення, а також системний час реєстрації події. Така структура виглядає надлишковою. Здавалося б, достатньо повертати самий масив відліків, проте передача лише масиву призвела б до парадоксальної ситуації, коли код, що обробляє форму сигналу, мусив би мати додатковий доступ до конфігурації запису для масштабування часової осі. Натомість самодостатня структура AcquiredWaveform несе у собі все необхідне для подальшої обробки, а викликач не повинен пам'ятати, з якою конфігурацією виконувався запис.

Створення конкретного об’єкта пристрою у застосунку відбувається не прямим викликом конструктора відповідного класу, а через окрему функцію create_device у модулі core/device_factory.py. Це теж не випадкове рішення. Прямий виклик конструктора зв'язав би код, що вирішує, який бекенд використовувати. З самими класами реалізацій, SiPMSimulator і CAENDigitizer, довелося б імпортувати скрізь, де потрібно створити пристрій. Натомість фабрика інкапсулює цей вибір у одному місці. Функція приймає рядковий прапор бекенду, sim, caen або auto, і повертає вже готовий екземпляр відповідної реалізації. У режимі auto функція спочатку намагається створити об’єкт реального драйвера. У разі невдачі вона переключається на симулятор з виведенням попередження. Така схема дозволила запускати застосунок з єдиним прапором --backend auto як на робочій станції з підключеною платою, так і на будь-якому комп’ютері без неї. Особливу цінність ця можливість мала на початковому етапі роботи над комплексом, коли реальне обладнання ще не було доступне і вся розробка велася виключно на симуляторі.

3.2. РЕАЛІЗАЦІЯ ДРАЙВЕРА CAEN ЧЕРЕЗ CTYPES

Перейдемо до конкретної реалізації абстрактного інтерфейсу для роботи з реальною платою. Бібліотека CAENDigitizer, у якій зосереджено високорівневий інтерфейс роботи з цифровайзерами цієї лінійки, постачається виробником у вигляді скомпільованої динамічної бібліотеки з C-сумісним API [11]. Для її використання з Python-коду потрібен механізм виклику функцій зовнішніх бінарних бібліотек. Стандартним інструментом для цієї задачі у Python є модуль ctypes, який є частиною основної бібліотеки мови, що не потребує встановлення додаткових пакетів і дозволяє завантажувати у простір процесу динамічні бібліотеки з дотриманням C-сигнатур [18].

Реалізуємо клас CAENDigitizer у модулі core/adc_caen.py, що успадковується від ADCDevice. Перше нетривіальне рішення, яке доводиться приймати — це питання, у якому місці виконувати реальні зв’язки з

обладнанням. Очевидний варіант помістити їх безпосередньо у конструктор класу виявляється поганим. У режимі auto функція-фабрика create_device спочатку намагається створити об’єкт CAENDigitizer, і якби вже на цьому етапі починалися спроби завантажити бібліотеки або відкрити сесію, кожен запуск застосунку без обладнання видавав би помилку ще до старту графічного інтерфейсу. Користувач без CAEN-плати взагалі не зміг би запустити програму, навіть для роботи з симулятором.

Тому конструктор класу свідомо не виконує жодних дій з обладнанням. Він лише ініціалізує внутрішні поля і запам'ятовує налаштування, після чого повертає керування. Усі операції з реальною платою, завантаження бібліотек, відкриття сесії, опитування інформації, виносимо до методу open. Виклик open відкладається до моменту, коли користувач явно переходить у режим Hardware або коли функція-фабрика у режимі auto справді намагається активувати реальний пристрій. У цей момент open або успішно завершується, або генерує виключення, яке акуратно обробляється на верхньому рівні з м'яким перемиканням на симулятор.

Перша нетривіальна проблема, з якою стикається open — це особливості завантаження динамічних бібліотек на сучасних версіях Python для Windows. Починаючи з Python 3.8, стандартний механізм пошуку залежностей DLL через системну змінну PATH замінено на явний реєстр пошукових каталогів, керований функцією os.add_dll_directory [17]. Зміна була зроблена з міркувань безпеки, щоб довільні шляхи зі змінної PATH не призводили до завантаження невідомих бібліотек з повноваженнями Python-процесу. Проте побічним ефектом стала потреба явно реєструвати кожен каталог, з якого потрібно завантажити DLL. Це стосується не лише головної бібліотеки CAENDigitizer.dll, а й трьох її залежностей: CAENComm.dll, CAENVMELib.dll та PlxApi.dll. Усі чотири бібліотеки мають знаходитися у каталозі, зареєстрованому як пошуковий, інакше Windows-завантажувач видає помилку про відсутність залежності, причому без чіткої вказівки, якої саме.

Розглянемо альтернативи. Найочевиднішим підходом було б покладатися на стандартну установку CAEN-драйверів. Інсталятор виробника розміщує бібліотеки у фіксованих шляхах і вносить їх до загальної конфігурації системи. Однак цей варіант робить програму непрацездатною на комп'ютерах без виконаної інсталяції CAEN, навіть якщо самі бібліотеки фізично присутні у іншому каталозі. Іншим варіантом було б копіювати бібліотеки у каталог самого виконуваного файлу застосунку, що робить програму самодостатньою, але вимагає підтримки правильного набору бібліотек у дистрибутиві.

Реалізуємо комбінований підхід — функцію пошуку, що послідовно перевіряє три кандидати. Першим перевіряється каталог самого виконуваного файлу. Якщо чотири DLL лежать поруч з ним, використовується саме він. Другим перевіряється стандартний шлях установки CAEN на Windows — це дозволяє користуватися встановленими драйверами, якщо вони присутні. Третім — поточний робочий каталог, для випадків ручного запуску з консолі. Перший знайдений каталог з потрібним набором бібліотек реєструється функцією os.add_dll_directory, після чого виконується завантаження бібліотеки через ctypes.WinDLL. Реалізацію функції пошуку (*див. у Додатку А, фрагмент А.2*).

Відкриємо сесію з платою через виклик CAEN_DGTZ_OpenDigitizer2 [11]. Ця функція приймає тип лінка зв'язку, номер пристрою на лінку, адресу базового регістра і вказівник на змінну, у яку буде записано числовий ідентифікатор сесії — handle. У разі успіху функція повертає нульове значення, у разі помилки — від'ємний код, що відповідає одному з визначених у документації кодів. Тут постає важливе питання обробки помилок. Підхід «ігнорувати коди повернення» неприйнятний, оскільки, мовчазна помилка на ранньому етапі призведе до загадкових збоїв у подальшій логіці. Тому усі виклики функцій бібліотеки супроводжуємо перевіркою коду повернення. А саме, при ненульовому коді генеруємо виключення з текстовим описом помилки, отриманим за кодом через довідкову функцію CAEN_DGTZ_DecodeError. Завдяки цьому будь-яка несправність на рівні апаратури або драйверів виявляється негайно і трактується

як виключення з осмисленим повідомленням замість прихованого пошкодження стану.

Безпосередньо після відкриття сесії викличемо CAEN_DGTZ_GetInfo, функцію, що повертає структуру з ідентифікаційною інформацією про плату. З цієї структури вилучимо назву моделі, за якою програмно визначається частота дискретизації і нативна розрядність АЦП. Тут постає вибір: тримати ці параметри у внутрішній таблиці модуля або зчитувати їх з самої плати. Документація CAEN передбачає функції для зчитування частоти дискретизації з регістрів плати, проте їх поведінка неуніфікована між моделями цифровайзерів сімейства. Натомість таблиця у коді легко розширюється при появі нових моделей і дає прогнозовану поведінку. Зведемо у внутрішню таблицю модуля такі відповідності: для DT5761 встановлюється частота 4 GS/s і розрядність 10 бітів, для DT5751 — 1 GS/s і 14 бітів, для DT5730 — 500 MS/s і 14 бітів. Така автодетекція дозволяє одній збірці застосунку працювати з різними моделями цифровайзерів сімейства без переписування коду.

Перейдемо тепер до конфігурації параметрів запису. Метод configure транслює поля об’єкта AcquisitionConfig у послідовність викликів CAEN-функцій [11].

Довжина запису в семплах задається через CAEN_DGTZ_SetRecordLength, зміщення нуля — через CAEN_DGTZ_SetChannelDCOffset, полярність тригера — через CAEN_DGTZ_SetTriggerPolarity, розмір післяісторії — через CAEN_DGTZ_SetPostTriggerSize. Окрему увагу слід приділити взаємодії між шістнадцятирозрядним простором значень, прийнятим у застосунку, і нативним DAC-простором плати. У конфігурації застосунку зміщення нуля задається числом у проміжку від 0 до 65535 — це уніфікований простір, що використовується і у файлах розробленого формату, і у GUI. Проте функція CAEN_DGTZ_SetChannelDCOffset працює зі своїм власним діапазоном, тому

при виклику виконуємо лінійне масштабування. Завдяки цьому користувач у GUI оперує єдиним діапазоном значень незалежно від моделі плати.

Окремо налаштовується режим тригера, причому для трьох його варіантів використовуються різні функції бібліотеки. Програмний тригер вмикається через CAEN_DGTZ_SetSWTriggerMode, тригер за рівнем сигналу — через CAEN_DGTZ_SetChannelSelfTrigger з подальшим заданням порогу через CAEN_DGTZ_SetChannelTriggerThreshold, зовнішній — через CAEN_DGTZ_SetExtTriggerInputMode. Здавалося б природним вибрати лише одну з функцій при кожному виклику configure, проте такий підхід виявляється помилковим. Кожна з функцій вмикає або вимикає свій канал тригера незалежно від інших, і якщо при попередньому записі був активний один режим, а тепер користувач вибрав інший, неактивний раніше канал може зберігати своє вимкнене значення, проте раніше активований лишиться увімкненим. Результатом стає плата, що спрацьовує на події одночасно з двох джерел, спотворюючи характеристики тригера. Тому при кожному перемиканні режиму ми явно вимикаємо обидва неактивні канали і явно вмикаємо обраний. Реалізацію логіки перемикання (*див. у Додатку А, фрагмент А.3*).

Зчитування форми сигналу після спрацювання тригера виконується у методі read_waveform. Це найскладніша операція у драйвері, бо вона включає виділення пам'яті, опитування плати, декодування події і повернення результату. Послідовність дій така: виділяємо приймальний буфер у пам'яті комп’ютера через CAEN_DGTZ_MallocReadoutBuffer, запускаємо збір даних через CAEN_DGTZ_SWStartAcquisition, у циклі опитуємо плату через CAEN_DGTZ_ReadData до отримання непорожнього блоку, декодуємо подію через CAEN_DGTZ_GetEventInfo і CAEN_DGTZ_DecodeEvent, копіюємо масив відліків з декодованої події у numpy-масив, зупиняємо збір через CAEN_DGTZ_SWStopAcquisition і звільняємо буфер.

Останній крок, конвертація сирих байтів у numpy-масив, потребує окремого розгляду. Найочевиднішою реалізацією було б створити numpy-масив через np.array, що зкопіювало б елементи з C-буфера у нову область пам'яті, керовану Python. Для одиничних записів обсягом у сотні кілобайтів це здається прийнятним. Проте у режимах серійного запису, коли потрібно зберегти десятки таких записів за секунду, накладні витрати на копіювання стають помітними. Тому використовуємо альтернативний підхід — функцію numpy.frombuffer, яка не виконує фізичного копіювання даних, а лише створює представлення numpy-масиву над тим самим блоком пам'яті, що його заповнила бібліотека CAEN [22]. Таким чином, передача даних від плати у Python-код відбувається без жодного зайвого копіювання, індексація окремих відліків з Python-коду виконується безпосередньо процесором над C-буфером, з тією самою швидкодією, що і у чисто C-реалізації.

Для коректного зчитування семплів з плати DT5761 потрібно врахувати ще одну особливість. Нативна розрядність її АЦП — 10 бітів, проте бібліотека повертає семпли у шістнадцятирозрядних словах з даними у молодшому байті. Без додаткової обробки значення у numpy-масиві лежали б у діапазоні 0–1023, тоді як у застосунку прийнято єдиний шістнадцятирозрядний простір 0–65535. Уніфікація просторів значень спрощує реалізацію усіх верхніх рівнів застосунку, тобто, GUI, файлові операції, алгоритм аналізу не повинні знати про нативну розрядність кожної конкретної моделі плати. Тому при читанні з плати DT5761 виконуємо зсув значень на 6 бітів вліво, а при записі рівнів тригера на плату — зсув на 6 бітів вправо. Цей бітовий зсув виконується векторно над усім масивом за один виклик numpy і не вносить помітних накладних витрат.

3.3. СИМУЛЯТОР СИГНАЛУ КРЕМНІЄВОГО ФОТОПОМНОЖУВАЧА

Реалізуємо другу конкретну реалізацію абстрактного інтерфейсу — симулятор сигналу кремнієвого фотопомножувача. Така реалізація необхідна з двох незалежних причин. Перш за все, як зазначалось раніше, вона дозволяє

виконувати розробку і відлагодження застосунку у тих умовах, коли реальне обладнання фізично не доступне розробникові. Крім того, симулятор виявляється цінним і тоді, коли обладнання вже підключене, він забезпечує тестове джерело даних з відомими наперед характеристиками подій, що дозволяє перевіряти коректність роботи алгоритмів пошуку імпульсів шляхом порівняння одержаних результатів з очікуваними.

Постає питання, наскільки реалістичним має бути симулятор. Найпростіший варіант, генерувати чистий гаусівський шум з періодично доданими прямокутними імпульсами, дозволив би перевірити лише найзагальніші аспекти роботи застосунку, а саме, проходження даних через файлові операції, побудову графіків, спрацювання детектора на перевищення порогу. Проте на таких сигналах принципово неможливо було б відлагодити алгоритм пошуку імпульсів з його реальними особливостями, оцінкою базової лінії, обробкою хвостів імпульсів, фільтрацією шумових викидів. Тому реалізуємо симулятор настільки реалістичним, наскільки це можливо без надмірних зусиль.

Реалізуємо клас SiPMSimulator у модулі core/adc_simulator.py. Симулятор генерує сигнал у формі суми кількох незалежних компонент, кожна з яких відповідає одному джерелу подій або шумів у справжньому детекторному тракті. Розглянемо ці компоненти послідовно.

Базовий рівень сигналу формується як константа, що дорівнює значенню зміщення нуля з об’єкта конфігурації. До цієї константи додається гаусівський шум із заданим середньоквадратичним значенням. Цей шум моделює сумарну дію теплового шуму вхідного підсилювача, шуму квантування АЦП і інших джерел широкосмугового стохастичного фону. Постає питання вибору генератора випадкових чисел. Стандартний модуль random бібліотеки Python зорієнтований на скалярні значення і дає погану продуктивність на масивах у мільйони елементів. Класична бібліотека numpy.random з функцією randn

працює швидко, проте використовує застарілий глобальний стан генератора, що ускладнює відтворюваність експериментів. Тому використаємо сучасний генератор numpy.random.default_rng. Він забезпечує і швидкість векторизованих операцій numpy, і можливість локального стану з детермінованим зерном. Завдяки цьому два запуски симулятора з однаковим зерном дають побітово ідентичні потоки сигналу, що принципово для повторюваності тестів алгоритмів.

Розглянемо тепер модель одиничного імпульсу. Як було показано у підрозділі 1.2, форма імпульсу SiPM описується біекспоненційною функцією з швидким переднім фронтом і повільнішим хвостом [3]:

$$s(t) = A \cdot \left(exp\left(-\frac{t}{\tau_{decay}}\right) - exp\left(-\frac{t}{\tau_{rise}}\right) \right) \cdot H(t), \qquad (3.1)$$

де A — амплітуда події, τ_{rise} — стала часу переднього фронту порядку одиниць наносекунд, τ_{decay} — стала часу спадання у типовому випадку 25–50 наносекунд, H(t) — східчаста функція Гевісайда, що обнуляє форму до моменту настання події. Цю форму додаємо до сигналу зі знаком, що залежить від обраної полярності. Для негативної полярності, характерної для SiPM, форма віднімається від базової лінії.

Перейдемо до моделювання моментів приходу подій. Реальний SiPM генерує імпульси з двох незалежних джерел: поглинання зовнішніх фотонів та власних термічних флуктуацій кристала. Обидва процеси є пуассонівськими — це означає, що моменти послідовних подій є випадковими і незалежними один від одного, з експоненційним розподілом інтервалів між ними. Реалізуємо два незалежних пуассонівських процеси, для фотонних подій і для темнового рахунку, кожен зі своєю інтенсивністю. Сумарний потік формується об’єднанням двох процесів з маркуванням кожної події за типом.

Розглянемо квантовану структуру амплітуд фотонних подій. Як було показано у підрозділі 1.2, амплітуда k -фотоелектронної події SiPM описується виразом:

$$A_k = A_{1pe} \cdot k + \delta_k, \tag{3.2}$$

де A_{1pe} — амплітуда однофотоелектронного відгуку у одиницях АЦП, δ_k — гаусівський розкид, що моделює внутрішні флуктуації коефіцієнта підсилення мікропікселя, а кількість одночасно поглинутих фотонів k розподілена за Пуассоном із середнім, що залежить від інтенсивності світлового потоку. Темнові події завжди мають k =1, оскільки ініціюються одним термічно емітованим електроном на мікропіксель [10]. Така квантована структура є принциповою для тестування алгоритмів пошуку імпульсів. Без неї було б неможливо переконатися, що алгоритм правильно розрізняє події одно-, дво- і трифотонного відгуку, і що гістограма амплітуд утворює характерну квантовану структуру одноелектронного спектра.

Окремою компонентою сигналу є мережева наводка на частоті 50 Гц, що додається як гармонічна функція з амплітудою у одиниці кодів АЦП. На перший погляд її моделювання здається непотрібним — це джерело можна було б розглядати як зайвий шум, що псує тестові дані. Проте у реальних умовах така наводка завжди присутня у будь-якому неекранованому тракті лабораторної електроніки, і її моделювання у симуляторі робить тестування алгоритмів обробки ближчим до реальних експериментальних даних. Алгоритм пошуку імпульсів, що добре працює на симульованих даних з наводкою, з більшою ймовірністю буде працювати і на реальному сигналі.

Усі компоненти сигналу об’єднуємо в один потік семплів типу uint16 у методі _build_chunk. Реалізацію цього методу (*див. у Додатку А, фрагмент А.4*). Метод викликається з двох контекстів: методу read_waveform, що повертає об’єкт AcquiredWaveform, і методу read_live_chunk, що використовується для роботи режиму осцилографа.

Розглянемо реалізацію трьох режимів тригера у симуляторі. Тут постає природне питання, чи повторювати у симуляторі логіку реальної плати, чи спростити поведінку. До прикладу, у режимі тригера за рівнем просто повертати фрагмент сигналу з вмонтованим у нього імпульсом у заданій позиції. Спрощення зекономило б код, проте призвело б до розбіжностей у поведінці між симулятором і реальною платою, а саме цього треба уникати, бо ціль симулятора у тому, щоб тестування на ньому давало висновки, що переносяться на реальне обладнання. Тому реалізуємо повноцінну логіку всіх трьох режимів. Програмний тригер запускає захоплення негайно при виклику software_trigger. Тригер за рівнем програмно сканує згенерований потік семплів у пошуку першого перетину сигналом заданого порогу у потрібному напрямку і повертає фрагмент сигналу з налаштованою передісторією і післяісторією відносно цієї точки. Зовнішній тригер реалізує очікування з випадковим інтервалом затримки, що моделює надходження умовного зовнішнього сигналу.

Параметри симулятора винесено у поля класу і можуть бути модифіковані ззовні. Швидкість фотонних подій photon_rate_hz за замовчуванням 50 кГц, швидкість темнового рахунку dark_rate_hz за замовчуванням 100 кГц, постійна часу спадання імпульсу pulse_tau_ns за замовчуванням 30 нс, амплітуда однофотоелектронного відгуку single_pe_amp_codes за замовчуванням 800 кодів, середньоквадратичне значення шуму noise_rms_codes за замовчуванням 40 кодів, амплітуда мережевої наводки mains_amp_codes за замовчуванням 5 кодів. Ці значення підібрано так, щоб згенерований сигнал виглядав якомога ближче до того, що видає реальний детекторний модуль з SiPM при кімнатній температурі.

3.4. БІНАРНИЙ ФОРМАТ ФАЙЛУ .PHADC

Запропонуємо власний бінарний формат файлу для зберігання записаних форм сигналу. Назвемо його .phadc, за початковими літерами назви проекту Photon ADC Studio.

Постає природне питання, для чого взагалі розробляти власний формат, якщо існує безліч готових. Розглянемо альтернативи. Найпростіший варіант — зберігати масив відліків у вигляді сирого бінарного потоку, як це робить штатна утиліта виробника WaveDump [5]. Такий підхід має мінімальну реалізацію і працює з будь-якою мовою програмування одним викликом читання файлу. Проте він має серйозний недолік: відсутність будь-якої супровідної інформації про запис. Дізнатися пізніше, при якій частоті дискретизації, на якій моделі плати, з яким зміщенням нуля був виконаний запис, неможливо. У дослідницькій практиці, де результати накопичуються серіями за різних умов, такі дані критично необхідні для пізнішої інтерпретації, і їх відсутність змушує користувача вести зовнішній журнал, що схильно до помилок.

Інший варіант — застосувати поширені наукові формати загального призначення, такі як HDF5 або ROOT. Вони підтримують структуровані метадані, ієрархічне зберігання, стиснення, версіонування. Проте для нашого випадку, одного запису фрагменту сигналу, використання таких форматів є непропорційно складним. HDF5 потребує встановлення бібліотеки h5py обсягом у кілька десятків мегабайтів, додаткових залежностей у вигляді бібліотеки HDF5 на рівні операційної системи, складнішого коду читання і запису. ROOT — ще важче, бо вимагає окремої повної інсталяції фреймворку CERN ROOT і Python-обв’язок до нього.

Запропонуємо компромісне рішення, а саме, простий бінарний формат з фіксованим самоописовим заголовком, що поєднує переваги обох підходів. З сирого формату беремо мінімалізм і легкість читання. Файл відкривається будь-якою мовою програмування без спеціалізованих бібліотек.

Побудуємо формат за принципом «фіксований заголовок плюс масив відліків». Перші 512 байтів файлу відведемо під заголовок, решту — під послідовний масив шістнадцятирозрядних відліків типу uint16 у форматі little-endian. Загальна структура файлу подана нижче, на Рис. 3.1.

Offset	Length	Content
0	8	Magic "PHADCv01" (ASCII)
8	4	JSON header length (uint32 LE)
12	N	JSON header (UTF-8 text)
12 + N	M	Zero padding до 512 байтів
512	...	Samples (uint16 LE × n_samples)

Рис. 3.1. *Приклад структури реального файлу.*

Розглянемо обґрунтування кожного компонента. Перші 8 байтів — це магічне число, рядок PHADCv01 у ASCII-кодуванні. Його присутність — стандартна практика для бінарних форматів, що дозволяє програмі-зчитувачу відразу відкинути файл як невалідний, якщо ці байти не відповідають очікуваним. Без магічного числа єдиним способом перевірки правильності формату було б розширення файлу, що ненадійно, оскільки, користувач може помилково перейменувати інший, довільний, файл у .phadc.

Наступні 4 байти містять довжину JSON-заголовка у байтах, записану як беззнакове ціле у little-endian. Без такого поля зчитувач не знав би, де закінчується JSON і починається масив даних. Явна довжина дає зчитувачу прямий перехід до даних без сканування.

Далі йде сам JSON-заголовок у кодуванні UTF-8, доповнений нульовими байтами до межі 512 байтів. Постає питання, чому саме JSON, а не C-struct з фіксованим розміром полів. C-struct компактніший і швидший у обробці, проте він жорстко закодовує склад полів, що означає, що додавання нового поля порушує сумісність зі старими файлами. JSON натомість самоописовий. Решта 512-байтного блоку заповнена нулями для забезпечення вирівнювання масиву даних на цю межу. Така фіксована довжина блоку метаданих робить читання даних особливо простим.

Запишемо у заголовок такі поля: версію формату для майбутньої сумісності, кількість збережених відліків, частоту дискретизації у герцах, значення зміщення нуля, рядкове позначення полярності імпульсів, рядкове позначення режиму тригера, числове значення порогу тригера, кількість семплів передісторії, часовий штамп створення файлу у форматі ISO 8601 з часовою зоною UTC, текстову примітку оператора.

Приклад заголовка одного з реальних записів, виконаних у режимі тригера за рівнем на платі DT5761:


```
{
  "sample_rate_hz": 4000000000,
  "dc_offset": 32768,
  "polarity": "negative",
  "trigger_mode": "level",
  "trigger_level": 32091,
  "timestamp_ns": 179448,
  "record_length": 100016,
  "created_utc": "2026-05-15T13:16:56Z",
  "app_version": "1.0.0",
  "user_note": "Level trigger test, thr=32091"
}
```


Масив відліків записуємо без жодної додаткової інкапсуляції — це безперервна послідовність шістнадцятирозрядних значень без розділювачів, заголовків блоків чи контрольних сум. Кожен відлік займає рівно два байти, що

для запису довжиною 100 016 відліків дає розмір файлу 200 032 байти даних плюс 512 байтів заголовка, всього 200 544 байти або приблизно 196 кілобайтів.

Перевіримо тепер, наскільки простий запропонований формат у читанні з зовнішніх інструментів. Засобами numpy файл відкривається одним рядком коду:

```
samples = np.fromfile("level_test_0008.phadc", dtype="<u2", offset=512)
```

Тут параметр dtype="<u2" означає беззнакове ціле розрядністю 2 байти у little-endian, а параметр offset=512 пропускає заголовок.

Функції запису і читання .phadc-файлів реалізовано у модулі core/file_io.py. Запис виконує функція write_waveform, яка приймає масив відліків, параметри запису та текстову примітку, формує JSON-заголовок, доповнює його до 512 байтів і записує разом з масивом у файл. Зворотна функція read_waveform зчитує перші 8 байтів файлу, перевіряє магічне число, читає довжину JSON, розпаковує заголовок з утворенням об’єкта WaveformFileMeta, після чого зчитує масив відліків. Обидві функції мають час виконання, лінійний від об’єму даних.

3.5. ГРАФІЧНИЙ ІНТЕРФЕЙС І ФУНКЦІОНАЛЬНІ РЕЖИМИ

Перейдемо до опису графічної частини комплексу. Першим питанням постає вибір бібліотеки для побудови інтерфейсу. У Python-середовищі існує кілька зрілих рішень для цієї задачі: Tkinter як вбудована стандартна бібліотека, wxPython як обв’язка до фреймворку wxWidgets, PyQt6 і PySide6 як обв’язки до Qt та Kivy як спеціалізована бібліотека для сенсорних інтерфейсів. Для нашого випадку, дослідницького застосунку з інтенсивною графічною візуалізацією сигналу у реальному часі, PyQt6 [19] виявляється найкращим вибором з трьох причин.

Перша причина — наявність повного набору віджетів промислової якості, від кнопок і полів вводу до файлових діалогів і складених композитних

компонентів. Альтернативний Tkinter, хоча і доступний без додаткової установки, виглядає застарілим і не дає достатньо контролю над оформленням.

Друга — природний механізм асинхронної взаємодії з фоновими потоками через сигнально-слотову систему Qt. Збір даних з цифровайзера у режимі довгих записів триває секунди, і якщо виконувати його у головному потоці, графічний інтерфейс заморозиться. І остання причина — інтеграція з бібліотекою pyqtgraph, оптимізованою для побудови графіків у режимі реального часу з оновленням десятки разів за секунду без втрати швидкодії [20]. Без такої бібліотеки реалізація режиму осцилографа була б майже неможливою, оскільки, стандартні графічні засоби Qt погано тримають великі об’єми даних з частими оновленнями.

Реалізуємо головне вікно застосунку у класі MainWindow модуля gui/main_window.py. У верхній частині вікна розташуємо інформаційний рядок з назвою застосунку, версією, індикатором поточного джерела даних, назвою активного каталогу запису і випадаючим списком перемикання бекенду. Нижче розмістимо рядок з чотирма функціональними вкладками.

Реалізуємо кожну вкладку окремим модулем у пакеті gui. Чотири вкладки, які взаємодіють з єдиним об’єктом пристрою ADCDevice і єдиним об’єктом конфігурації запису AcquisitionConfig, що передаються у вкладки при ініціалізації. Завдяки цьому зміни параметрів на одній вкладці автоматично відображаються на іншій без потреби в явній синхронізації коду. До прикладу, рівень тригера, перетягнутий мишею на осцилографі, одразу видно у числовому полі вкладки запису.

Стильове оформлення інтерфейсу задано каскадними правилами Qt у модулі gui/styles.py.

3.5.1. РЕЖИМ ОСЦИЛОГРАФА РЕАЛЬНОГО ЧАСУ

Розглянемо перший з чотирьох функціональних режимів. Він призначений для безперервного перегляду сигналу від пристрою без прив'язки до тригера.

Реалізуємо режим у класі LiveOscilloscopeTab модуля gui/tab_oscilloscope.py. Центральним елементом є графічний віджет pyqtgraph, на якому в реальному часі рисується останній зчитаний фрагмент сигналу. Над графіком розташовуємо елементи керування зчитуванням, вибором масштабу та розміром вікна відображення. Праворуч від графіка виводимо блок індикаторів реального часу: мінімальне, максимальне, середнє, розмах, середньоквадратичне значення сигналу у поточному вікні разом з частотою оновлення кадрів і загальною кількістю зчитаних відліків.

Реалізуємо три режими масштабування за вертикальною віссю. У режимі автоматичного підбору межі графіка адаптуються до поточного діапазону значень. У режимі повного діапазону межі фіксуються у проміжку від нуля до 65535. У ручному режимі межі задаються користувачем безпосередньо, що використовується для деталізованого перегляду форми окремих імпульсів з масштабом, налаштованим під їх амплітуду. Усі три режими корисні у різних сценаріях, і відмова від будь-якого з них зробила б інтерфейс менш гнучким.

Створимо на графіку дві інтерактивні горизонтальні лінії. Перша, сіра штрих-лінія на рівні поточного зміщення нуля, служить орієнтиром для базової лінії. Друга, червона перетягувана лінія порогу тригера, видима у режимі тригера за рівнем. Перетягуючи цю лінію мишею, користувач задає рівень порогу безпосередньо на графіку. Це принципово зручніше за введення числа через клавіатуру. Користувач бачить, де саме поріг проходить відносно сигналу, і відчуває, наскільки далеко він від базової лінії та від типових амплітуд імпульсів. Нове значення автоматично транслюється у об’єкт конфігурації і відображається у відповідному полі режиму запису. Зворотна синхронізація теж реалізована.

Постає питання, як саме виконувати зчитування даних з пристрою для оновлення графіка. Використаємо таймер Qt з фіксованим періодом 50 мілісекунд, що відповідає частоті кадрів близько 20 Гц. При кожному спрацюванні таймера здійснюється виклик методу read_live_chunk поточного

об’єкта ADCDevice, який повертає неперервний фрагмент сигналу заданої довжини. Отриманий масив масштабується у потрібний діапазон і оновлює дані на графіку.

Принципово, що оновлення графіка через метод setData бібліотеки pyqtgraph не перерисовує сцену з нуля, а лише модифікує дані вже існуючого об’єкта-кривої [20]. Тому навіть для довгих векторів у мільйон відліків цей виклик виконується за мілісекунди.

3.5.2. РЕЖИМ ЗАПИСУ З НАЛАШТУВАННЯМ ПАРАМЕТРІВ

Розглянемо другий режим — режим запису з налаштуванням параметрів. На відміну від попереднього, тут зчитування виконується з прив'язкою до тригера у одному з трьох передбачених режимів: програмного, за рівнем сигналу або зовнішнього. Результат запису зберігається у файл .phadc у активному каталозі.

Реалізуємо режим у класі AcquisitionTab модуля gui/tab_acquisition.py. Ліву частину вкладки відведемо під три блоки керування з полями введення параметрів. Блок параметрів запису охоплює довжину запису у семплах, зміщення нуля у одиницях шістнадцятирозрядного простору, кількість семплів передісторії та полярність імпульсів. Блок параметрів тригера дозволяє вибрати один з трьох режимів і у режимі за рівнем задати числове значення порогу. Блок параметрів серійного запису містить кількість послідовних записів за одне натискання, префікс імен файлів, а також, поле для текстової примітки до серії.

З’єднаємо усі поля інтерфейсу з об’єктом конфігурації AcquisitionConfig двостороннім зв'язком. При зміні значення у будь-якому полі викликається обробник _on_any_changed, що оновлює відповідне поле об’єкта конфігурації. На перший погляд завантаження нової конфігурації виглядає тривіальною операцією, просто записати нові значення у віджети. Проте при кожному виклику setValue для віджета викликається сигнал valueChanged, який запускає

обробник, що намагається оновити об’єкт конфігурації зі ще не повністю переписаного стану інтерфейсу. Результатом стало б затирання щойно заданих значень нулями з віджетів, що ще незаповнились. Цей баг було виявлено під час тестування на ранній версії застосунку, нові конфігурації загадково перетворювалися на нулі. Щоб уникнути цього, на час групової зміни усі віджети тимчасово блокуємо через об’єкти QSignalBlocker, що зупиняють виклик сигналів до завершення оновлення.

Натискання кнопки запуску запису ініціює процес у фоновому потоці виконання. Спочатку перевіряється наявність заданого активного каталогу запису. Якщо такий каталог не задано, відкривається діалог з пропозицією перейти у режим керування каталогами і вибрати теку. Винесення тривалої операції збору у окремий потік є необхідним, оскільки, якби запис виконувався у головному потоці, графічний інтерфейс заморозився б на весь час операції, що для тривалих серій з тисячами семплів на запис могло б тривати десятки секунд.

Рушій виконує задану кількість послідовних записів, по завершенню кожного зберігає результат у файл .phadc в активному каталозі з автоматичною нумерацією. Періодично у головний потік надсилається сигнал прогресу, який оновлює відповідний індикатор на вкладці. Праворуч від лівого блоку керування розташовуємо графічний віджет з мініатюрою останньої записаної форми сигналу. Він дозволяє відразу побачити, що саме потрапило у файл. У сценаріях серійного запису мініатюра оновлюється після кожного збереженого файлу.

3.5.3. КЕРУВАННЯ КАТАЛОГАМИ ВИМІРЮВАНЬ

Виокремимо керування каталогами вимірювань у самостійний режим. Це рішення може здатися надлишковим, оскільки, здається, що для запису достатньо стандартного файлового діалогу при кожному натисканні кнопки збереження. Проте у реальній експериментальній практиці така схема виявляється незручною. Одне дослідження зазвичай включає десятки серій записів з різними умовами: рівнями освітлення, напругами на детекторі,

температурою, об’єктами вивчення. Користувач, що натискає кнопку запису десятки разів за годину, не повинен щоразу обирати теку у файловому діалозі, тому що це гальмує роботу і породжує помилки. Натомість запровадимо концепцію «активного каталогу». Користувач один раз обирає теку для поточної серії, після чого всі записи автоматично зберігаються до неї з послідовною нумерацією.

Реалізуємо режим у класі WorkspaceTab модуля gui/tab_workspace.py. Ліву панель відведемо під три кнопки керування каталогами і список нещодавно відкритих каталогів. Центральна панель показує список файлів .phadc у поточному відкритому каталозі. Права панель відображає метадані вибраного файлу. Кнопка Set Recording Folder встановлює поточний каталог як активний — саме у нього виконуватимуться нові записи з режиму збору даних.

Реалізуємо список нещодавно використаних каталогів. Без такої функції, користувач при кожному запуску програми мусив би заново знаходити свою робочу теку через файлову систему, що особливо незручно для теки з глибоким шляхом. Зберігаємо список у файлі workspace.json, який створюється у домашньому каталозі користувача за шляхом ~/.photon_adc_studio/. Файл містить ідентифікатори до десяти останніх каталогів і автоматично оновлюється при відкритті нової теки або встановленні нової теки як активної. Логіку зберігання та читання цього файлу винесено у клас Workspace модуля core/workspace.py.

Відображення метаданих обраного файлу реалізуємо через окрему функцію read_metadata_only модуля core/file_io.py, яка зчитує лише заголовок файлу без читання масиву семплів.

Кнопка Open in Analysis на правій панелі переключає у режим пост-обробки з автоматичним завантаженням обраного файлу.

3.5.4. РЕЖИМ ПОСТ-ОБРОБКИ ЗАПИСАНИХ ФАЙЛІВ

Розглянемо останній з чотирьох режимів — режим пост-обробки записаних файлів. Він реалізує другу основну функцію комплексу після збору даних, а саме, пошук імпульсів у записаних файлах і обчислення їх характеристик. Цей режим є самостійним у тому сенсі, що може працювати з

будь-яким раніше записаним файлом .phadc незалежно від поточного стану інших режимів і незалежно від типу пристрою, яким файл був записаний.

Реалізуємо режим у класі AnalysisTab модуля gui/tab_analysis.py. У верхній лівій частині розташуємо блок вибору файлу з кнопкою відкриття стандартного діалогу. Нижче — блок параметрів детектора імпульсів. Тут задаються режим визначення порогу (адаптивний як кратність оцінки шуму або абсолютний у кодах АЦП), значення коефіцієнта кратності або абсолютного порогу, а також два параметри постфільтрації: мінімальна ширина імпульсу і максимальна відстань між сусідніми імпульсами для їх злиття. Кнопка Run analysis запускає обчислення.

Після завершення обчислень інтерфейс заповнюється результатами. У лівій нижній області виводиться блок зведеної статистики: кількість знайдених імпульсів, оцінка базової лінії, оцінка середньоквадратичного значення шуму, фактично використаний поріг, середнє і медіанне значення амплітуди, середня ширина імпульсу, середнє значення повної ширини на половині максимуму, частота рахунку імпульсів у одиницях підрахунків за секунду з автоматичним вибором масштабу.

Центральну частину займають три графіки. Великий графік у верхній частині показує повну осцилограму запису з відміченими піками всіх знайдених імпульсів. На цьому ж графіку відображаються горизонтальні лінії базової лінії та фактично використаного порогу, що дозволяє візуально оцінити коректність роботи алгоритму. Два менші графіки внизу показують гістограми розподілу

амплітуд знайдених імпульсів і розподілу повної ширини цих імпульсів на половині максимуму. У типовому записі з SiPM перша гістограма дає характерну квантовану структуру з піками на рівнях, кратних амплітуді однофотоелектронного відгуку, друга — близький до експоненційного розподіл ширин з модою в околі тривалості одиничного імпульсу.

Праворуч під графіками виводимо таблицю знайдених імпульсів. Кожен рядок таблиці містить сім полів: порядковий номер імпульсу, індекс початку перевищення порогу, індекс піка максимальної амплітуди, індекс кінця імпульсу, амплітуду у кодах АЦП, ширину у семплах, повну ширину на половині максимуму та площу. Також, реалізуємо можливість автоматичного масштабування на область відповідного імпульсу, що дозволяє швидко переглянути форму будь-якої окремої події з усієї серії і виявити нетипові імпульси, що відрізняються від загальної маси, шляхом натискання на будь-який рядок таблиці.

Експорт результатів виконується кнопкою Export pulses (CSV), що зберігає таблицю імпульсів у CSV-файл поряд з джерельним .phadc-файлом.

3.6. АЛГОРИТМ ПОШУКУ ІМПУЛЬСІВ

Розглянемо алгоритм пошуку імпульсів у записаному фрагменті сигналу. Реалізуємо його у модулі core/signal_processing.py у вигляді функції detect_pulses. На вхід функція приймає масив відліків записаного сигналу разом з конфігурацією детектора, на виході формує список об’єктів з характеристиками кожного знайденого імпульсу і агрегатну статистику. Виклик функції повністю векторизований. У ньому немає жодного явного циклу Python над окремими семплами, всі операції виконуються над цілими масивами через функції бібліотеки numpy [22]. Повну реалізацію функції (*див. у Додатку А, фрагмент А.5*).

3.6.1. ОЦІНКА БАЗОВОЇ ЛІНІЇ І РІВНЯ ШУМУ

Перейдемо до першого етапу обробки — одержання робастних оцінок базової лінії сигналу і середньоквадратичного значення шуму. Ці величини далі служать опорними для визначення порогу детекції. Постає питання, який спосіб оцінки обрати. Найочевидніший варіант — це обчислити середнє арифметичне і стандартне відхилення всіх відліків запису. Цей підхід дав би коректні оцінки,

якби сигнал містив лише шум. Проте у реальному сигналі від детектора практично завжди присутні справжні події, що мають характер викидів відносно базової лінії, причому амплітуда таких викидів може у десятки разів перевищувати рівень шуму. Класичне середнє арифметичне було б систематично зсунуте такими подіями, а класичне стандартне відхилення — суттєво завищене. Поріг, обчислений на таких зсунутих оцінках, виявився б завищеним, і слабкі справжні події алгоритм би пропустив.

Тому використовуємо робастні оцінки, тобто, медіану замість середнього арифметичного і медіану абсолютних відхилень замість стандартного відхилення [14]. Медіана має ту властивість, що навіть якщо половина значень у вибірці є викидами, її оцінка лишається близькою до значення, яке вона мала б за відсутності викидів. Це робить її стійкою до присутності справжніх подій у вибірці для оцінювання базової лінії.

Постає питання вибору ділянки сигналу, на якій виконувати оцінку. Простий варіант, використати весь запис цілком, має недолік, що при високих частотах рахунку імпульсів частка справжніх подій у вибірці може перевищити стійкість медіани і оцінка стане ненадійною. Альтернативний варіант — узяти лише початкове вікно сигналу. Здається малим, але насправді виявляється достатньо великим. Тому беремо вікно перших 4096 відліків з початку запису. На цьому вікні обчислюємо медіану, яку приймаємо за оцінку базової лінії.

У тому ж опорному вікні обчислюємо медіану модулів відхилень кожного відліку від медіани вікна, тобто, статистику MAD. Для нормально розподілених даних значення MAD пов'язане зі стандартним відхиленням множником, що дорівнює оберненій квантильній функції нормального розподілу в точці 0.75 і чисельно дорівнює приблизно 1.4826 [14]:

$$\sigma_{noise} \approx 1.4826 \cdot MAD. \quad (3.3)$$

Такою оцінкою отримуємо середньоквадратичне значення шуму, нечутливе до присутності справжніх подій у вибірці.

Нормалізуємо сигнал за полярністю. Сформуємо масив, у якому всі справжні події завжди дають додатний відгук. Для полярності Negative (як у SiPM) це робиться відніманням сигналу від базової лінії, для полярності Positive — навпаки. Таке нормалізування виконуємо одразу, щоб уся подальша обробка працювала з єдиним кодом без розгалужень за полярністю.

Обчислимо поріг детекції. В адаптивному режимі він дорівнює добутку оцінки шуму на коефіцієнт K, заданий користувачем (за замовчуванням 5.0):

$$\theta = K \cdot \sigma_{noise}. \quad (3.4)$$

У режимі абсолютного порогу, значення задається безпосередньо у кодах АЦП. Постає питання вибору значення K за замовчуванням. У фізиці частинок прийнято п'ятисигмовий критерій, тобто, поріг на рівні п'яти стандартних відхилень над базовою лінією [15]. Цей вибір ґрунтується на тому, що для гаусівського шуму ймовірність випадкового перетину одним відліком п'ятисигмового порогу становить приблизно $2.87 \cdot 10^{-7}$, що для запису довжиною у 100 тисяч відліків очікувано дає менше однієї хибної події за запис. Меншу величину K брати небезпечно, бо кількість хибних спрацювань зростає швидко через "хвости" гаусівського розподілу, а кожне хибне спрацювання спотворює статистику справжніх подій. Більшу — теж не варто, бо тоді алгоритм почне пропускати слабкі однофотонні події, які за амплітудою близькі до п'ятисигмової

межі. Для типового запису з реальної плати з оцінкою шуму 95 кодів і K=5 фактичний поріг склав 474 коди АЦП.

3.6.2. ВИЯВЛЕННЯ ІНТЕРВАЛІВ ПЕРЕТИНУ ПОРОГУ

Знайдемо інтервали, у яких сигнал перевищує поріг. Створимо бінарну маску з елементами True для тих індексів, де нормалізований сигнал перевищує поріг, і False для решти. Через дискретне диференціювання маски обчислимо точки переходів, індекси, у яких маска змінює значення з False на True (початок інтервалу) і з True на False (кінець інтервалу). Кожна пара початок-кінець

утворює один кандидат на імпульс. Уся операція виконується за один прохід по масиву через векторизовані виклики numpy.

Постає питання, чи можна обійтися без додаткових фільтрів і одразу інтерпретувати знайдені інтервали як справжні імпульси. Виявляється, не можна. Простий пошук дає два типи спотворень. Перший тип — реальний імпульс на хвості може мати короткочасне просідання нижче порогу через шум, що алгоритм сприйме як кінець одного імпульсу і початок наступного. У результаті одна фізична подія перетвориться на дві або більше окремих кандидатів. Другий — шум сам по собі може випадково створювати поодинокі викиди, що ледь перевищують поріг на один-два відліки, особливо при K, близькому до п'ятисигмового, де ймовірність таких викидів не зовсім нульова.

Пропустимо список кандидатів через два фільтри для усунення обох типів спотворень. Перший фільтр злиття об’єднує сусідні інтервали у єдиний інтервал, якщо вони розділені меншою кількістю семплів ніж заданою, за замовчуванням 2. Така операція компенсує короткочасні просідання сигналу нижче порогу всередині хвоста одного імпульсу. Другий фільтр відкидає інтервали з довжиною менше ніж min_width, за замовчуванням 3 семпли, як шумові викиди. Справжні фотонні події на платі з частотою дискретизації 4 GS/s завжди мають тривалість щонайменше десяти-двадцяти семплів — це наслідок наносекундних сталих часу

SiPM, а шумові викиди обмежені лише кореляційною довжиною шуму, що зазвичай становить один-два відліки.

3.6.3. ОБЧИСЛЕННЯ ХАРАКТЕРИСТИК ОКРЕМИХ ІМПУЛЬСІВ

Обчислимо чотири основні характеристики кожного імпульсу, що пройшов фільтрацію. Для кожного інтервалу вирізаємо відповідний фрагмент сигналу і виконуємо обчислення над ним.

Амплітуду визначаємо як максимальне значення у фрагменті, що дорівнює піковому перевищенню над базовою лінією у кодах АЦП. Це найочевидніша характеристика події, проте не завжди найкраща для подальшого аналізу, оскільки, амплітуда чутлива до високочастотних флуктуацій шуму на піку імпульсу, і два імпульси з однаковою енергією можуть мати різну амплітуду через випадкові вкладки шуму.

Площу обчислюємо як суму відліків фрагмента, яка в одиницях кодів-семплів пропорційна повному заряду події. У фізичному сенсі площа є найточнішим індикатором кількості фотоелектронів, оскільки інтегрує всю енергію події незалежно від часового джиттера. На відміну від амплітуди, шум на хвостах часткового компенсується усередненням, і два імпульси з однаковою енергією будуть мати близькі площі навіть при відмінностях у формі.

Повна ширина імпульсу на половині максимуму вимагає більш складного обчислення. Найпростіший підхід, підрахувати кількість послідовних семплів, що перевищують рівень половини амплітуди, дає точність плюс-мінус один семпл, що при частоті дискретизації 4 GS/s становить плюс-мінус 250 пікосекунд. Це значно гірше, ніж можна одержати з тих самих даних при акуратнішій обробці. Враховуючи, що сама природа сигналу безперервна, а семпли є лише її дискретизованим поданням, можна оцінити справжнє положення півмаксимуму між двома сусідніми семплами через лінійну інтерполяцію.

Знайдемо точно положення лівого і правого країв уявної кривої, що проходить через сусідні семпли, у точці її перетину з рівнем половини амплітуди. Лівий край — це найлівіший семпл, що перевищує половину амплітуди. Разом з його лівим сусідом утворюють пару точок, через які проводиться пряма. Точка перетину цієї прямої з горизонталлю на рівні половини амплітуди і є шуканим положенням лівого краю. Правий край знаходимо аналогічно. Різниця цих двох положень і є точно інтерпольованим значенням повної ширини на половині максимуму. Завдяки інтерполяції точність визначення FWHM становить порядку 0.1 семпла, тобто близько 25 пікосекунд при частоті дискретизації 4 GS/s, що у десять разів краще, ніж без інтерполяції.

Час наростання за рівнями 10–90 відсотків обчислюємо аналогічно через інтерполяцію. Знаходимо положення на передньому фронті імпульсу, у яких сигнал перетинає рівні 10% і 90% амплітуди, з лінійною інтерполяцією між сусідніми семплами. Різниця цих положень і є шуканою величиною. Ця характеристика є стандартним електронним параметром швидкодії детектора і використовується для класифікації подій за формою. До прикладу, для відокремлення прямих фотонних подій від післяімпульсів, які мають довший передній фронт.

3.6.4. АГРЕГАТНА СТАТИСТИКА І ПОВЕРНЕННЯ РЕЗУЛЬТАТІВ

Сформуємо агрегатні статистики, що описують усю серію в цілому. Обчислимо середнє і медіанне значення амплітуди разом зі стандартним відхиленням, середнє значення ширини, середнє значення повної ширини на половині максимуму. Окремо обчислимо швидкість рахунку імпульсів як відношення їх загальної кількості до тривалості запису у секундах, з автоматичним вибором одиниці виміру. Така адаптивна одиниця виміру виявляється зручною, бо швидкість рахунку SiPM при різних умовах освітлення може відрізнятися на чотири-п'ять порядків, і фіксована одиниця ускладнювала б порівняння результатів.

Усі ці агрегатні величини разом з оцінками базової лінії, шуму, фактично використаним порогом і списком об’єктів окремих імпульсів повертаються з функції detect_pulses у вигляді об’єкта класу PulseDetectionResult. Цей об’єкт далі використовується у режимі пост-обробки для побудови графіків, заповнення таблиці імпульсів і виведення зведеної статистики у блоці результатів інтерфейсу.

4. ТЕСТУВАННЯ І АПРОБАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ

У розділі викладено результати поетапної перевірки розробленого програмного комплексу — від випробувань тракту на еталонному сигналі від лабораторного генератора, через тестування на симуляторі без підключення обладнання та до реєстрації фотонних подій реальним кремнієвим фотопомножувачем з аналізом отриманих характеристик.

4.1. ПЕРЕВІРКА АПАРАТНО-ПРОГРАМНОГО ТРАКТУ ЕТАЛОННИМ СИГНАЛОМ

Перш ніж переходити до реєстрації сигналу від кремнієвого фотопомножувача, доцільно переконатися: що сам апаратно-програмний тракт працює коректно, що плата DT5761 справно оцифровує вхідну напругу, що драйверний шар правильно зчитує дані, що файлові операції зберігають їх без спотворень. Найнадійнішим способом цього є подача на вхід тракту сигналу зі заздалегідь відомою формою і параметрами.

Як джерело еталонного сигналу скористаємось лабораторним генератором стандартних сигналів Г4-42. Це прилад радянського виробництва, що формує синусоїдальний сигнал у широкому діапазоні частот від десятків кілогерц до десятків мегагерц з регульованою амплітудою. Хоча Г4-42 не є сучасним приладом, для нашої задачі sanity-check він цілком придатний. Точність форми синусоїди забезпечується за рахунок ретельно спроєктованих аналогових кіл і не залежить від епохи виробництва. Лабораторну установку зображено на Рис. 4.1.

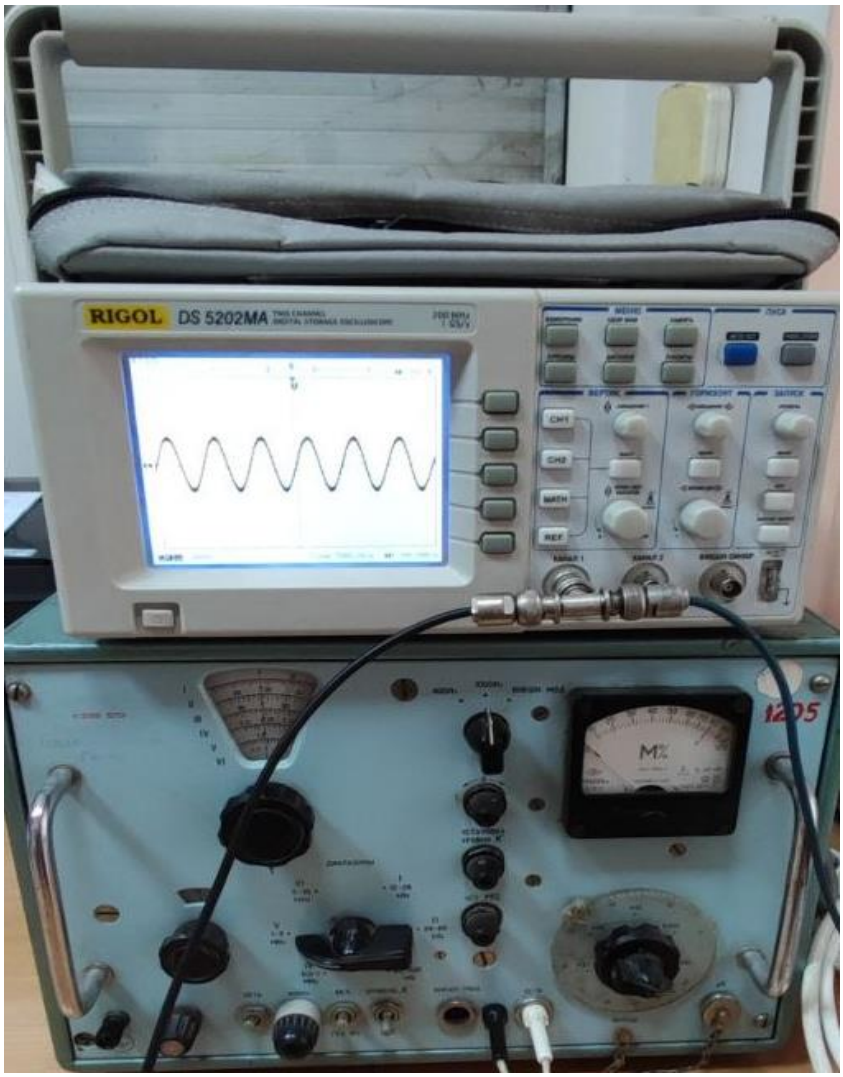

Рис. 4.1. *Лабораторна установка для перевірки апаратно-програмного тракту: лабораторний генератор стандартних сигналів Г4-42 (внизу) як джерело еталонного синусоїдального сигналу, цифровий осцилограф Rigol DS5202MA (вверху) для контрольного перегляду форми сигналу. Сигнал з виходу генератора одночасно подавався на вхід осцилографа Rigol і на вхід каналу CH0 цифровайзера CAEN DT5761, що знаходиться поза кадром.*

Налаштуємо генератор на видачу синусоїди частотою близько 1.6 МГц з амплітудою близько 800 мВ від піка до піка. Така частота вибрана з кількох міркувань: вона достатньо висока, щоб перевірити швидкодію тракту і відсутність обмежень з боку аналогової смуги пропускання, і водночас достатньо низька, щоб форма сигналу залишалась впізнаваною на осцилографі з частотою

дискретизації 1 GS/s. Амплітуду обираємо у межах динамічного діапазону DT5761 без необхідності у додатковому підсиленні або послабленні.

Виведемо сигнал на цифровий осцилограф Rigol DS5202MA для контрольного перегляду форми. Отримана осцилограма наведена на Рис. 4.2.

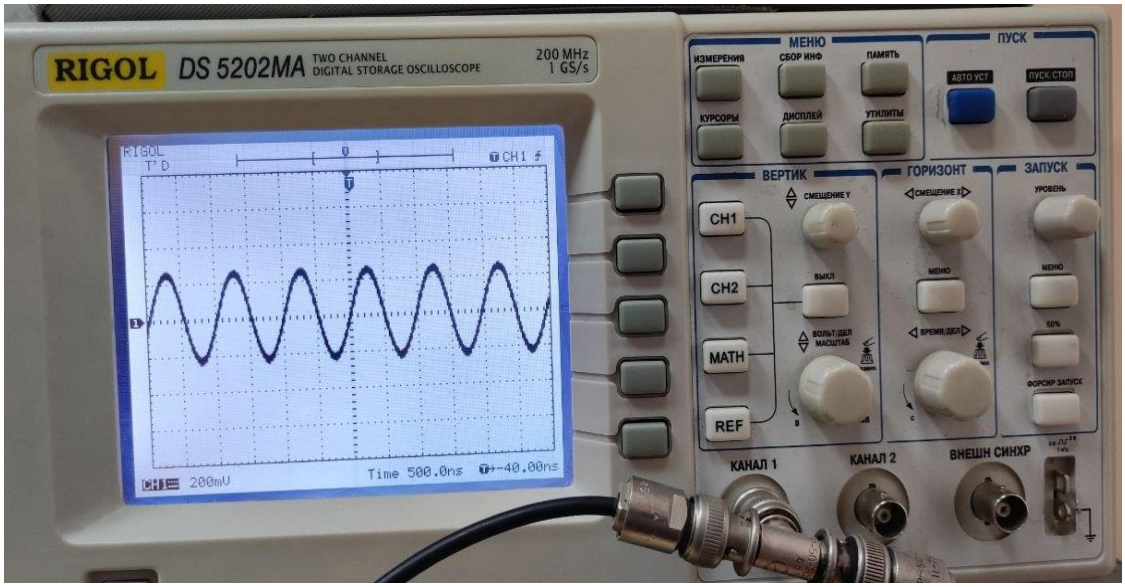

Рис. 4.2. *Осцилограма синусоїдального сигналу з виходу генератора Г4-42, виведена на двоканальний цифровий осцилограф Rigol DS5202MA. Шкала по вертикалі — 200 мВ/поділка, по горизонталі — 500 нс/поділка. На осцилограмі видно близько восьми повних періодів синусоїди тривалістю приблизно 625 нс кожна, що відповідає частоті близько 1.6 МГц. Амплітуда сигналу від піка до піка складає приблизно 800 мВ.*

Таким чином, маємо незалежний еталон форми сигналу від цифрового осцилографа з відомими характеристиками. Той самий сигнал подаємо паралельно на вхід CH0 цифровайзера DT5761. Запустимо застосунок у режимі Hardware, перейдемо у режим осцилографа реального часу і запустимо безперервне зчитування.

Спостережуваний на екрані сигнал відтворює форму синусоїди з тим самим періодом 100 нс і пропорційною амплітудою, перерахованою у одиниці АЦП-кодів. Live-індикатори дають числові підтвердження: різниця між

максимальним і мінімальним значенням за вікно перегляду відповідає очікуваним 800 мВ після перерахунку, період коливань вкладається у вікно зі ста тисяч відліків як приблизно 40 повних періодів, що при частоті дискретизації 4 GS/s і тривалості вікна 25 мкс дає одиничний період близько 625 нс. Виконаємо запис довжиною кілька десятків мікросекунд і завантажимо отриманий файл у режим аналізу — у завантаженому файлі присутні синусоїдальні коливання з тим самим періодом, що і на осцилографі.

Таким чином, проста перевірка з еталонним сигналом підтверджує коректну роботу всього тракту захоплення. Генератор формує синусоїду, плата DT5761 її оцифровує, драйверний шар через ctypes передає дані у Python-код, GUI відображає сигнал у режимі реального часу, файлові операції зберігають дані без втрат. Кожна ланка ланцюга працює як очікувалося, і можна переходити до основних експериментів з кремнієвим фотопомножувачем.

4.2. ТЕСТУВАННЯ У РЕЖИМІ СИМУЛЯЦІЇ

Перейдемо тепер до тестування комплексу у режимі симулятора. Це паралельний шлях верифікації, що дозволяє переконатися у правильній роботі всіх рівнів програми вище за драйверний шар, рушія збору, файлових операцій, графічного інтерфейсу, алгоритму пошуку імпульсів, без залежності від доступності реального обладнання.

Запустимо застосунок, за замовчуванням, в режимі симулятора. Відкриється головне вікно, у заголовку якого індикатор поточного джерела даних показуватиме симулятор. Перейдемо у режим осцилографа і запустимо безперервне зчитування. На графіку буде відображатися сигнал з характерним для SiPM виглядом. Це підтверджує, що симулятор генерує сигнал згідно з закладеною фізичною моделлю, описаною у підрозділі 3.3.

Виконаємо тестовий запис у режимі програмного тригера. Перейдемо у режим запису, встановимо довжину запису 100 тисяч відліків, виберемо

програмний режим тригера і натиснемо кнопку ACQUIRE. Файл зберігається у поточному активному каталозі у форматі .phadc. Якщо активного каталогу немає, застосунок коректно відкриває діалог з пропозицією перейти у режим керування каталогами і вибрати теку.

Перейдемо у режим пост-обробки і завантажимо записаний файл. Алгоритм пошуку імпульсів з параметрами за замовчуванням знаходить кілька десятків подій. У зведеній статистиці відображаються оцінки базової лінії і шуму, фактично використаний поріг, середня і медіанна амплітуди, середнє значення FWHM, швидкість рахунку імпульсів. Гістограма амплітуд має характерну квантовану структуру з піками на кратних рівнях — це закладено у моделі симулятора через рівняння (3.2). Кількість знайдених імпульсів, помножена на середній інтервал між подіями за пуассонівським процесом, узгоджується з заданими у симуляторі швидкостями фотонних і темнових подій.

Перевірка симулятора у всіх трьох режимах тригера показала очікувану поведінку. У режимі програмного тригера запис відбувається миттєво. У режимі тригера за рівнем застосунок чекає, поки сигнал перетне заданий поріг, для типового сигналу симулятора з амплітудою однофотоелектронного відгуку близько 800 кодів поріг на рівні 1000 кодів спрацьовує практично відразу через щільний потік подій. У режимі зовнішнього тригера симулятор моделює очікування з випадковим інтервалом затримки і коректно повертає запис після умовного спрацювання.

Тестування у режимі симуляції дозволило відлагодити переважну більшість логіки застосунку без необхідності у фізичному доступі до обладнання, і саме завдяки цьому етап інтеграції з реальною платою (підрозділ 4.3) виявився відносно швидким, оскільки, основні проблеми були виявлені і вирішені на симуляторі.

4.3. ПІДКЛЮЧЕННЯ РЕАЛЬНОЇ ПЛАТИ DT5761 І ДІАГНОСТИКА ЗАЛЕЖНОСТЕЙ

Перейдемо до інтеграції з реальною платою CAEN DT5761 у лабораторії. Підключимо плату до робочої станції через стандартний кабель USB. Світлодіодна індикація передньої панелі засвідчує наявність живлення і встановлення USB-з’язку. Запустимо застосунок, перейдемо в режим HARWARE.

Перша спроба запуску виявила проблему. Застосунок видав помилку при завантаженні бібліотеки CAENDigitizer.dll. Windows не зміг знайти одну з її залежностей, не вказавши конкретну. Стандартна установка CAEN-драйверів була виконана, бібліотеки лежали у стандартних шляхах, проте Python не зміг їх знайти. Ця проблема є проявом особливостей завантаження динамічних бібліотек на Python 3.8 і новіших версіях, описаних у підрозділі 3.2.

Діагностику виконано через інструмент pefile, Python-бібліотеку для аналізу залежностей PE-бінарників. Аналіз показав, що CAENDigitizer.dll вимагає чотирьох супутніх бібліотек: CAENComm.dll, CAENVMELib.dll, PlxApi.dll і стандартної msvcrXX.dll з пакету Visual C++ Redistributable. Перші три постачаються разом з CAEN-драйверами, а остання є стандартним компонентом Windows і присутня у системі. Проблема полягала у тому, що бібліотеки лежали у різних каталогах стандартної інсталяції CAEN, і Windows-завантажувач у Python 3.8+ не виконує наскрізного пошуку залежностей по всіх таких каталогах.

Рішенням стало розширення логіки пошуку бібліотек у модулі core/adc_caen.py, функція пошуку перевіряє кілька кандидатів каталогів і реєструє знайдені через os.add_dll_directory, як описано у підрозділі 3.2. Альтернативним рішенням є розміщення усіх чотирьох DLL у одному каталозі поруч з виконуваним файлом застосунку. Це і було зроблено для портативності. У каталозі проекту тепер лежать чотири DLL загальним розміром близько 500

КБ, що робить застосунок самодостатнім і не залежним від конкретної інсталяції CAEN на робочій станції.

Після виправлення завантаження бібліотек повторний запуск пройшов успішно. У заголовку вікна застосунку з’явився рядок з ідентифікацією плати — DT5761 @ 4.0 GS/s, отриманий через виклик CAEN_DGTZ_GetInfo. Індикатор поточного джерела даних переключився у режим Hardware. Це підтверджує коректну роботу всього циклу відкриття сесії, від завантаження бібліотек до автодетекції моделі плати з відповідним налаштуванням частоти дискретизації і розрядності АЦП.

Безпосередньо до плати підключимо детекторний модуль з SiPM і блоком живлення Hamamatsu C11204-01, описаний у підрозділі 2.2. Сигнал з виходу модуля поступає на вхід CH0 цифровайзера через коаксіальний кабель з імпедансом 50 Ом. Високовольтне живлення детектора підключимо до C11204-01 згідно з документацією виробника. Перейдемо у режим осцилографа реального часу і запустимо безперервне зчитування для попередньої оцінки сигналу.

На Рис. 4.3. наведено осцилограму у режимі вільного запуску, отриману з реальної плати DT5761 з підключеним SiPM. Видно базову лінію на рівні близько 32660 кодів АЦП, що відповідає налаштованому зміщенню нуля близько середини шкали. На цій базовій лінії помітні короткочасні відхилення вниз — це і є шукані фотонні події SiPM, з характерною негативною полярністю і амплітудою у десятки-сотні кодів АЦП.

Рис. 4.3.. *Осцилограма у режимі вільного запуску (Live Oscilloscope), отримана з реальної плати DT5761 з підключеним модулем SiPM. У режимі Auto-fit вертикальної шкали добре видно структуру сигналу: базова лінія на рівні близько 32660 кодів АЦП, гаусівський шум з RMS близько 100 кодів, поодинокі негативні імпульси різної амплітуди — справжні фотонні події. Загальна кількість зчитаних відліків за час перегляду — близько 100 тисяч, частота оновлення кадрів — близько 7 Гц через відносно велике вікно перегляду (також 100 тисяч відліків). На правій панелі індикаторів виведено фактичні оцінки сигналу: min 30784, max 33280, mean 32662.1, peak-to-peak 2496, noise RMS 98.54.*

Live-індикатори справа на Рис. 4.4 дають оцінку базової лінії на рівні 32680 кодів АЦП і середньоквадратичне значення шуму на рівні близько 103 кодів. Ці значення приймемо за вихідні параметри для подальших налаштувань — рівень тригера для режиму запису за рівнем варто виставляти на кілька сигм нижче за базову лінію, тобто десь у проміжку 32000–32400 кодів.

4.4. РЕЄСТРАЦІЯ СИГНАЛУ У ТРЬОХ РЕЖИМАХ ТРИГЕРА

Виконаємо тестування трьох режимів тригера, передбачених у комплексі — програмного, за рівнем сигналу і зовнішнього. Для кожного режиму

виконаємо одиничний запис довжиною 100 тисяч відліків з налаштованою на негативну полярністю імпульсів і зміщенням нуля 32768.

4.4.1. ПРОГРАМНИЙ ТРИГЕР

У режимі програмного тригера запис починається миттєво після натискання кнопки ACQUIRE. Це найпростіший режим, що використовується для діагностики тракту і для безперервної фонової реєстрації, коли важливе не настання конкретної події, а статистика сигналу за деякий час.

Запустимо запис з префіксом імен файлів sw_test і відповідною приміткою. Результат збереження показано на Рис. 4.4.

Рис. 4.4. *Запис, виконаний у режимі програмного тригера на платі DT5761 з підключеним SiPM. Параметри запису: довжина 100 016 відліків, тривалість 25 мкс при частоті 4 GS/s, зміщення нуля 32768, передісторія 5000 відліків. У правій частині інтерфейсу відображається повна осцилограма запису — на тлі базової лінії з шумом видно регулярну послідовність негативних імпульсів різної амплітуди. Внизу під осцилограмою — числові підсумки: 100 016 відліків, тривалість 25.0 мкс, мінімальне і максимальне значення 31232 і 33280 кодів АЦП, тип тригера software. Файл збережено як sw_test_0003.phadc.*

У осцилограмі чітко видно структуру реального сигналу SiPM. Базова лінія тримається на рівні близько 32700 кодів. На ній рівномірно у часі розкидані негативні імпульси з амплітудами у діапазоні від 200 до 1500 кодів — це події з різною кількістю одночасно поглинутих фотоелектронів. Частота появи імпульсів за результатами наступного аналізу складає кілька сотень кілогерц, що відповідає очікуваному рівню темнового рахунку SiPM при кімнатній температурі.

4.4.2. ТРИГЕР ЗА РІВНЕМ СИГНАЛУ

У режимі тригера за рівнем плата самотригерується при перетині сигналом заданого порогу у заданому напрямку. Це основний робочий режим для фотонної реєстрації, який гарантує, що кожна подія, амплітуда якої перевищує заданий рівень, потрапить у запис разом з контрольованою передісторією до моменту тригера.

З урахуванням оцінки базової лінії 32680 і шуму RMS близько 100 кодів, виставимо рівень тригера на 32091 коду — це приблизно на 6σ нижче за базову лінію. При такому рівні випадкові шумові перетини практично виключені, а більшість справжніх імпульсів від однофотоелектронних подій і вище будуть надійно зареєстровані. Результат запису у цьому режимі — на Рис. 4.5.

Рис. 4.5. *Запис, виконаний у режимі тригера за рівнем сигналу на платі DT5761 з підключеним SiPM. Параметри запису ті самі, що і у попередньому випадку, але режим тригера — Level, рівень 32091 коду АЦП. Видно, що відразу на початку запису (через 1 мкс) присутній характерний негативний імпульс — це і є подія, що спрацювала тригер. Передісторія у 5000 відліків забезпечує видимість базової лінії до моменту події. Далі вздовж запису рівномірно розкидані інші імпульси різної амплітуди, що з'явилися у вікні запису вже після основного спрацювання тригера. Файл збережено як level_test_0008.phadc.*

Видимий характерний імпульс на початку запису, з найбільшою амплітудою серед усіх подій у вікні, є саме тим, що ініціював тригер. Інші імпульси, що з'являються пізніше, мають менші амплітуди, вони були б недостатніми для самотригерування і потрапили у запис лише завдяки тому, що першого імпульсу вистачило для запуску.

4.4.3. ЗОВНІШНІЙ ТРИГЕР

Третій з підтримуваних комплексом режимів — це зовнішній тригер, у якому запис ініціюється TTL-імпульсом на вході TRG-IN цифровайзера. Програмна сторона цього режиму описана у підрозділі 3.2 — її роботу

перевірено у режимі симуляції, де симулятор моделює надходження зовнішнього сигналу з випадковим інтервалом затримки (підрозділ 4.2).

Окрема апаратна верифікація цього режиму на реальній платі у рамках даної роботи не виконувалася, оскільки сценарій його використання передбачає наявність у лабораторії зовнішнього джерела синхронізаційних TTL-імпульсів, погоджених із досліджуваним фотонним потоком — імпульсного лазера, світлодіодного генератора, керованого секвенсором, або іншого детектора у схемі збігів. У наявних умовах лабораторії такого джерела не було, а тестування зовнішнього тригера від довільного генератора, не пов'язаного з джерелом світла, не дає більше корисної інформації за ту, що вже отримана у режимі симуляції.

Таким чином, для цього режиму на рівні апаратно-програмного тракту перевірено правильність виклику відповідних функцій CAEN-бібліотеки (CAEN_DGTZ_SetExtTriggerInputMode та явне вимкнення двох інших каналів тригера згідно з підрозділом 3.2), а сама поведінка плати при надходженні TTL-сигналу залишається предметом подальшої апробації у відповідно оснащеному експерименті.

Програмний тригер і тригер за рівнем сигналу перевірено як на симуляторі, так і на реальній платі з ідентичною поведінкою для користувача. Зовнішній тригер на цьому етапі апробовано на симуляторі і на рівні драйверного шару. Його повноцінна апаратна перевірка із синхронізованим джерелом світла залишається предметом подальшої роботи.

4.5. ПОСТ-ОБРОБКА ЗАПИСАНИХ ФАЙЛІВ І SINGLE-PHOTOELECTRON SPECTRUM

Перейдемо до пост-обробки збережених у попередньому підрозділі файлів. Скористаємось одним з записів у режимі програмного тригера, файл sw_test_0003.phadc, як характерним прикладом, з типовою для SiPM

статистикою імпульсів. Завантажимо файл у режим аналізу і запустимо алгоритм пошуку імпульсів з параметрами за замовчуванням: адаптивний поріг K=5, мінімальна ширина 3 семпли, проміжок злиття 2 семпли. Результат показано на Рис. 4.6.

Рис. 4.6. *Результат аналізу записаного файлу sw_test_0003.phadc у режимі пост-обробки. На верхньому графіку показано повну осцилограму запису з відміченими помаранчевими маркерами піками знайдених імпульсів — алгоритм виявив 6 подій у вікні. Знизу зліва — блок зведеної статистики: Pulses found 6, Baseline 32640.0, Noise RMS 94.89, Threshold 474.4, Mean amplitude 746.7, Median amplitude 608.0, Std amplitude 306.2, Mean width 34.8 семплів (8.7 нс), Mean FWHM 31.2 семплів (7.8 нс), Pulse rate 239.96 kcps. Внизу по центру і праворуч — гістограми розподілу амплітуд та ширини кожного зі знайдених імпульсів. У правій частині — таблиця з характеристиками всіх 6 подій: start, peak, amplitude, width, FWHM, area.*

Розглянемо одержані результати детальніше. Оцінка базової лінії склала 32640 кодів АЦП — це дуже близько до значення, отриманого у режимі осцилографа (32680), і узгоджується з налаштованим зміщенням нуля 32768 з невеликою корекцією на постійну складову сигналу від детектора. Оцінка

середньоквадратичного значення шуму через MAD-статистику склала 94.89 коду — це характеристика всього вхідного тракту, включно з шумом самого АЦП, теплового шуму вхідного підсилювача і паразитних наводок. Фактичний поріг детекції при K=5 склав 474.4 коду, що відповідає п'ятисигмовому критерію над оцінкою шуму.

Алгоритм знайшов 6 імпульсів у 25-мікросекундному вікні запису, що при перерахунку дає швидкість рахунку 239.96 тисяч подій за секунду. Це значення є оцінкою темнового рахунку детектора при кімнатній температурі. У нашому експерименті світловий потік на детектор не подавався, тому всі зареєстровані події походять з власних термічних флуктуацій SiPM. Така швидкість темнового рахунку є типовою для SiPM Hamamatsu з активною площею порядку квадратного міліметра при кімнатній температурі і узгоджується з паспортними характеристиками детектора цього класу [2].

Середня амплітуда подій склала 746.7 коду, медіанна — 608.0 коду, при стандартному відхиленні 306.2 коду. Помітна різниця між середнім і медіанним значеннями свідчить про присутність кількох подій з підвищеними амплітудами — це події з 2-фотоелектронним і 3-фотоелектронним відгуком, амплітуди яких приблизно у два і три рази перевищують амплітуду однофотоелектронної події. Таку структуру амплітудного спектра прийнято називати single-photoelectron spectrum, або скорочено SPS [3]. У нашому випадку через малу кількість зареєстрованих подій повноцінного спектра з чіткими піками не сформувалося, проте розсіювання амплітуд у вузький діапазон 512-1408 кодів і характерне переважання менших амплітуд над більшими є очікуваним для SPS.

Середнє значення повної ширини імпульсу на половині максимуму склало 31.2 семпла, що при частоті дискретизації 4 GS/s відповідає 7.8 наносекундам. Це значення цілком узгоджується з паспортними характеристиками одиничних імпульсів SiPM, які мають типову тривалість на півмаксимумі у проміжку 5-20 наносекунд залежно від моделі і налаштувань кола гасіння [3]. Завдяки лінійній

інтерполяції FWHM, описаній у підрозділі 3.6.3, оцінка отримана з точністю порядку 25 пікосекунд.

Для подальшої ілюстрації роботи алгоритму скористаємось інтерактивною можливістю таблиці імпульсів — клік на конкретний рядок зумить осцилограму на відповідну подію. Виберемо одну з імпульсів у середній частині запису. Результат показано на Рис. 4.7.

Рис. 4.7. *Деталь одного імпульсу з запису sw_test_0003.phadc, отримана через клік на відповідний рядок у таблиці імпульсів. На горизонтальній осі — час у мікросекундах, видно ділянку близько 0.25 мкс. Чітко видно характерну форму одиничного імпульсу SiPM — швидкий спад на передньому фронті (тривалістю кілька наносекунд), піковий мінімум на рівні близько 31900 кодів АЦП, повільне експоненційне повернення до базової лінії з тривалістю спадання близько 30-50 наносекунд. Помаранчевий маркер відмічає пік імпульсу, виявлений алгоритмом. Видимий другий невеликий імпульс безпосередньо за основним є афтерпульсом — характерним для SiPM ефектом, описаним у підрозділі 1.2.*

Зум показує характерну форму одиничного імпульсу від SiPM з усіма очікуваними особливостями — швидким переднім фронтом, чітким мінімумом,

експоненційним поверненням до базової лінії. Це фінальне підтвердження того, що весь тракт реєстрації, від детектора до пост-обробки, відтворює фізику процесу без спотворень. Безпосередньо за основним імпульсом видно меншу подію — це афтерпульс, що виникає внаслідок затриманого вивільнення носіїв заряду з пасток у напівпровіднику. Така подія неминуче супроводжує реєстрацію SiPM і присутність її у наших записах є додатковим аргументом на користь того, що ми спостерігаємо саме реальну фізику детектора, а не якісь артефакти тракту.

Експорт результатів виконаємо через кнопку Export pulses (CSV). У результаті у тому ж каталозі з'являється файл sw_test_0003.pulses.csv, у якому таблиця імпульсів збережена у форматі, що читається стандартними інструментами обробки. Перший рядок файлу містить заголовки колонок, наступні шість — характеристики всіх знайдених подій з перерахунком семплових координат у наносекунди. Такий CSV-файл можна відкрити у Excel, Origin, Matlab або власних Python-скриптах для подальшого аналізу, побудови розширених статистик, порівняння кількох записів, формування звітів.

4.6. ВИСНОВКИ ДО РОЗДІЛУ

У розділі викладено результати трирівневого тестування розробленого програмного комплексу, від перевірки тракту на еталонному сигналі через ізольоване тестування у режимі симуляції до повної інтеграції з реальним обладнанням і реєстрації фотонних подій.

Перевірка апаратно-програмного тракту на еталонному синусоїдальному сигналі від лабораторного генератора Г4-42 з паралельним контролем форми незалежним цифровим осцилографом Rigol DS5202MA підтвердила коректну роботу всього ланцюга «генератор — плата DT5761 — драйверний шар — застосунок» без спотворень форми сигналу.

Тестування у режимі симуляції без підключення обладнання показало правильну роботу всіх рівнів програмного комплексу вище за драйверний шар

— рушія збору, файлових операцій, графічного інтерфейсу, алгоритму пошуку імпульсів. Усі три режими тригера повноцінно реалізовані у симуляторі і ведуть себе ідентично до реальної плати, що дозволило виконати більшість роботи над застосунком без фізичного доступу до обладнання.

Інтеграція з реальною платою DT5761 виявила проблему завантаження динамічних бібліотек CAEN, спричинену змінами у механізмі пошуку залежностей DLL у Python 3.8+. Проблему діагностовано і вирішено через власну логіку пошуку у драйверному модулі застосунку.

Реєстрація сигналу від кремнієвого фотопомножувача через CAEN DT5761 у режимах програмного тригера і тригера за рівнем сигналу підтвердила коректне отримання форм одиничних імпульсів детектора з характерною негативною полярністю, наносекундною тривалістю переднього фронту і експоненційним спадом хвоста. Зовнішній тригер на цьому етапі залишився перевіреним лише у режимі симуляції та на рівні драйверного шару, його повноцінна апаратна апробація потребує синхронізованого з фотонним потоком джерела TTL-імпульсів і виноситься за межі даної роботи.

Пост-обробка записаних файлів дала кількісні характеристики сигналу: оцінку базової лінії 32640 кодів, оцінку шуму через MAD-статистику 94.89 коду, поріг детекції на рівні п'ять сигм 474.4 коду, середнє значення повної ширини імпульсів на половині максимуму 7.8 наносекунд, частоту рахунку темнових подій 239.96 тисяч за секунду. Усі значення узгоджуються з паспортними характеристиками детектора і підтверджують коректність роботи всього розробленого комплексу.

ВИСНОВКИ

Дана робота присвячена розробці спеціалізованого програмного забезпечення для реєстрації оптичних сигналів з фотонною роздільною здатністю за допомогою високошвидкісного аналого-цифрового перетворювача CAEN DT5761 у зв’язці з кремнієвим фотопомножувачем як джерелом сигналу. Проаналізовано фізичні засади фотонно-лічильної реєстрації, виявлено обмеження готових програмних рішень виробника АЦП у частині гнучкості налаштування і відсутності засобів пост-обробки.

Спроєктовано і повністю реалізовано модульний програмний комплекс мовою Python з графічним інтерфейсом на основі бібліотеки PyQt6, що покриває повний цикл лабораторної роботи з фотонним детектором. Архітектуру побудовано навколо абстракції пристрою-джерела даних, що дозволяє безперешкодно перемикатися між реальним обладнанням і програмним симулятором. Розроблений комплекс протестовано на трьох рівнях — на еталонному сигналі генератора Г4-42, на власному симуляторі та на реальній платі CAEN DT5761 у лабораторії кафедри медичної радіофізики. Отримано записи сигналу від кремнієвого фотопомножувача з характеристиками, що узгоджуються з паспортними даними детектора.

Основні результати магістерської роботи:

1. Спроєктовано модульну архітектуру програмного комплексу на основі абстракції пристрою-джерела даних ADCDevice, що формалізує взаємодію з обладнанням мінімальним набором з дев'яти методів. Завдяки цій абстракції весь застосунок працює з єдиним інтерфейсом і не залежить від конкретного типу пристрою.
2. Реалізовано драйверний шар для роботи з платою CAEN DT5761 через стандартний модуль ctypes мови Python з власною логікою завантаження динамічних бібліотек CAEN, автоматичним розпізнаванням моделі плати та програмною підтримкою трьох режимів формування тригера (програмного,

за рівнем сигналу та зовнішнього), з яких апаратно на реальній платі протестовано перші два.

3. Розроблено програмний симулятор сигналу кремнієвого фотопомножувача з біекспоненційною формою імпульсу (3.1), пуассонівськими моментами приходу подій, квантованими амплітудами одно-, дво- і трифотоелектронного відгуку (3.2), темновим рахунком і мережевою наводкою 50 Гц.
4. Запропоновано бінарний формат файлу .phadc з 512-байтним самоописовим JSON-заголовком і послідовним масивом шістнадцятирозрядних відліків, що відкривається стандартними засобами numpy одним рядком коду.
5. Реалізовано графічний інтерфейс комплексу на основі PyQt6 з чотирма функціональними режимами — осцилограф реального часу, конфігурація і виконання записів, керування каталогами вимірювань, пост-обробка записаних файлів.
6. Розроблено алгоритм пошуку імпульсів на робастних оцінках базової лінії через медіану і шуму через MAD-статистику (3.3), з адаптивним порогом за п’ятисигмовим критерієм (3.4) і лінійною інтерполяцією при визначенні FWHM з точністю порядку 25 пікосекунд.
7. Виконано трирівневе тестування комплексу. Реєстрація сигналу від SiPM на реальній платі дала числові характеристики, що узгоджуються з паспортними даними детектора — оцінку базової лінії 32640 кодів АЦП, оцінку шуму 94.89 коду, поріг детекції 474.4 коду, середнє значення FWHM імпульсів 7.8 наносекунд, частоту рахунку темнових подій 239.96 тисяч за секунду.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. O'Neill S. Counting photons could redefine the future of CT imaging. Physics World. 2024. URL: https://physicsworld.com/a/counting-photons-could-redefine-the-future-of-ct-imaging/
2. Multi-Pixel Photon Counter (MPPC). Technical Note. Hamamatsu Photonics K.K., 2021. 44 p. URL: https://www.hamamatsu.com/
3. Acerbi F., Gundacker S. Understanding and simulating SiPMs. Nuclear Instruments and Methods in Physics Research Section A. 2019. Vol. 926. P. 16–35. DOI: 10.1016/j.nima.2018.11.118. URL: https://www.sciencedirect.com/science/article/pii/S0168900218317704
4. DT5761 – 10 bit 4 GS/s Digitizer User Manual. Rev. 1. CAEN S.p.A., 2021. 78 p. URL: https://www.caen.it/products/dt5761/
5. UM2091 – WaveDump User Manual. Rev. 15. CAEN S.p.A., 2020. 52 p. URL: https://www.caen.it/products/wavedump/
6. High-Voltage Power Supply C11204-01. Technical Data Sheet. Hamamatsu Photonics K.K., 2020. URL: https://www.hamamatsu.com/eu/en/product/optical-sensors/mppc/mppc_mppc-array/C11204-01.html
7. Willemink M. J., Persson M., Pourmorteza A., Pelc N. J., Fleischmann D. Photon-counting CT: Technical Principles and Clinical Prospects. Radiology. 2018. Vol. 289, No. 2. P. 293–312. URL: https://doi.org/10.1148/radiol.2018172656
8. Flohr T., Petersilka M., Henning A., Ulzheimer S., Ferda J., Schmidt B. Photon-counting CT review. Physica Medica. 2020. Vol. 79. P. 126–136. URL: https://doi.org/10.1016/j.ejmp.2020.10.030
9. Renker D., Lorenz E. Advances in solid state photon detectors. Journal of Instrumentation. 2009. Vol. 4. P04004. URL: https://iopscience.iop.org/article/10.1088/1748-0221/4/04/P04004
10. Klanner R. Characterisation of SiPMs. Nuclear Instruments and Methods in Physics Research Section A. 2019. Vol. 926. P. 36–56. URL: https://www.sciencedirect.com/science/article/pii/S0168900218317716

11. UM1935 – CAENDigitizer Library User Manual. Rev. 22. CAEN S.p.A., 2023. 110 p. URL: https://www.caen.it/products/caendigitizer-library/
12. WP2081 – Digital Pulse Processing in Nuclear Physics. Rev. 4. CAEN S.p.A., 2022. 34 p. URL: https://www.caen.it/products/dpp/
13. Jordanov V. T., Knoll G. F. Digital synthesis of pulse shapes in real time for high resolution radiation spectroscopy. Nuclear Instruments and Methods in Physics Research Section A. 1994. Vol. 345, No. 2. P. 337–345. URL: https://doi.org/10.1016/0168-9002(94)91011-1
14. Rousseeuw P. J., Croux C. Alternatives to the median absolute deviation. Journal of the American Statistical Association. 1993. Vol. 88, No. 424. P. 1273–1283. DOI: 10.1080/01621459.1993.10476408. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1993.10476408
15. Nakhostin M. Signal Processing for Radiation Detectors. Hoboken, NJ: John Wiley & Sons, 2018. 336 p.
16. GD2783 – First Installation Guide to Desktop Digitizers & MCA. Rev. 3. CAEN S.p.A., 2018. 28 p. URL: https://www.caen.it/products/desktop-digitizers/
17. Python 3 Documentation. Python Software Foundation. URL: https://docs.python.org/3/
18. ctypes — A foreign function library for Python. Python Software Foundation. URL: https://docs.python.org/3/library/ctypes.html
19. PyQt6 Reference Documentation. Riverbank Computing. URL: https://www.riverbankcomputing.com/static/Docs/PyQt6/
20. pyqtgraph — Scientific Graphics and GUI Library for Python. URL: https://www.pyqtgraph.org/
21. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1994. 395 p.
22. Harris C. R., Millman K. J., van der Walt S. J. et al. Array programming with NumPy. Nature. 2020. Vol. 585. P. 357–362. URL: https://doi.org/10.1038/s41586-020-2649-2

ДОДАТОК А

ФРАГМЕНТИ ПРОГРАМНОГО КОДУ

А.1. Абстрактний клас ADCDevice

Модуль core/adc_base.py. Абстракція пристрою-джерела даних, від якої успадковуються конкретні реалізації — симулятор сигналу SiPM та драйвер реальної плати CAEN.

```
from abc import ABC, abstractmethod
from dataclasses import dataclass
from typing import Optional
import numpy as np
from .config import AcquisitionConfig
@dataclass
class AcquiredWaveform:
samples: np.ndarray          # uint16 масив відліків
sample_rate_hz: int          # частота дискретизації
dc_offset: int               # зміщення нуля
polarity: str                # полярність імпульсів
trigger_mode: str            # режим тригера
trigger_level: int           # рівень тригера
timestamp_ns: int            # системний час реєстрації
record_length: int           # довжина запису у семплах
class ADCDevice(ABC):
"""Абстракція пристрою-джерела цифрових відліків."""


@abstractmethod
```

```
def open(self) -> None:
    """Відкрити сесію зв'язку з пристроєм."""
@abstractmethod
def close(self) -> None:
    """Закрити сесію зв'язку з пристроєм."""
@abstractmethod
def is_open(self) -> bool:
    """Перевірити стан сесії зв'язку з пристроєм."""
@abstractmethod
def configure(self, config: AcquisitionConfig) -> None:
    """Застосувати параметри запису."""
@abstractmethod
def arm(self) -> None:
    """Перевести пристрій у стан очікування тригера."""
@abstractmethod
def software_trigger(self) -> None:
    """Ініціювати запис програмно."""
@abstractmethod
def read_waveform(self, timeout_s: float = 5.0) -> Optional[AcquiredWaveform]:
    """Зчитати одну зареєстровану форму сигналу."""
@abstractmethod
def read_live_chunk(self, n_samples: int) -> np.ndarray:
    """Зчитати неперервний фрагмент сигналу без прив'язки до тригера."""
@abstractmethod
def info(self) -> dict:
    """Повернути словник з ідентифікаційною інформацією про пристрій."""
```

А.2. Завантаження CAEN-бібліотек з пошуком залежностей

Модуль core/adc_caen.py. Функція реалізує власну логіку пошуку CAEN-бібліотек у трьох кандидатних каталогах з реєстрацією знайденого через os.add_dll_directory для коректної роботи на Python версії 3.8 і новіших.

```
import os
import ctypes
from pathlib import Path

def _load_caen_library() -> ctypes.WinDLL:
    """Знайти і завантажити CAENDigitizer.dll з реєстрацією каталогу
    пошуку залежностей через os.add_dll_directory."""
    project_root = Path(__file__).resolve().parent.parent
    candidates = [
        project_root,
        Path(r"C:\Program Files\CAEN\Digitizers\library\bin"),
        Path(r"C:\Program Files\CAEN\Comm\lib\x86_64"),
        Path.cwd(),
    ]
    for path in candidates:
        if (path / "CAENDigitizer.dll").exists():
            os.add_dll_directory(str(path))
            return ctypes.WinDLL(str(path / "CAENDigitizer.dll"))
    raise FileNotFoundError(
        "CAENDigitizer.dll not found. Install CAEN drivers "
        "or place the DLL next to the application executable."
    )
```

А.3. Налаштування режимів тригера

Модуль core/adc_caen.py, метод класу CAENDigitizer. Реалізує перемикання між трьома режимами тригера через комбінацію CAEN-функцій з явним вимиканням неактивних каналів тригера для уникнення спрацювань одночасно з кількох джерел.

```
def _apply_trigger_mode(self, mode: TriggerMode) -> None:
    """Налаштувати один з трьох режимів тригера, явно вимикаючи інші."""
    DISABLED = 0
    ACQ_ONLY = 1   # CAEN_DGTZ_TRGMODE_ACQ_ONLY
    CHANNEL_MASK = 1  # маска першого і єдиного каналу

    if mode == TriggerMode.SOFTWARE:
        self._check(self.lib.CAEN_DGTZ_SetSWTriggerMode(
            self.handle, ACQ_ONLY))
        self._check(self.lib.CAEN_DGTZ_SetChannelSelfTrigger(
            self.handle, DISABLED, CHANNEL_MASK))
        self._check(self.lib.CAEN_DGTZ_SetExtTriggerInputMode(
            self.handle, DISABLED))

    elif mode == TriggerMode.LEVEL:
        self._check(self.lib.CAEN_DGTZ_SetSWTriggerMode(
            self.handle, DISABLED))
        self._check(self.lib.CAEN_DGTZ_SetChannelSelfTrigger(
            self.handle, ACQ_ONLY, CHANNEL_MASK))
        self._check(self.lib.CAEN_DGTZ_SetExtTriggerInputMode(
            self.handle, DISABLED))
```

```
    elif mode == TriggerMode.EXTERNAL:
        self._check(self.lib.CAEN_DGTZ_SetSWTriggerMode(
            self.handle, DISABLED))
        self._check(self.lib.CAEN_DGTZ_SetChannelSelfTrigger(
            self.handle, DISABLED, CHANNEL_MASK))
        self._check(self.lib.CAEN_DGTZ_SetExtTriggerInputMode(
            self.handle, ACQ_ONLY))
```

А.4. Генерація фрагменту сигналу симулятора SiPM

Модуль core/adc_simulator.py, метод класу SiPMSimulator. Формує неперервний фрагмент сигналу заданої довжини, поєднуючи базову лінію з гаусівським шумом, мережеву наводку 50 Гц, фотонні події з пуассонівськими моментами приходу та квантованими амплітудами, темнові події.

```
def _build_chunk(self, n_samples: int) -> np.ndarray:
    """Згенерувати фрагмент сигналу заданої довжини."""
    baseline = self.config.dc_offset

    # Базова лінія з гаусівським шумом
    samples = np.full(n_samples, baseline, dtype=np.float64)
    samples += self.rng.normal(0.0, self.noise_rms_codes, n_samples)

    # Мережева наводка 50 Гц
    t = np.arange(n_samples) / self.sample_rate_hz
    samples += self.mains_amp_codes * np.sin(2.0 * np.pi * 50.0 * t)

    # Фотонні події з пуассонівськими моментами приходу
    duration_s = n_samples / self.sample_rate_hz
    event_times = self._generate_poisson_events(
```

```
        duration_s, self.photon_rate_hz, is_dark=False)
    for t_event, n_pe in event_times:
        amplitude = self._sample_quantized_amplitude(n_pe)
        self._add_biexponential_pulse(samples, t_event, amplitude)

    # Темнові події (одно-фотоелектронні)
    dark_times = self._generate_poisson_events(
        duration_s, self.dark_rate_hz, is_dark=True)
    for t_event, _ in dark_times:
        amplitude = self._sample_quantized_amplitude(1)
        self._add_biexponential_pulse(samples, t_event, amplitude)

    # Інверсія для негативної полярності
    if self.config.polarity == Polarity.NEGATIVE:
        samples = 2 * baseline - samples

    # Обмеження діапазоном 16-bit ADC
    np.clip(samples, 0, 65535, out=samples)
    return samples.astype(np.uint16)
```

А.5. Обчислення повної ширини імпульсу на половині максимуму з лінійною інтерполяцією

Модуль core/signal_processing.py. Функція для обчислення FWHM окремого імпульсу з точністю кращою за один семпл через лінійну інтерполяцію між сусідніми відліками на лівому і правому краях інтервалу перевищення половинного рівня. Половинне значення передається аргументом, що дозволяє в коді обробки імпульсів обчислювати його однократно на запис, а не повторно для кожного імпульсу.

```
def _fwhm(segment: np.ndarray, half: float) -> float:

"""Обчислити FWHM фрагменту сигналу з лінійною інтерполяцією."""

if segment.size < 2 or half <= 0:
    return float(segment.size)
above = segment >= half

if not above.any():
    return 0.0
idx = np.flatnonzero(above)
left, right = idx[0], idx[-1]

# Лінійна інтерполяція лівого краю

if left > 0:
    y0, y1 = segment[left - 1], segment[left]
    if y1 != y0:
        left_f = (left - 1) + (half - y0) / (y1 - y0)
    else:
        left_f = float(left)
else:
    left_f = float(left)

# Лінійна інтерполяція правого краю
```

```
if right < segment.size - 1:
    y0, y1 = segment[right], segment[right + 1]
    if y1 != y0:
        right_f = right + (half - y0) / (y1 - y0)
    else:
        right_f = float(right)
else:
    right_f = float(right)
return max(0.0, right_f - left_f)
```

ДОДАТОК Б

ТЕХНІЧНІ ХАРАКТЕРИСТИКИ ЦИФРОВАЙЗЕРА CAEN DT5761

У додатку наведено повний перелік технічних характеристик цифровайзера CAEN DT5761, на якому базується апаратна частина розробленого комплексу. Дані взято з офіційної документації виробника [4].

Параметр	Значення
Кількість каналів	1
Частота дискретизації	4 GS/s (250 пс/відлік)
Розрядність АЦП	10 бітів (1024 рівні)
Аналогова смуга пропускання	1.5 ГГц (3 дБ)
Вхідний динамічний діапазон	1 Vpp (±0.5 В)
Вхідний імпеданс	50 Ом
Тип вхідного з’єднання	DC
Зміщення нуля	16-bit DAC, повний діапазон
Розрядність DAC зміщення нуля	16 бітів
Об’єм буферної пам'яті каналу	~ 8 MSamples
Максимальна тривалість запису	~ 2 мс
Інтерфейс підключення	USB 2.0
Тригерні входи	TRG-IN (TTL), CLK-IN
Тригерні виходи	GPO, S-IN
Світлодіодна індикація	PWR, USB, TRG, BUSY

Живлення	Зовнішній адаптер +12 В
Споживана потужність	~ 15 Вт
Габарити корпусу	154 × 50 × 164 мм
Маса	близько 1.0 кг
Робочий температурний діапазон	від 0 до +50 °C

Архітектурно перетворювач DT5761 побудований за принципом паралельного перетворення на основі масиву компараторів з пріоритетним шифратором, що забезпечує час одного перетворення, обмежений лише власною швидкодією компонентів аналогової частини. Внутрішня пам'ять плати організована за принципом кільцевого буфера з можливістю налаштовуваного співвідношення передісторії і післяісторії відносно моменту тригера.

Стек програмного забезпечення для роботи з платою включає чотири рівні: системний драйвер USB (CAENUSBdrvB на Windows), низькорівневу бібліотеку CAENVMELib, комунікаційну бібліотеку CAENComm, високорівневу прикладну бібліотеку CAENDigitizer. Усі бібліотеки постачаються виробником у вигляді скомпільованих DLL для мов сімейства C і C++. Для зв’язку з Python-кодом розробленого комплексу використовується стандартний модуль ctypes, як описано у підрозділах 3.2.