Санкт-Петербургский государственный университет

ЕФИМОВ Максим Русланович

Выпускная квалификационная работа

Драйвер высокоскоростного цифрового осциллографа в АСУ TANGO ускорительного комплекса NICA

Уровень образования: *Магистратура*

Направление 03.04.01 *“Прикладные математика и физика”*

Основная образовательная программа

ВМ.5896* *“Информационные и ядерные технологии”*

Научный руководитель:
доцент, кафедры ИиЯТ ПМ-ПУ,
кандидат физ.-мат. наук
Сидорин А.О.

Рецензент:
гл. инженер Базовой установки
Нуклотрон ЛФВЭ ОИЯИ,
доктор физ.- мат. наук, профессор
Сыресин Е.М

Санкт-Петербург
2019

Содержание

Введение

Объединенный институт ядерных исследований — международная межправительственная организация, являющая собой интеграцию фундаментальных теоретических и экспериментальных исследований с разработкой и применением новейших технологий и университетским образованием. Институт создан в целях объединения усилий, научного и материального потенциала государств-членов для изучения фундаментальных свойств материи. Основными направлениями в исследованиях ОИЯИ являются: фиизка элементарных частиц, ядерная физика и физика конденсированных сред [1].

NICA (Nuclotron based Ion Collider fAcility) – это ускорительный комплекс, создающийся на базе ОИЯИ с целью изучения свойств плотной барионной материи. После ввода коллайдера NICA учёные института планируют воссоздать в лабораторных условиях состояние вещества, в котором пребывала наша Вселенная первые мгновения после Большого Взрыва, так называемую кварк-глюонную плазму (КГП). Строительство было начато в 2013 году, ввод в эксплуатацию планируется в 2022 [2].

Комплекс NICA содержит обширный список устройств которыми необходимо управлять и объектов за которыми необходимо следить. Любые новые устройства необходимо интегрировать в общую систему управления Tango (TAco Next Generation Objects) Controls для оптимизации и унификации процессов управления комплексом. В перечень таких устройств входят и осциллографы.

Постановка задачи

Целью работы является интегрирование семейства устройств PicoScope 6000 Series в систему управления Tango комплекса NICA.

Для достижения цели были поставлены следующие задачи:

- Изучить систему управления Tango Controls.
- Изучить алгоритм написания приложения для Tango Controls с использованием внутренних утилит.
- Изучить SDK (набор средств разработки) для осциллографов PicoScope 6000 Series.
- На основе SDK для осциллографов PicoScope 6000 Series написать класс для работы с данными устройствами на языке программирования C++.
- Написанный класс интегрировать в систему управления Tango Controls и протестировать для дальнейшей работы устройств PicoScope 6000 Series в автоматической системе управления комплекса NICA.

Обзор литературы

Система управления Tango Controls имеет широкое распространение в научных комплексах во всем мире, таких как SOLELIL во Франции, DESY в Германии, SKAO в Великобритании и многих других, что описано на официальном сайте [3]. На сайте также приведен список преимуществ данной системы: это бесплатный открытый устройство-ориентированный набор инструментов для управления устройствами и софтом любого типа и построения программного пакета для обеспечения работы в реальном времени систем сбора, обработки, отображения и архивирования информации о наблюдаемом или управляемом объекте. Tango Controls это независимая система и поддерживает C++, Java и Python для всех компонентов.

Сообщество Tango Controls активно ведет свою деятельность, проводятся различные встречи, заседания, доклады. На 32-м съезде Tango сообщества был представлен доклад «JINR NICA beam transfer channels CS», где сотрудниками ОИЯИ были представлены сведения об использовании системы Tango Controls в управлении ускорительным комплексом NICA. Отмечается, что система управления Tango имеет множество преимуществ, благодаря которым и получила такое распространение. Среди них быстрая установка и развертка, хорошая документация, активное сообщество, удобная с точки зрения программирования структура, поддержка структурированных данных и высокая производительность [4].

Введение системы Tango в комплекс NICA начиналось с Нуклотрона. Несколько подсистем были переведены в структуры на основе Tango, что описано в статье [5]. Одной из таких подсистем была система медленного вывода пучка Нуклотрона. Для управления данной системой было написано программное обеспечение в системе Tango, описанный в статье [6].

Дальнейший этап интегрирования системы Tango описывается в статье [7]. В данной статье описывается структура, как «клиентской», так и «серверной» частей написанных в Tango программ и управления ими на примере системы управления комплекса NICA.

Осциллографы серии PicoScope 6000 производства Pico Technology это семейство компактных устройств с высокой производительностью использующиеся в комплексе NICA. С пропускной способностью от 250 МГц до 500 МГц в режиме сбора данных в реальном времени достигают разрешения в 5 гигасемплов в секунду с точностью отображения данных в 200 пикосекунд. С помощью дополнительных настроек можно увеличить разрешение до 50 гигасемплов в секунду, что эквивалентно точности отображения данных в 20 пикосекунд. Каждый осциллограф серии PicoScope 6000 Series оснащен 4 каналами сбора данных, интерфейсом USB 3.0 и поддержкой любой из платформ: Windows, Mac, Linux. Данная информация находится на официальном сайте Pico Technology [8].

Глава 1. Разработка приложений в системе Tango Controls

Каждое устройство в Tango имеет уникальный идентификатор из трех частей домен, семейство, элемент, а также состояние и статус. Интерфейс состоит из свойств, задающихся в базе данных, из набора атрибутов, команд и пайпов. Для разработки в системе Tango необходимо последовательно воспользоваться утилитами из набора, а сам код пишется в выбранной среде разработки.

В данной работе была поставлена задача написания драйвера для высокоскоростного осциллографа на языке программирования C++. Для решения данной задачи были использованные следующие утилиты из программного пакета Tango: Pogo, Jive, AtkPanel, Astor [9].

1.1 Утилита Pogo

Утилита Pogo служит для генерации шаблона исходного кода по указанному интерфейсу Tango-устройства (Рис. 1). В редакторе можно добавить все необходимые атрибуты, команды и т. д. (Рис. 2).

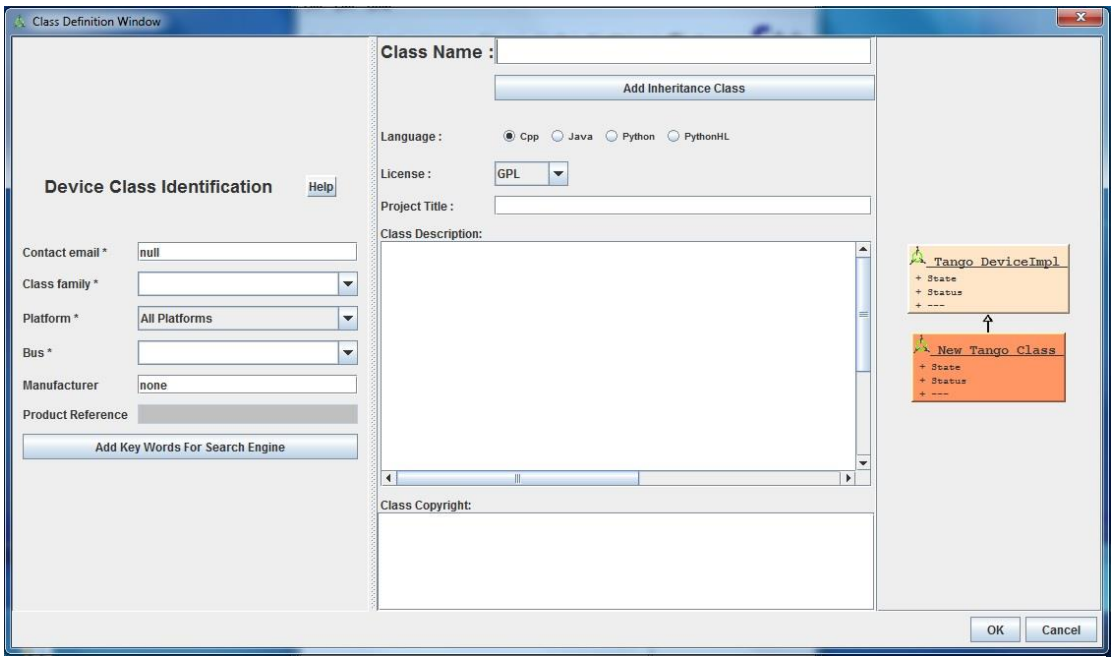

Рис. 1 Окно создания устройства в Pogo

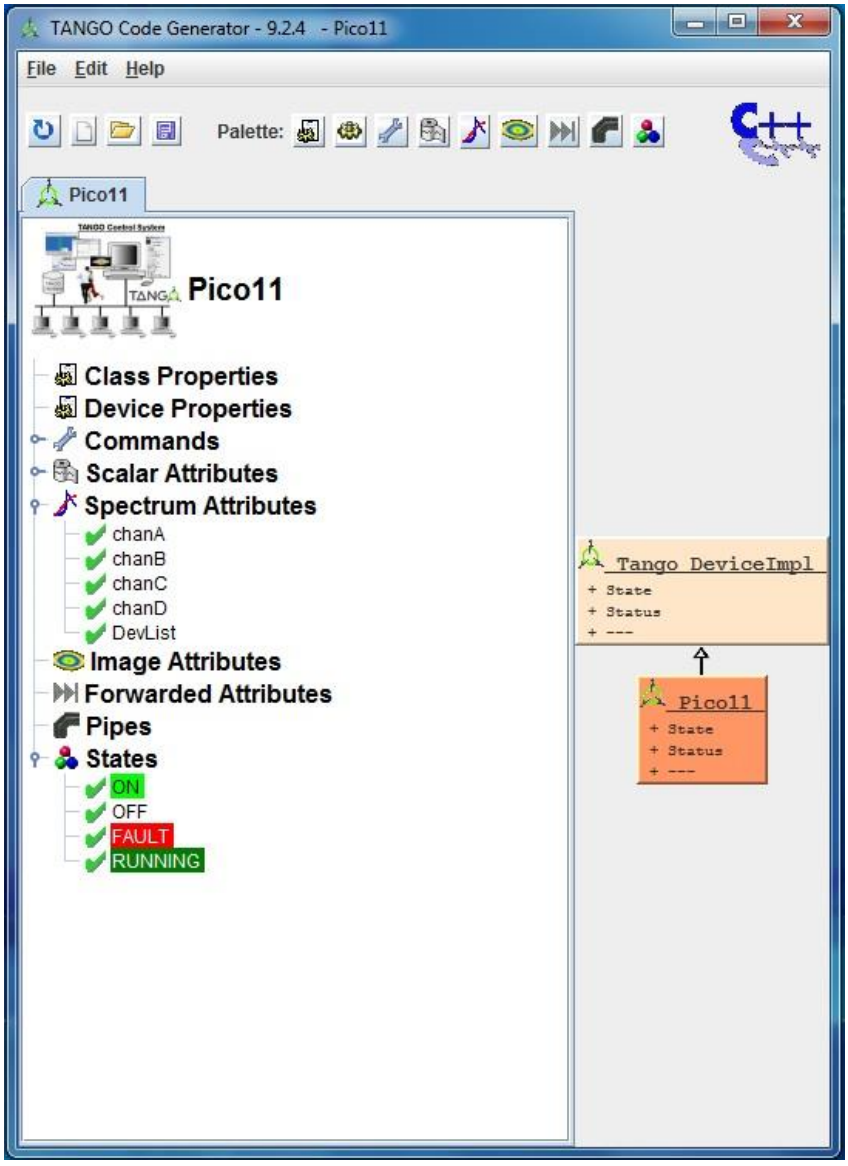

Рис. 2 Окно настройки параметров устройства в Pogo

После чего генерируется шаблон будущей программы, с которым уже можно работать в любой удобной среде. В шаблоне присутствуют защищенные области, куда и следует писать будущий код. В таком случае, при необходимости изменить структуру шаблона в Pogo, уже написанный код потерян не будет (Рис. 3).

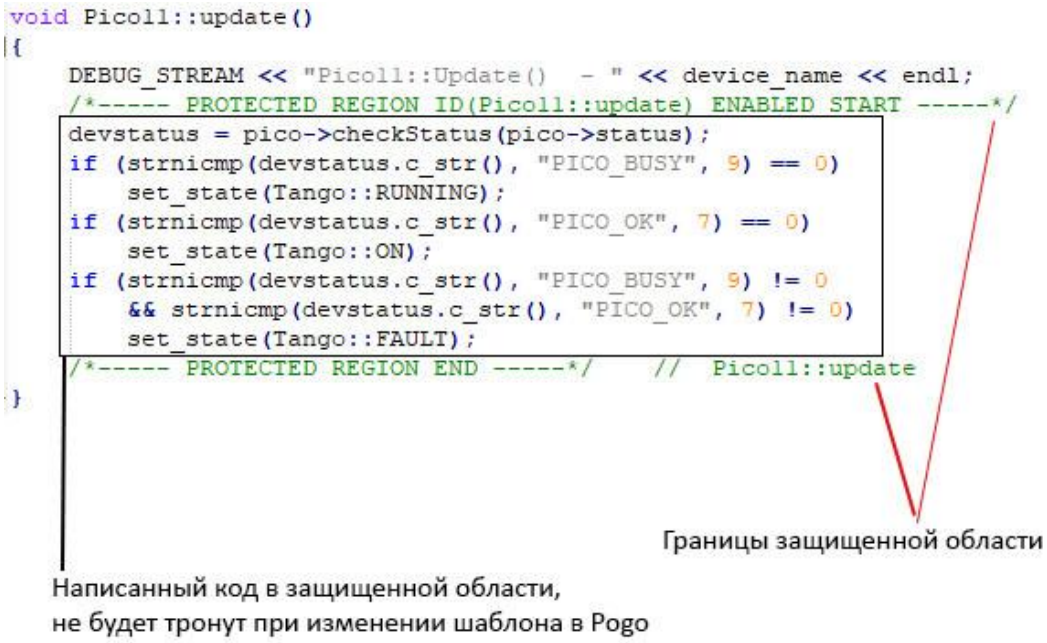

Рис. 3 Сгенерированный шаблон в Pogo

1.2 Утилита Jive

Это утилита для отображения и работы с базой данных устройств и редактирования некоторых свойств этих устройств. Например, при помощи данной утилиты добавляют новые устройства в базу данных Tango и вызывают утилиты для управления ими (Рис. 4).

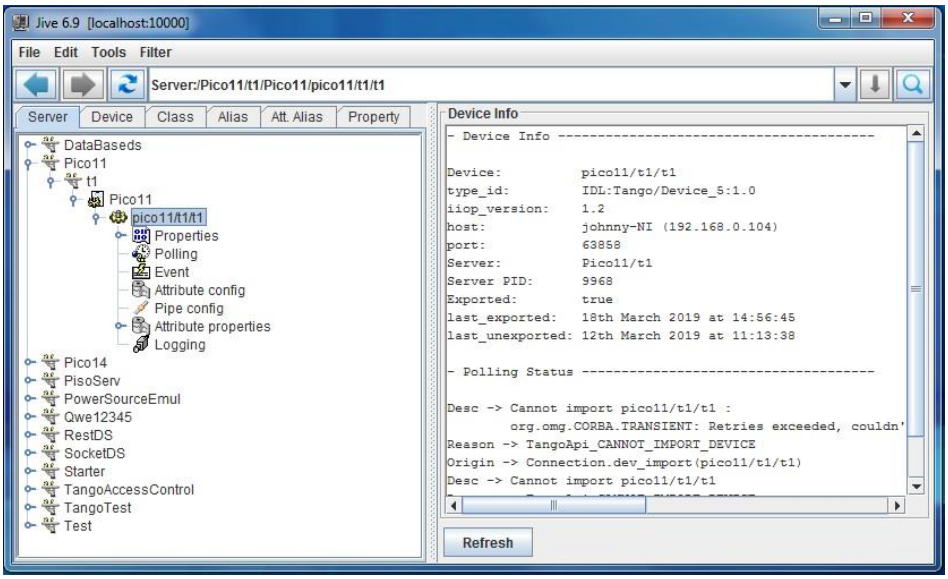

Рис. 4 Интерфейс Jive

1.3 Утилита AtkPanel

Из Jive можно вызвать AtkPanel – утилита, представляющая собой базовый интерфейс для устройств в базе данных. Поддерживается большинство типов данных из Tango. Используя AtkPanel можно просматривать и устанавливать значения атрибутов, выполнять команды, тестировать устройства и получать доступ к данным диагностики. Использование утилиты подробно описано в Главе 4.

1.4 Утилита Astor

Astor – утилита, целью которой является возможность быстро определить все ли хорошо в контролируемой системе, и если нет, то иметь возможность диагностировать проблему и решить ее. На ряду с этим возможно выполнять настройки контролируемой системы и ее компонентов, устанавливать последовательность запуска этих компонентов (Рис. 5).

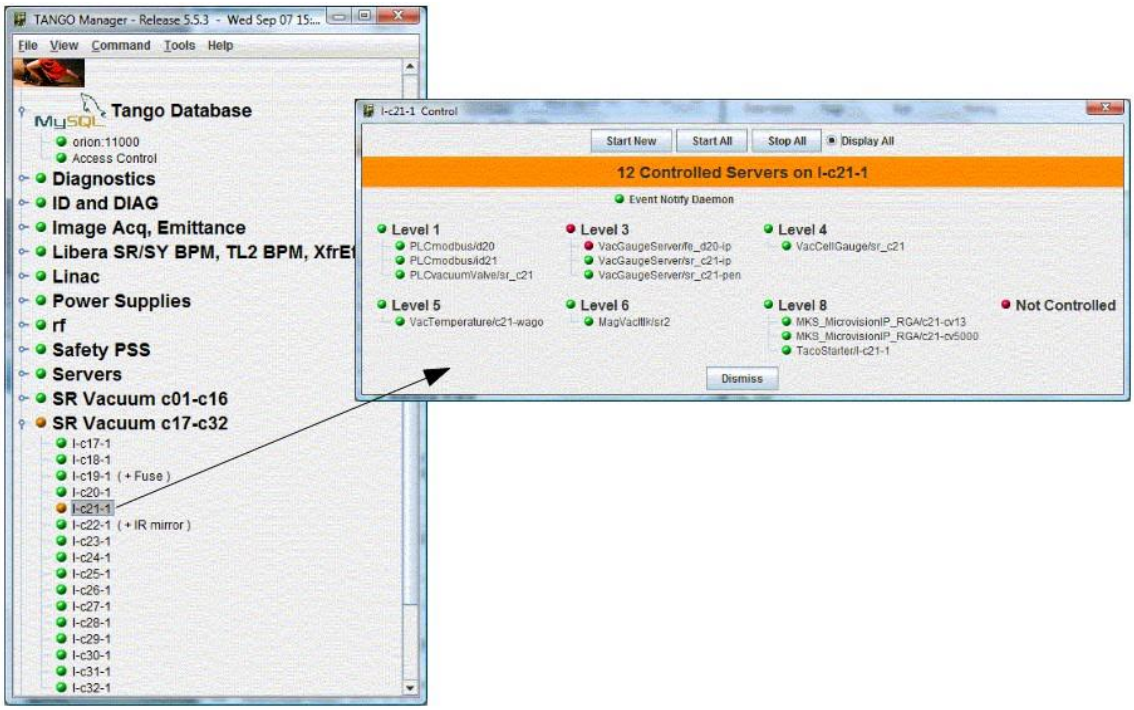

Рис. 5 Интерфейс Astor

Глава 2. Структура кода класса драйвера

Процесс написание драйвера для осциллографов PicoScope 6000 Series начинается с генерации шаблона будущей программы в Pogo, где мы задаем необходимые для работы функции такие как: включить, выключить, задать триггер, начать сбор данный, закончить сбор данных, сброс и другие, атрибуты, некоторые из которых отвечают за выбор устройства, пороги срабатывания триггера, смещение, задержку и так далее.

Когда шаблон сгенерирован, с ним можно работать в Visual Studio. Вначале подключено SDK для работы с осциллографами PicoScope 6000 Series, которое включает в себя необходимые для работы драйвера методы.

Для удобства разработки был создан класс, в котором реализованы все методы, которые в дальнейшем вызываются в Tango части драйвера [10].

2.1 Функции класса драйвера

В классе драйвера реализованы следующие функции:

– Инициализация

При запуске драйвер записывает базовые значения в переменные. Далее сканирует подключенные устройства PicoScope 6000 Series.

– Формирование списка подключенных устройств и выбор определенного устройства для работы

После завершения инициализации драйвера и поиска подключенных устройств формируется список этих устройств для дальнейшей работы с ними. Получив список устройств с индивидуальными номерами реализована возможность выбора конкретного устройства.

– Включение и выключение устройства

Имея возможность найти подключенные устройства и выбрать конкретное для работы был разработан базовый функционал, такой как включение и выключение.

– Потоковый сбор данных

Сбор данных потоком реализуется в несколько этапов:

- Задаются буферы для данных с каждого канала.
- Запускается сбор данных.
- Считывание данных из заданных выше буферов.
- Получив данные перезаписываем их в память для дальнейших операций.
- Останавливается сбор данных путем установки соответствующего флага в значение FALSE.

– Сбор блока данных

В режиме сбора блока осциллограф собирает блок данных во внутреннюю память. Когда осциллограф собрал весь блок, он сообщает что готов передать весь блок в память компьютера.

- Задаются буферы данных.
- Запускается сбор блока данных.
- Выполняется проверка на готовность к передаче.
- Как только данные готовы к передаче, перезаписываем их в память для дальнейшей работы.

– Включение и выключение каналов осциллографа

В устройствах PicoScope 6000 Series есть возможность выключения каналов и включения с определенными параметрами: тип получаемого сигнала, диапазон сигнала и пропускная способность канала.

– Установка настроек триггера

Осциллографы серии PicoScope 6000 Series имеют обширные возможности по настройке триггера.

В начале задаются свойства каналов осциллографа.

Далее задаются условия для триггера.

Следующим этапом установки триггера является задание направления сработки для каждого канала.

После есть возможность задать отсрочку.

– Сброс настроек

Для удобства использования драйвера удобно иметь возможность сбросить все настройки к стартовым.

– Выбор единиц измерения

Реализована возможность получения итоговых данных в милливольтах и отсчетах АЦП.

2.2 Вывод данных для Web интерфейса

Данный драйвер изначально разрабатывался под Web интерфейс, что имеет некоторые ограничения. Для оптимизации работы было реализовано преобразование данных к выводу и вывод в специально подготовленный поток данных.

Во время работы осциллографа генерируется колоссальный объем данных. Передача и отображение такого количества точек может быть затруднительна для клиента. Для такого случая реализовано прореживания методом Наибольшего Треугольника Трех Корзин. Данный метод заключается в разделении данных на блоки примерно равных размеров. Однако, первый и последний блоки содержат первую и последнюю точки соответственно. Следующим шагом из каждого блока выбирается одна точка. Метод работает с тремя блоками одновременно слева направо. Первая точка, формирующая

левый угол треугольника всегда зафиксирована, как уже выбранная ранее, средняя будет сейчас выбрана, конечная точка выбирается как среднее всех точек в блоке. В большинстве случаев такой выбор конечной точки не хуже перебора, но намного эффективнее (Рис. 6).

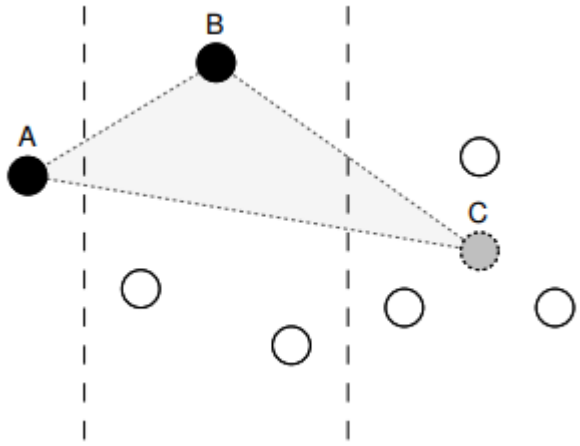

Рис. 6 Наибольший треугольник, сформированный из точек, выбранных алгоритмом

Средняя точка выбирается путем оценки площади треугольника для каждой из точек с учетом предыдущей уже выбранной и следующей средней. В итоге берется точка, образующая наибольшую площадь [11].

В драйвере реализована возможность выбора количество точек, к которому приведет оптимизация для улучшения точности, либо скорости работы, путем выбора оптимального для отображения количества точек (Рис. 7).

```
for (int i = 0; i < threshold - 2; i++) {
    // Calculating point average for next bucket (containing c)
    double avgX = 0;
    double avgY = 0;
    int avgRangeStart = (int)floor((double)((i + 1) * every) + 1);
    int avgRangeEnd = (int)floor((double)((i + 2) * every) + 1);
    avgRangeEnd = (avgRangeEnd < dataLength) ? avgRangeEnd : dataLength;

    int avgRangeLength = avgRangeEnd - avgRangeStart;

    for (; avgRangeStart < avgRangeEnd; avgRangeStart++) {
        avgX += BuffData.at(avgRangeStart);
        avgY += BuffTime.at(avgRangeStart);
    }
    avgX /= avgRangeLength;
    avgY /= avgRangeLength;

    int rangeOffs = (int)floor((double)((i + 0) * every) + 1);
    int rangeTo = (int)floor((double)((i + 1) * every) + 1);

    // Point a
    double pointAX = BuffData.at(a) * 1;
    double pointAY = BuffTime.at(a) * 1;

    double maxArea = -1;
    for (; rangeOffs < rangeTo; rangeOffs++) {
        // Calculate triangele area over three buckets
        double area = abs(
            (pointAX - avgX) * (BuffTime.at(rangeOffs) - pointAY) -
            (pointAX - BuffData.at(rangeOffs)) * (avgY - pointAY)
        ) * 0.5;
        if (area > maxArea) {
            maxArea = area;
            maxAreaPoint.first = BuffData.at(rangeOffs);
            maxAreaPoint.second = BuffTime.at(rangeOffs);
            nextA = rangeOffs; // Next a is this b
        }
    }
    datasampled.push_back(maxAreaPoint.first);
    timesampled.push_back(maxAreaPoint.second);
    a = nextA;
}
```

Рис. 7 Реализация метода Наибольшего Треугольника Трех Корзин

Глава 3. Интеграция класса в систему Tango Controls

Получившийся класс для работы с осциллографами PicoScope 6000 Series интегрируется в систему Tango Controls:

– Инициализация Tango устройства

У каждой команды, сгенерированной в Pogo, есть функция, срабатывающая при запуске, в ней и вызывается класс драйвера, а также задаются некоторые параметры по умолчанию.

– Отображение подключенных устройств в Tango

Получив список устройств в классе, необходимо их записать в Tango атрибут для дальнейшего отображения.

– Включение и выключение в Tango

Для включения и выключения осциллографа вызывается соответствующая функция из класса.

– Команды для потокового сбора данных

Для начала потокового сбора данных вызывается соответствующая команда из класса.

Остановка потокового сбора данных происходит путем установки флага в значение FALSE.

– Запуск сбора блока данных

Чтобы запустить сбор блока данных необходимо вызвать соответствующую функцию из класса.

– Получение прореженных данных

Сбор блока данных и потоковый сбор данных заполняют память, однако перед выводом необходимо проредить эти данные для оптимальной скорости работы драйвера, количество получаемых точек можно задать. Прореживание реализовано методом Наибольшего Треугольника Трех Корзин, что позволяет значительно снизить объем отправляемых данных, но сохранив максимально

возможную точность отображаемых данных. Так же для каждой точки присваивается временная метка для возможности навигации по данным.

– Задание параметров триггера

Для каждого из каналов задаются параметры триггера.

После задания параметров для каналов триггер можно активировать соответствующим методом.

– Отслеживание ошибок

В процессе взаимодействия с осциллографами PicoScope 6000 Series устройство возвращает свой статус на каждое действие. Следовательно, для отслеживания состояния и получения ошибок необходимо отслеживать этот статус. В драйвере реализована данная функция, отображающая статус с периодичностью 0.1 секунды.

Глава 4. Методика тестирования и отладки кода

В данной работе для тестирования работы и отладки команд и атрибутов разрабатываемого приложения использовалась утилита AtkPanel (Рис. 8).

В процессе написания драйвера, при добавлении новой команды или атрибута в утилите AtkPanel проверялась работоспособность данной команды или атрибута путем ее использования и отслеживания ошибок. При отсутствии

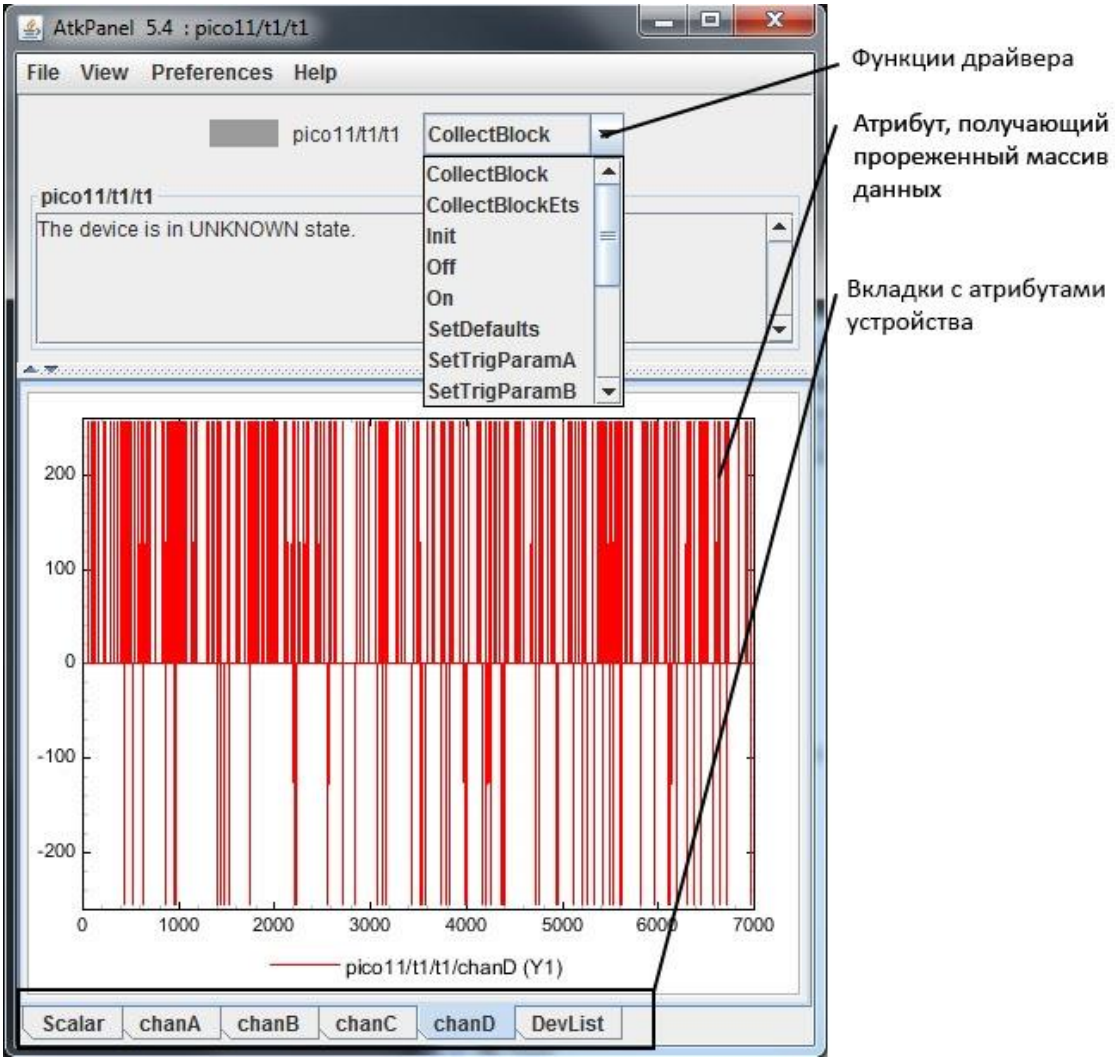

Рис. 8 Интерфейс AtkPanel во время тестирования

каких-либо ошибок во время отладки и работе в соответствии с требованиями, команда или атрибут считаются исправными.

Выводы

В процессе данной работы изучена многоуровневая система управления Tango Controls для ускорительного комплекса NICA. На примере написания драйвера для семейства высокоскоростных осциллографов PicoScope 6000 Series освоен процесс написания сервера устройства в системе Tango Controls с использованием внутренних утилит: Pogo, Jive, AtkPanel, Astor. Pogo используется для генерации шаблона будущей программы, Jive участвует в работе с базой данных устройств, а AtkPanel помогает во время отладки разрабатываемой программы, Astor необходим для общего контроля за системой, например, задания последовательности активации подсистем в общей схеме.

Было освоено SDK для устройств семейства PicoScope 6000 Series и написан класс для работы с осциллографами. В классе реализован различный функционал, который позволят пользователю полноценно пользоваться устройством. Реализованные функции включают в себя корректный запуск SDK, сканирование подключенных устройств и формирование их серийных номеров в список, включение и выключение конкретных устройств, включение и выключение потокового сбора данных в память, сбор блока данных в память, включение и выключения различных каналов устройства, включение и настройки триггера, сброс настроек к стандартным, а так же возможность выбора единиц измерения данных и вывод данных.

Драйвер разрабатывался для работы в паре с Web интерфейсом. Отображение данных в Web имеет ограничения: своевременно отображать большое количество данных затруднительно. Для решения данной проблемы было реализовано прореживание данных методом Наибольшего Треугольника Трех Корзин в целях оптимизации работы. Таким образом пользователь может

выбрать разрешение получаемых данных с максимально возможным сохранением точности.

В заключительной части работы написанный ранее класс для работы с устройствами PicoScope 6000 Series был интегрирован в систему Tango, путем добавления класса в генерируемый Pogo шаблон и реализации работы методов класса из команд и атрибутов Tango устройства. В конечном счете был получен и протестирован драйвер для осциллографов PicoScope 6000 Series в системе управления Tango ускорительного комплекса NICA. Реализация основных частей драйвера представлена в приложении.

Заключение

- Изучена система управления Tango Controls.
- Изучен алгоритм написания приложения для Tango Controls с использованием внутренних утилит.
- Изучено SDK для осциллографов PicoScope 6000 Series.
- На основе SDK для осциллографов PicoScope 6000 Series написан класс для работы с данными устройствами.
- Написанный класс интегрирован в систему управления Tango Controls и протестирован для дальнейшей работы устройств PicoScope 6000 Series в автоматической системе управления комплекса NICA.

Данная работа была представлена на 23-ей Международной научной конференции молодых ученых и специалистов ОИЯИ AYSS-2019 [12].

Список литературы

1. Объединенный институт ядерных исследований | Объединенный институт ядерных исследований. http://www.jinr.ru/about
2. NICA - Nuclotron-based Ion Collider fAcility. http://nica.jinr.ru/
3. Home – TANGO Controls. https://www.tango-controls.org/
4. 32nd TANGO Collaboration Meeting (5-7 June 2018). https://indico.eli-beams.eu/event/310/session/5/contribution/97
5. Gorbachev E.V., Sedykh G.S. The equipment database for the control system of the NICA accelerator complex // Proceedings of ICALEPCS2013. – San Francisco, CA, USA, 2013, P. 1111-1113.
6. Andreev V. A., Volkov V. I., Gorbachev E. V., Isadov V. A., Kirichenko A. E., Romanov S. V., Sedykh G. S. TANGO standard software to control the Nuclotron beam slow extraction // Physics of Particles and Nuclei Letters, 2016. P. 605-608.
7. Gorbachev E. V., Andreev V. A., Kirichenko A. E., Monakhov D. V., Romanov S. V., Rukoyatkina T. V., Sedykh G. S., Volkov V. I. NUCLOTRON AND NICA CONTROL SYSTEM DEVELOPMENT STATUS // Proceedings of ICALEPCS2015, Melbourne, Australia, 2015, P. 437-440.
8. PicoScope 6000 – high performance USB scope | Pico Technology. https://www.picotech.com/oscilloscope/6000/picoscope-6000-overview
9. Developer’s Guide – Tango Controls 9.2.5 documentation. https://tango-controls.readthedocs.io/en/latest/development/index.html
10. PicoScope 6000 Series Programmer's Guide. https://www.picotech.com/download/manuals/picoscope-6000-series-programmers-guide.pdf

11. Steinarsson S. Downsampling Time Series for Visual Representation. Iceland: Faculty of Industrial Engineering, Mechanical Engineering and Computer Science University of Iceland, 2013. 65c.
12. The XXIII International Scientific Conference of Young Scientists and Specialists (AYSS-2019) (15-19 April 2019). https://indico.jinr.ru/event/756/session/4/contribution/365

Приложение

Реализация основных функций драйвера:

1. Код функции сканирования подключенных устройств.

```
do
{
    memset(&(allUnits[devCount]), 0, sizeof(UNIT));
    status = OpenDevice(&(allUnits[devCount]), NULL);
    if (status == PICO_OK || status == PICO_USB3_0_DEVICE_NON_USB3_0_PORT)
    {
        allUnits[devCount++].openStatus = status;
    }

} while (status != PICO_NOT_FOUND);

if (devCount == 0)
{
    ss << "Picoscope devices not found\n";
}
else
{
    for (listIter = 0; listIter < devCount; listIter++)
    {
        if ((allUnits[listIter].openStatus == PICO_OK ||
            allUnits[listIter].openStatus == PICO_USB3_0_DEVICE_NON_USB3_0_PORT))
        {
            set_info(&allUnits[listIter]);
            openIter++;
        }
    }
    Devices.clear();
    Devices = enumer(openIter, devCount);
}
```

2. Код функции формирования списка подключенных устройств.

```
std::vector<std::string> p6000::enumer(uint16_t openIter, uint16_t devCount)
{
    uint16_t listIter = 0;
    std::string PDev;
    char PPDev[10];
    std::vector<std::string> PicoDev;

    int8_t devChars[] = "1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz#";
    for (listIter = 0; listIter < devCount; listIter++)
    {
        PDev = ((char*)allUnits[listIter].serial);
        PicoDev.push_back(PDev);
    }
    return PicoDev;
}
```

3. Код функции выбора устройства.

```
std::vector<std::string> p6000::enumer(uint16_t openIter, uint16_t devCount)
{
    uint16_t listIter = 0;
    std::string PDev;
    char PPDev[10];
    std::vector<std::string> PicoDev;

    int8_t devChars[] = "1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz#";
    for (listIter = 0; listIter < devCount; listIter++)
    {
        PDev = ((char*)allUnits[listIter].serial);
        PicoDev.push_back(PDev);
    }
    return PicoDev;
}
```

4. Код функций включений и выключения.

```
void p6000::off()
{
    CloseDevice(&allUnits[ChoosenDev]);
}

void p6000::on()
{
    status = OpenDevice(&(allUnits[ChoosenDev]), NULL);
}
```

5. Код функции, записывающей прореженные данные для Web интерфейса.

```
std::string p6000::VTS(std::vector<double> timeS,
    std::vector<double> chanA, std::vector<double> chanB,
    std::vector<double> chanC, std::vector<double> chanD)
{
    std::stringstream ssl;
    ssl << "[";
    for (int i = 0; i < timeS.size(); ++i)
    {
        if (i > 0)
            ssl << ", ";
        ssl << "{";
        ssl << "\"ts\": " << std::to_string(timeS.at(i)) << ", ";
        ssl << "\"cha\": " << chanA.at(i) << ", ";
        ssl << "\"chb\": " << chanB.at(i) << ", ";
        ssl << "\"chc\": " << chanC.at(i) << ", ";
        ssl << "\"chd\": " << chanD.at(i);
        ssl << "}";
    }
    ssl << "]";
    return ssl.str();
}
```

6. Код функций преобразующих формат данных.

```
int16_t p6000::mv_to_adc(int16_t mv, int16_t ch)
{
    return (mv * PS6000_MAX_VALUE) / inputRanges[ch];
}

int32_t p6000::adc_to_mv(int32_t raw, int32_t ch)
{
    return (raw * inputRanges[ch]) / PS6000_MAX_VALUE;
}
```

7. Код выделения буферов для записи данных.

```
for (i = PS6000_CHANNEL_A; (int32_t)i < unit->channelCount; i++)
{
    if (unit->channelSettings[i].enabled)
    {
        buffers[i * 2] = (int16_t*)calloc(sampleCount, sizeof(int16_t));
        buffers[i * 2 + 1] = (int16_t*)calloc(sampleCount, sizeof(int16_t));

        status = ps6000SetDataBuffers(unit->handle, (PS6000_CHANNEL)i, buffers[i * 2],
            buffers[i * 2 + 1], sampleCount, PS6000_RATIO_MODE_NONE);
        checkStatus(status);

        appBuffers[i * 2] = (int16_t*)calloc(sampleCount, sizeof(int16_t));
        appBuffers[i * 2 + 1] = (int16_t*)calloc(sampleCount, sizeof(int16_t));
    }
}
```

8. Код сброса настроек.

```
void p6000::SetDefaults(UNIT * unit)
{
    int32_t i;

    status = ps6000SetEts(unit->handle, PS6000_ETS_OFF, 0, 0, NULL);

    for (i = 0; i < unit->channelCount; i++)
    {
        status = ps6000SetChannel(unit->handle, (PS6000_CHANNEL)(PS6000_CHANNEL_A + i),
            unit->channelSettings[PS6000_CHANNEL_A + i].enabled,
            (PS6000_COUPLING)unit->channelSettings[PS6000_CHANNEL_A + i].DCcoupled,
            (PS6000_RANGE)unit->channelSettings[PS6000_CHANNEL_A + i].range, 0, PS6000_BW_FULL);

        IsChanOn[i] == unit->channelSettings[PS6000_CHANNEL_A + i].enabled;
    }
}
```

9. Код инициализации Tango драйвера.

```
/*----- PROTECTED REGION ID(Picoll::init_device) ENABLED START -----*/
pico = new p6000();

*attr_TimeStamp_read = 0;
*attr_ChanMaxVolt_read = 0;
*attr_ChanMinVolt_read = 0;
hhh = pico->Devices.size();
pico->ADCorMV = 1;
sampleNum = 1000;
attr_DevStatus_read = &the_strdata;
attr_DataSt_read = &the_str;
attr_ChanCh_read = &the_chcheck;
attr_DevList_read = strDev;

pico->BusyReadingDataA = FALSE;
pico->BusyReadingDataB = FALSE;
pico->BusyReadingDataC = FALSE;
pico->BusyReadingDataD = FALSE;
pico->BusyCopyDataA = FALSE;
pico->BusyCopyDataB = FALSE;
pico->BusyCopyDataC = FALSE;
pico->BusyCopyDataD = FALSE;
/*----- PROTECTED REGION END -----*/    //  Picoll::init_device
```

10.Код выбора устройства в Tango.

```
void Picoll::write_DevChoose(Tango::WAttribute &attr)
{
    DEBUG_STREAM << "Picoll::write_DevChoose(Tango::WAttribute &attr) entering... " << endl;
    Tango::DevDouble    w_val;
    attr.get_write_value(w_val);
    /*----- PROTECTED REGION ID(Picoll::write_DevChoose) ENABLED START -----*/
    if ((int)(w_val - 1) < hhh  && (int)(w_val - 1) > -1)
    {
        pico->pick_device((int)(w_val - 1));
        CurDev = w_val - 1;
    }
    /*----- PROTECTED REGION END -----*/    //  Picoll::write_DevChoose
}
```

11.Код отслеживания статуса устройства в Tango.

```
void Picoll::update()
{
    DEBUG_STREAM << "Picoll::Update()  - " << device_name << endl;
    /*----- PROTECTED REGION ID(Picoll::update) ENABLED START -----*/
    devstatus = pico->checkStatus(pico->status);
    if (strnicmp(devstatus.c_str(), "PICO_BUSY", 9) == 0)
        set_state(Tango::RUNNING);
    if (strnicmp(devstatus.c_str(), "PICO_OK", 7) == 0)
        set_state(Tango::ON);
    if (strnicmp(devstatus.c_str(), "PICO_BUSY", 9) != 0
        && strnicmp(devstatus.c_str(), "PICO_OK", 7) != 0)
        set_state(Tango::FAULT);
    /*----- PROTECTED REGION END -----*/    //  Picoll::update
}
```

12. Код Tango получения прореженных данных в формате оптимизированном для Web интерфейса.

```
Tango::DevString Picoll::get_data()
{
    Tango::DevString argout;
    DEBUG_STREAM << "Picoll::GetData()  - " << device_name << endl;
    /*----- PROTECTED REGION ID(Picoll::get_data) ENABLED START -----*/
    chanA.clear();
    pico->DowSam(pico->chanAT, sampleNum);
    for (int gg = 0; gg < sampleNum; gg++)
    {
        chanA.push_back(pico->datasampled[gg]);
    }
    chanB.clear();
    pico->DowSam(pico->chanBT, sampleNum);
    for (int gg = 0; gg < sampleNum; gg++)
    {
        chanB.push_back(pico->datasampled[gg]);
    }
    chanC.clear();
    pico->DowSam(pico->chanCT, sampleNum);
    for (int gg = 0; gg < sampleNum; gg++)
    {
        chanC.push_back(pico->datasampled[gg]);
    }
    chanD.clear();
    pico->DowSam(pico->chanDT, sampleNum);
    for (int gg = 0; gg < sampleNum; gg++)
    {
        chanD.push_back(pico->datasampled[gg]);
    }
    timeS.clear();
    timestep = ((pico->timesampled[sampleNum - 1]) - (pico->timesampled[0])) / sampleNum;
    if (timestep < 0.00001)
        timestep = 0.00001;
    for (int gg = 0; gg < sampleNum; gg++)
    {
        timeS.push_back((pico->timesampled[0]) + timestep * gg);
    }
    datadata = pico->VTS(timeS, chanA, chanB, chanC, chanD);
    argout = Tango::string_dup(datadata.c_str());
    /*----- PROTECTED REGION END -----*/    //  Picoll::get_data
    return argout;
}
```

13. Код атрибута в Tango отвечающего за выбор типа выводимых данных.

```
void Picoll::write_MVorADC(Tango::WAttribute &attr)
{
    DEBUG_STREAM << "Picoll::write_MVorADC(Tango::WAttribute &attr) entering... " << endl;
    //  Retrieve write value
    Tango::DevDouble    w_val;
    attr.get_write_value(w_val);
    /*----- PROTECTED REGION ID(Picoll::write_MVorADC) ENABLED START -----*/
    if (w_val == 0)
    {
        pico->ADCorMV = 0;
    }
    else
    {
        pico->ADCorMV = 1;
    }

    /*----- PROTECTED REGION END -----*/    //  Picoll::write_MVorADC
}
```

14. Код атрибута в Tango отвечающего за количество точек, к которому будут прорежены данные.

```
void Picoll::write_SampleNumber(Tango::WAttribute &attr)
{
    DEBUG_STREAM << "Picoll::write_SampleNumber(Tango::WAttribute &attr) entering... " << endl;
    //  Retrieve write value
    Tango::DevDouble    w_val;
    attr.get_write_value(w_val);
    /*----- PROTECTED REGION ID(Picoll::write_SampleNumber) ENABLED START -----*/
    sampleNum = w_val;
    /*----- PROTECTED REGION END -----*/    //  Picoll::write_SampleNumber
}
```